



## Deliverable - D3

### *Specification of optimized GALS interfaces and application scenarios*

<b>Grant Agreement No:</b>	214364
<b>Project acronym:</b>	GALAXY
<b>Project title:</b>	GALS InterfAce for CompleX Digital System Integration
<b>Funding Scheme:</b>	STREP
<b>Date of latest version of Annex I against which the assessment will be made:</b>	15.07.2008.
<b>Contractual Date of Delivery to the EC:</b>	31. Dec. 08
<b>Actual Date of Delivery to the EC:</b>	24. Dec. 08
<b>Author(s):</b>	Milos Krstic, Xin Fan, Miroslav Marinkovic (IHP), Frank Gürkaynak (EPFL), Christoph Heer, Sören Sonntag (INFINEON)
<b>Participant(s):</b>	IHP
<b>Work Package:</b>	WP2
<b>Security:</b>	Public
<b>Nature:</b>	Report
<b>Version:</b>	2
<b>Total number of pages:</b>	60

#### **Abstract:**

This report assembles the results from the activities of Work Package 2 "GALS Interface Evaluation and Application Scenario" of the GALAXY project.

In this report we are analyzing different types of GALS interfaces including FIFO-based, synchronizer-based, and GALS with pausable clocking. All three types of interfaces are modelled and evaluated with special focus on the best applications for each particular architecture. The other important parameters that are analyzed are hardware complexity, latency, throughput, power consumption and EMI profile.

Additionally, we have analyzed the hardware extension of the standard GALS interfaces in order to achieve compatibility to some of the existing standards for IP cores. As an example for the evaluation we have taken GALS wrapper for pausable clocking and proposed the architecture of the OCP-compliant adapter for this GALS interface. We are analyzing the effectiveness of building such an OCP adapter and the complexity of the required approach.

Finally, we have analyzed the possibilities for further optimization of GALS interfaces. This work has been focused on the optimization of pausable clocking schemes for bursty data transfers. In addition to that we have explored throughput improvement for systems with pausable clocking.

**Keyword list: GALS, asynchronous, interconnect, OCP, AMBA**



# GALAXY

GALS InterfAce for CompleX Digital  
SYstem Integration

Confid. Level: Public  
Date : 24/12/2008  
Issue: 2

<i>Function</i>	<i>Responsibility</i>	<i>Date</i>	<i>Signature</i>
<b>Written by:</b>	Milos Krstic	5.12.2008	
<b>Checked by:</b>	Lilian Janin	28.7.2008	
<b>Approved by:</b>			

Reserved to EC

<b>Approved by:</b>			
---------------------	--	--	--



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA



Never stop thinking



# GALAXY

GALS InterfAce for CompleX Digital  
SYstem Integration

Confid. Level: Public  
Date : 24/12/2008  
Issue: 2

---

## CHANGE RECORDS

<i>ISSUE</i>	<i>DATE</i>	<i>§ : CHANGE RECORD</i>	<i>AUTHOR</i>
1	24-July-08	First revision without data that are subject to patenting	M. Krstic, X. Fan, S. Sonntag
2	5-Dec-08	Second complete revision	M. Krstic, X. Fan, S. Sonntag



## BIBLIOGRAPHIC RECORD

Project Number:	214364 GALAXY
Project Title:	GALAXY
Deliverable Type:	Report
Deliverable Number:	D3
Contractual Date of Delivery:	31. Dec. 2008
Actual Date of Delivery:	24. Dec. 2008
Title of Deliverable:	Specification of optimized GALS interfaces and application scenarios
Work package contributing to the Deliverable:	WP2
Authors:	M. Krstic, X. Fan, M. Marinkovic, F. Gürkaynak, C. Heer, S. Sonntag
Abstract	<p>This report assembles the results from the activities of Work Package 2 "GALS Interface Evaluation and Application Scenario" of the GALAXY project.</p> <p>In this report we are analyzing different types of GALS interfaces including FIFO-based, synchronizer-based, and GALS with pausable clocking. All three types of interfaces are modelled and evaluated with special focus on the best applications for each particular architecture. The other important parameters that are analyzed are hardware complexity, latency, throughput, power consumption and EMI profile.</p> <p>Additionally, we have analyzed the hardware extension of the standard GALS interfaces in order to achieve compatibility to some of the existing standards for IP cores. As an example for the evaluation we have taken GALS wrapper for pausable clocking and proposed the architecture of the OCP-compliant adapter for this GALS interface. We are analyzing the effectiveness of building such an OCP adapter and the complexity of the required approach.</p> <p>Finally, we have analyzed the possibilities for further optimization of GALS interfaces. This work has been focused on the optimization of pausable clocking schemes for bursty data transfers. In addition to that we have explored throughput improvement for systems with pausable clocking.</p>
Keywords	GALS, asynchronous, interconnect, OCP, AMBA
Confidentiality Level	Public
Name of Client:	EC
Distribution List:	GALAXY, EC, internet
Authorised by:	Milos Krstic
Issue:	2
Document ID:	D3
Total Number of Pages:	60
Contact Details:	<a href="mailto:krstic@ihp-microelectronics.com">krstic@ihp-microelectronics.com</a>



## TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION .....</b>	<b>8</b>
<b>2</b>	<b>REFERENCES .....</b>	<b>9</b>
2.1	<b>ACRONYMS .....</b>	<b>9</b>
2.2	<b>REFERENCE DOCUMENTS .....</b>	<b>9</b>
<b>3</b>	<b>GALS SOLUTION ANALYSIS.....</b>	<b>11</b>
3.1	<b>GALS INTERFACE BASED ON SYNCHRONIZERS.....</b>	<b>11</b>
3.2	<b>FIFO-BASED GALS INTERFACES .....</b>	<b>14</b>
3.3	<b>GALS WRAPPER WITH PAUSIBLE CLOCKING .....</b>	<b>18</b>
3.4	<b>CROSS-COMPARISON OF DIFFERENT GALS INTERFACES AND INDUSTRIAL REQUIREMENTS .....</b>	<b>21</b>
<b>4</b>	<b>DEFINITION OF STANDARDIZED GALS INTERFACES.....</b>	<b>23</b>
4.1	<b>OCP ADAPTER FOR GALS SYSTEMS .....</b>	<b>24</b>
4.2	<b>REQUIRED SUBSET OF OCP FUNCTIONS.....</b>	<b>26</b>
4.3	<b>OCP ADAPTER FOR GALS SYSTEMS BASED ON PAUSIBLE CLOCKING.....</b>	<b>26</b>
4.4	<b>COMPLEXITY OF THE OCP ADAPTER .....</b>	<b>34</b>
4.5	<b>AMBA AXI GALS INTERFACE .....</b>	<b>35</b>
4.6	<b>AMBA AHP AND AMBA APB GALS INTERFACE .....</b>	<b>36</b>
<b>5</b>	<b>OPTIMIZATION OF THE GALS INTERFACES .....</b>	<b>37</b>
5.1	<b>IMPROVING GALS BASED ON THE PAUSIBLE CLOCKING.....</b>	<b>37</b>
5.2	<b>GALS FOR BURSTY DATA TRANSFER BASED ON THE CLOCK COUPLING ..</b>	<b>39</b>
5.3	<b>ANALYSIS AND MODIFICATION OF THE PAUSIBLE CLOCK GENERATOR FOR GALS SYSTEMS FOR MULTIPOINT APPLICATIONS.....</b>	<b>46</b>
5.4	<b>CONCLUSIONS.....</b>	<b>58</b>
	<b>APPENDIX .....</b>	<b>59</b>
A	<b>VHDL HEADER FOR GALS OCP ADAPTER .....</b>	<b>59</b>

## LIST OF FIGURES

Figure 1:	Block diagram of 2-Phase to 4-Phase Handshake Converter with Synchronization Logic.....	11
Figure 2:	Block diagram of 2-phase to 4-phase handshake converter.....	12
Figure 3:	Signal waveforms of toggle cell.....	12
Figure 4:	Toggle cell implementation .....	13



# GALAXY

GALS InterfACE for CompleX Digital  
SYstem Integration

Confid. Level: Public  
Date : 24/12/2008  
Issue: 2

---

Figure 5:	Behaviour of the synchronizer interface .....	13
Figure 6:	A-to-S Gray FIFO.....	15
Figure 7:	S-to-A Gray FIFO.....	15
Figure 8:	Simulation run of the FIFO GALS.....	17
Figure 9:	Block diagram of a GALS system with pausable clocking.....	18
Figure 10:	GALS architecture for high-speed data communications (a), d2g (b), and g2g (c), port specifications and implementation.....	19
Figure 11:	Timing diagram for d2g port .....	20
Figure 12:	Typical OCP Configuration between the Master and Slave .....	24
Figure 13:	Adaptation of the GALS interfaces to accommodate OCP compliant cores.....	25
Figure 14:	OCP Wrapper for GALS interfaces with pausable clocking .....	27
Figure 15:	Simulation run of write and read commands over OCP adapter .....	29
Figure 16:	Simulation run of burst write.....	30
Figure 17:	Simulation run of precise burst read.....	31
Figure 18:	Simulation run of imprecise burst read .....	32
Figure 19:	Simulation run of single request burst read .....	33
Figure 20:	Simulation run of threaded read .....	34
Figure 21:	AMBA AXI Master and Slave .....	35
Figure 22:	2-phase d2g controller .....	38
Figure 23:	2-phase g2d controller .....	38
Figure 24:	Simulation run of the data transfer over 2-phase GALS controller.....	39
Figure 25:	Coupled GALS controllers for bursty data transfer .....	40
Figure 26:	Specifications of the master GALS controller .....	41
Figure 27:	Modifications of the clock generator .....	41
Figure 28:	Transfer of one burst over GALS link .....	42
Figure 29:	Hardware accelerator for 60 GHz OFDM baseband processor (a), and GALS partitioning for burst mode GALS (MP - master port, SP - slave port, LO - local oscillator) (b) and classical pausable clocking (DI - demand input port, DO - demand output port, LO - local oscillator) (c) .....	44
Figure 30:	Burst activity of designed system .....	45
Figure 31:	An example of GALS system (a) & waveform (b) .....	46
Figure 32:	Pausible clock generator.....	47
Figure 33:	Latency in Ack and data caused by RGW .....	48
Figure 34:	Multi-port pausable clock generator .....	48
Figure 35:	Another multi-port pausable clock generator .....	49
Figure 36:	Waveform for the system satisfying (1) & (2).....	50

---



# GALAXY

GALS InterfAce for CompleX Digital  
SYstem Integration

Confid. Level: Public  
Date : 24/12/2008  
Issue: 2

---

Figure 37:	Modified pausable clock generator .....	51
Figure 38:	Comparison in Ack latency and data sampling .....	52
Figure 39:	Modified multi-port pausable clock generator .....	53
Figure 40:	Two structures of MUTEX element .....	54
Figure 41:	Resolution times of MUTEX in two structures .....	55
Figure 42:	STG and synthesized logic of I/O ports .....	56
Figure 43:	Delay slice implementation .....	56
Figure 44:	Simulation waveform (0-280ns) (a) with the standard scheme and (b) with our improved scheme.....	58

## LIST OF TABLES

Table 1:	Summary of Converter Implementation .....	13
Table 2:	Summary of different GALS interfaces.....	21
Table 3:	Industrial preferences .....	22
Table 4:	Comparison in throughput of two schemes.....	57
Table 5:	Performance comparison among multi-port schemes .....	57



## 1 INTRODUCTION

---

The design of the modern Systems on Chip (SoCs) is a very challenging task. The complexity of digital systems grows enormously, and we are approaching nanoscale process geometries. It is expected that this trend will be continued in the following years. The increasing demands of modern SoC applications create several problems for system design and integration. The following issues will have the main importance: system integration of complex systems, timing closure including clock generation and control, system noise characteristics, and power consumption for mobile applications. As a result of this, the SoCs are more and more going from the core-centric architecture into direction of communication-centric systems. This results in a growing application of Network on Chip (architectures). One of the main solutions to target all those issues and to enable successful application of NoC approach is the utilization of a GALS methodology.

This report describes the analysis and optimization of GALS interfaces. The GALS interfaces are being developed over the last several years. There are already several mature GALS interfaces available. There were also several GALS demonstrators designed mainly in academia. On the other hand, the GALS concept was not frequently utilized in commercial products. In order to achieve wider commercial acceptance of GALS methodology some of the unresolved issues have to be answered. The following points are of utmost importance:

- During the last several years different types of GALS architectures have been developed. The main GALS methods are based on pausable clocking, FIFO-based GALS, and synchronizer-based GALS. It is very important to characterize different GALS solutions and to find appropriate applications for such GALS interfaces. Different GALS solutions have different properties and they are not suitable in the same way to all applications. In this report we are evaluating those solutions in respect to hardware complexity, performance, power consumption, EMI profile, throughput, and latency. Special attention is paid to the compatibility of different GALS approaches to NoC concept.
- Standardization of interfaces is very important for today's IP cores. Most of the IP cores today are fulfilling some of the standards such as AMBA AXI or OCP. On the other hand, GALS interfaces are mainly custom and differ very much from the available standards. There is a necessity to adapt GALS interfaces to existing standards. This way we can achieve the compatibility of GALS interfaces with IP cores that are not necessarily designed especially for GALS application. Consequently, in this report we will consider standardization of the existing GALS interfaces. The main focus in this report is on OCP compatible GALS interfaces, but additional analysis is paid to AMBA protocols.
- It seems that there are still some possibilities for further improvement of the GALS interfaces in respect to the important parameters. For example, the interesting properties of a GALS wrapper are hardware complexity, power consumption, EMI profile, throughput, and latency. In this report we will analyze the possibilities for optimization of the GALS interfaces. GALS interfaces based on FIFOs and synchronizers are known for decades and there is not much scope for improvement. Therefore, we will focus in this report to the optimization of the GALS interfaces based on the pausable clocking. In particular, we are considering the performance improvement of this methodology for bursty data transfer scenario and improvement of the pausable clocking blocks.

In the following text we will cover all those issues in detail. In this analysis, we have considered the industrial requirements for the SoC interconnects. We believe that our contribution improves state-of-the-art GALS solutions. We hope that those proposed improvements will lead to more frequent industrial applications for GALS methods.



## 2 REFERENCES

### 2.1 ACRONYMS

<b>AMBA</b>	Advanced Microcontroller Bus Architecture
<b>AHB</b>	Advanced High-speed Bus
<b>APB</b>	Advanced Peripheral Bus
<b>AXI</b>	Advanced eXtensible Interface
<b>EMI</b>	Electro-Magnetic Interference
<b>GALS</b>	Globally Asynchronous Locally Synchronous
<b>FIFO</b>	First-in First-out
<b>IP</b>	Intellectual Property
<b>LS</b>	Locally Synchronous
<b>OCP</b>	Open Core Protocol
<b>MPSoC</b>	Multi-processor System-on-Chip
<b>NI</b>	Network Interface
<b>NoC</b>	Network on Chip

### 2.2 REFERENCE DOCUMENTS

<b>Ref.</b>	<b>Document Title</b>
[AH06]	AMBA 3 AHB-Lite Protocol v1.0 Specification, 2001, 2006 ARM Limited.
[AP04]	AMBA 3 APB Protocol v1.0 Specification, 2003, 2004 ARM Limited.
[AX04]	AMBA AXI Protocol v1.0 Specification, 2003, 2004 ARM Limited.
[BEI06]	E. Beigne, P. Vivet, Design of On-chip and Off-chip Interfaces for a GALS NoC Architecture, <i>Proceedings of 12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06)</i> , Grenoble, France, pp. 172-181, March 2006.
[BEI08]	E. Beigne, F. Clermidy, S. Miermont, P. Vivet, , Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC, <i>Proceedings of 2nd ACM/IEEE International Symposium on Networks-on-Chip (NoCS 2008)</i> , pp. 129-138, April 2008.
[Bj05]	T. Bjerregard, S. Mahadevan, R. Grondahl and J. Sparso: "An OCP Compliant Network Adapter for GALS-based SoC Design Using the MANGO Network-on-Chip", <i>Proc. ASYNC 2005</i> .
[BR97]	David S. Bormann, Peter Y. K. Cheoung, Asynchronous Wrapper for Heterogeneous Systems, <i>Proceedings of International Conference on Computer Design (ICCD)</i> , October 1997.
[CG03]	Ajanta Chakraborty, Mark Greenstreet, Efficient Self-Timed Interfaces for Crossing Clock Domains, <i>Proceedings of 9th International Symposium on Asynchronous Circuits and Systems (ASYNC'2003)</i> , pp. 78-88, Vancouver, Canada, 2003.



# GALAXY

## GALS InterfAce for CompleX Digital System Integration

Confid. Level: Public  
Date : 24/12/2008  
Issue: 2

[CH84]	Daniel M. Chapiro, <i>Globally-Asynchronous Locally-Synchronous Systems</i> , PhD thesis, Stanford University, October 1984.
[CN00]	T. Chelcea, S. Nowick, Low-latency asynchronous FIFO's using token rings, <i>Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems</i> , pp. 210-220, April 2000.
[GU06]	F. Gurkaynak, <i>GALS System Design: Side Channel Attack Secure Cryptographic Accelerators</i> , Series in Microelectronics Volume 168, Hartung Gorre Verlag, ISBN-3-86628-065-3, 2006.
[KE97]	J. Kessels, and P. Marston. Design Asynchronous Standby Circuits for A Low Power Pager. <i>Proc. Intl. Symp. of Advanced Research in Asynchronous Circuits and System</i> , 1997
[KGV7]	M. Krstić, E. Grass, F. Gürkaynak, P. Vivet, Globally Asynchronous, Locally Synchronous Circuits: Overview & Outlook, <i>IEEE Design &amp; Test of Computers</i> , Vol. 24, No. 5. September-October 2007, pp. 430-441.
[KI02]	D. J. Kinniment, A. Bystrov, and A. V. Yakovlev. Synchronization Circuit Performance. <i>IEEE Journal of Solid-State Circuits</i> , 2002
[KR06]	M. Krstić, E. Grass, C. Stahl, M. Piz, System Integration by Request-driven GALS Design, <i>IEE Proc. Computers &amp; Digital Techniques</i> , Vol. 153, Issue 5, September 2006, pp 362-372.
[KR08]	M. Krstić, M. Piz, M. Ehrig, E. Grass, OFDM Datapath Baseband Processor for 1 Gbps Datarate, Proceedings of IFIP/IEEE VLSI-SoC 2008 - International Conference on Very Large Scale Integration, Rhodes, Greece, Oct 13-15 2008, pp. 156-159.
[IYE02]	Anop Iyer, Diana Marculescu, Power and Performance Evaluation of Globally Synchronous Locally Asynchronous Processors, Proceedings of 29th Annual International Symposium on Computer Architecture, pp. 158-170, 2002.
[MO00]	Simon Moore, George Taylor, Robert Mullins, Paul Cunningham, Peter Robinson, Self Calibrating Clocks for Globally Asynchronous Locally Synchronous System, <i>Proceedings of International Conference on Computer Design (ICCD)</i> , September 2000.
[MO001]	S. W. Moore, G. S. Taylor, P. A. Cunningham, R.D. Mullins and P. Robinson "Using Stoppable Clocks to Safely Interface Asynchronous and Synchronous Subsystems" Asynchronous Interfaces: Tools, Techniques, and Implementations, July 2000.
[MO02]	Simon Moore, George Taylor, Robert Mullins, Peter Robinson, Point to Point GALS interconnect, <i>Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems</i> , pp. 69-75, April 2002.
[MT00]	J. Muttersbach, T. Villiger, W. Fichtner: "Practical Design of Globally-Asynchronous Locally-Synchronous Systems", <i>Proc. of the ASYNC'2000</i> , Eilat, Israel, pp. 52-59, April 2-6, 2000.
[MT01]	J. Muttersbach "Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems" Doctor of Technical Sciences Dissertation, ETH, Zurich, Switzerland, 2001
[OC07]	Open Core Protocol Specification, release 2.2, 2007, OCP-IP Association.
[STX]	<a href="http://www.silistix.com">www.silistix.com</a>
[VL03]	T. Villiger, H. Kaeslin, F. Gurkaynak, S. Oetiker, W. Fichtner, Self-timed Ring for Globally-Asynchronous Locally-Synchronous Systems, Proceedings of the Ninth IEEE International Symposium on Asynchronous Circuits and Systems, Vancouver, pp. 141-150, 2003.
[YU96]	K. Yun, R. Donohue, Pausible Clocking: A first step toward heterogeneous systems, <i>Proceedings of International Conference on Computer Design (ICCD)</i> , October 1996.
[ZH02]	S. Zhuang, W. Li, J. Carlsson, K. Palmkvist, L. Wanhammar, An Asynchronous Wrapper with Novel Handshake Circuits for GALS Systems, <i>Proceedings of International Conference on Communications, Circuits and Systems (ICCCAS)</i> , Chuengdu, China, 2002.



### 3 GALS SOLUTION ANALYSIS

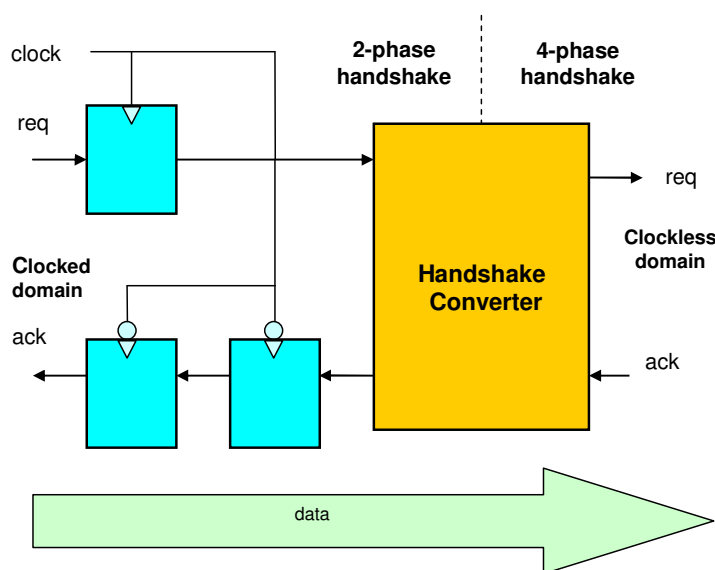
GALS methodology has been known for decades. However, GALS circuits were not up to now widely used in the products. One of the reasons for this was lack of interface evaluation and missing analysis of possible application fields for the particular GALS interface type. According to [KGV7] all GALS interfaces can be classified into three major groups: GALS interfaces based on synchronizers, FIFO GALS, and GALS with pausable clocking.

In the following subsection we have analyzed, modelled, simulated and evaluated the main GALS architectures from each of these groups. The main parameters that we have observed are hardware complexity, throughput, latency, EMI profile, power consumption. Networks on chips are very important application fields for GALS interfaces. Therefore, an additional important parameter is compatibility to the existing NoC platforms. Finally, we have identified which GALS methodology was most appropriate for particular applications.

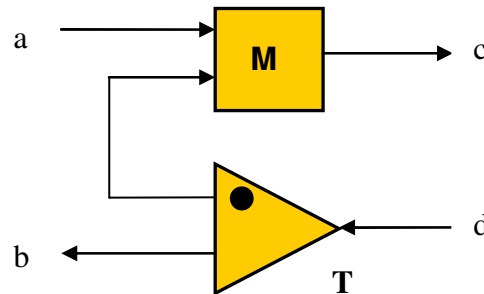
#### 3.1 GALS INTERFACE BASED ON SYNCHRONIZERS

One of the standard solutions to connect two mutually asynchronous domains is based on the synchronizer application. Normally, this presumes application of one-flop or two-flop synchronizers or some of their variants. Such solution for interconnecting different clock domains has been known for decades. In the following text we have analyzed one solution that is recently provided from [Bj05] and applied for building NoC (Network on Chip) interconnect. This is a variant of the classical two-flop synchronizer.

The provided interface synchronizes a synchronous interface from one side to the asynchronous surroundings. In order to improve the throughput on the synchronous side 2-phase protocol is used. On the other side, it is known that on the asynchronous side it is much more efficient to use 4-phase protocol. Otherwise, 2-phase handshake controllers are much more complex and performance is reduced. Therefore this circuit combines synchronizers and handshake converters. In order to convert 2-phase to 4-phase handshake data protocol, a 2-phase to 4-phase handshake converter with synchronization logic is designed. The block diagram of the converter is shown in Fig. 1.



**Figure 1: Block diagram of 2-Phase to 4-Phase Handshake Converter with Synchronization Logic**



**Figure 2: Block diagram of 2-phase to 4-phase handshake converter**

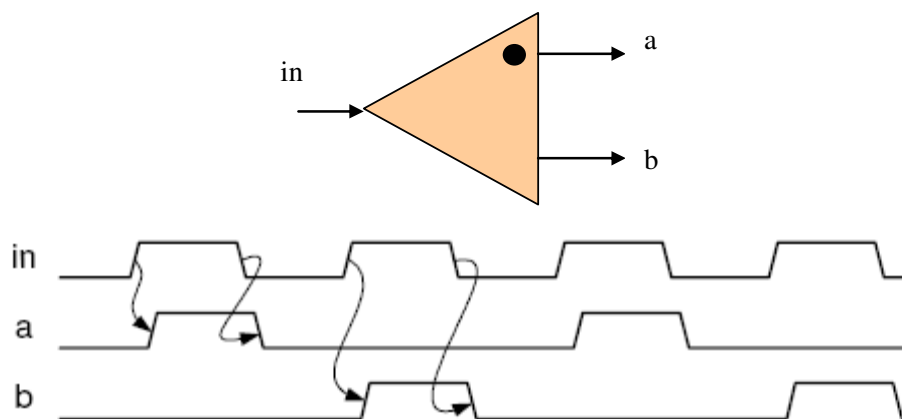
The synchronization logic consists of three flip-flops: one triggered at rising edge and two triggered at falling edge. The purpose of these flip-flops is to convert the asynchronous 4-phase handshake protocol into 2-phase synchronous protocol.

The block diagram of 2-phase to 4-phase handshake converter is shown in Fig. 2. It is implemented using merge and toggle cells. Merge and toggle blocks are classical asynchronous blocks widely used in asynchronous circuits.

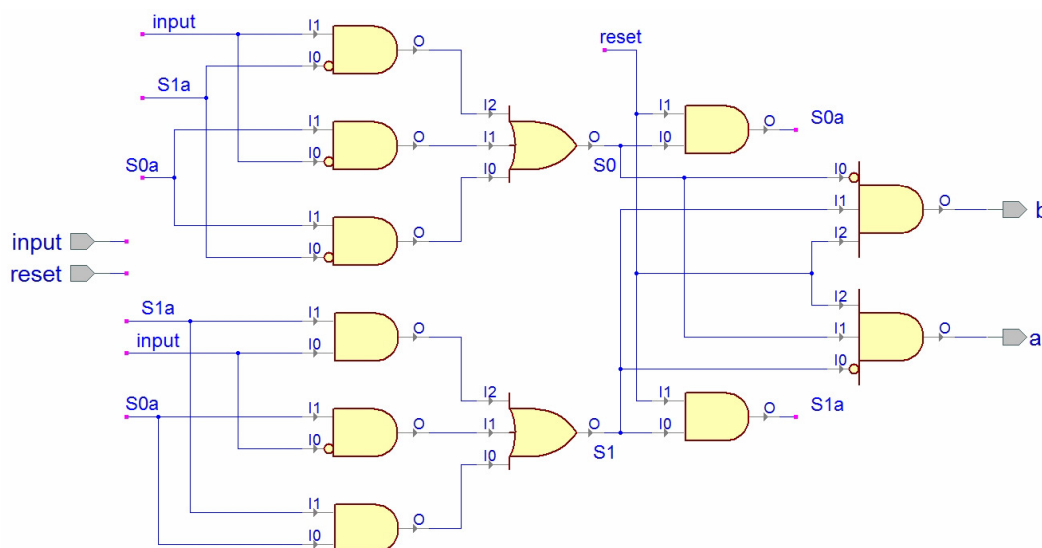
Input *a* and output *b* alternate, as do output *c* and input *d*. Every two phases *ab* enclose 4 phases *cdcd*.

The merge cell can be simply implemented using single XOR gate. Moreover, XOR gates are often called merge elements because they merge two event streams into one.

The toggle module steers incoming events to its outputs alternately; the first event to arrive is issued to the output marked with a dot, the second to the unmarked output, and so on. Signal waveforms of toggle cell are shown in Fig. 3. Figure 4 shows circuit implementation of toggle cell. In order to ensure correct initialization, a reset signal is introduced.

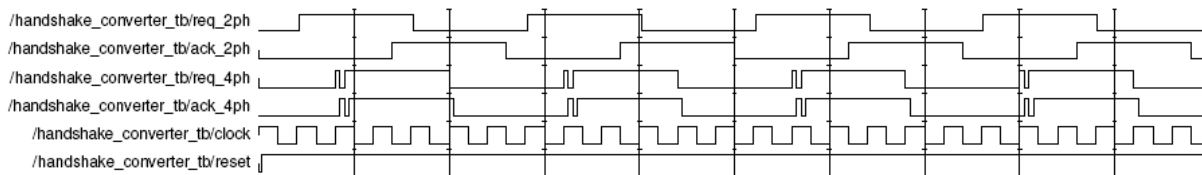


**Figure 3: Signal waveforms of toggle cell**



**Figure 4: Toggle cell implementation**

All building blocks and the complete 2-phase to 4-phase converter with synchronization logic are described in VHDL and verified in simulation. For the testing we have used a set of input stimuli according to handshaking protocol is defined. An example of the typical GALS synchronizer behaviour is provided in Fig. 5 and Fig. 6.



**Figure 5: Behaviour of the synchronizer interface**

In order to estimate the hardware complexity of the wrapper we have synthesized the following circuit in IHP's 0.25 um CMOS process. After synthesis process, the estimated area of the converter expressed in number of INV gates is 63.5.

The converter implementation features are given in Table 1.

Area (INV gates)	Throughput (in clock period)	Latency (in clock period)
63.5	3	2.5

**Table 1: Summary of Converter Implementation**

The major advantage of such a GALS interface is simplicity and safety in data transfer. It is very obvious that the hardware overhead introduced with this approach is very low. On the other hand, the major disadvantage is the throughput of this interface. Due to the synchronizing flip-flops data can not be transferred at every clock cycle of the synchronous block. The application area for this solution is then limited to applications with the limited communication needs. However, due to its low hardware overhead, this GALS architecture may be applied also for not so complex, fine-grain GALS blocks. The application of such synchronizer-based GALS circuitry can lead to the certain EMI reduction. However, much better results can be achieved with pausable clocking. Standard synchronizers may have some problems applied for systems with mesochronous clocking. In this case, the phase relation between the



clocks is always the same (excluding jitter). This way the synchronizer is either not needed or it is working in its non-safe area.

Those GALS architectures are already successfully applied for NoC interfaces. The advantage of synchronizer GALS solutions for NoC is that it can be easily placed at each stage of the NoC link. In addition to that the asynchronous part of the NoC interface can be small or even just localized to the synchronizer itself. Most of today's designers are lacking experience in asynchronous design and CAD tools are not providing enough support for asynchronous designs. Therefore, many designers would opt for solutions with very limited asynchronous part of the systems.

## 3.2 FIFO-BASED GALS INTERFACES

One important class of the GALS interfaces is based on the application of FIFO memories. The reason for applying FIFOs is simple. The synchronization can be hidden in the hardware overhead included in the FIFOs. Therefore the throughput can be increased up to the theoretical maximum of one data transferred per clock cycle. Of course the price is paid by increased the hardware complexity of the interface and latency of the data. The main issue while designing FIFO based GALS interfaces is to generate the correct behaviour of empty/full flags. Since read and write parts are mutually asynchronous during the generation of those signals metastability problems may appear and therefore the synchronization of those signals is needed.

There are several different FIFO-based GALS proposals. An interesting solution is presented in [CN00]. The FIFO presented there includes full/empty detector and special deadlock detector. The solution provided there is quite useful for ad-hoc point-to-point GALS systems. However, with very wide interconnect data buses, FIFO structures could be very expensive in terms of area. Also, the introduced latency might be significant and possibly not acceptable for high-speed applications. In the experiment described in [IYE02], the application of the proposed FIFO structure in a 5-clock domain GALS processor caused a performance drop in the range of 5 to 15 %.

Another interesting approach in the area of FIFO synchronisation is the STARI technique that was presented in [CG03]. This technique is based on a self-timed FIFO that compensates clock skew between different clock domains. However, this approach can lead to a significant performance loss when large data bursts have to be transferred.

More recently FIFO-based GALS interfaces were applied for NoC platform [BEI06]. This solution is based on gray-code FIFOs. The read and write pointers in this interface use a gray code instead of usual binary code. The reason for using gray codes is to increase the reliability of pointer switching for asynchronous FIFOs. Gray codes are specific due to their nature that increase of the pointer always lead to only one change in binary value. Therefore, with such solution it is possible to avoid transitional pointer values that may cause some erroneous behaviour of asynchronous full/empty detectors.

Two types of the FIFO interfaces are developed: A-to-S (asynchronous to synchronous) synchronization FIFO and S-to-A (synchronous to asynchronous) synchronization FIFO depending on the data transfer direction.

### **A-S FIFO interface**

Block diagram of the gray FIFO for data transfer from asynchronous to synchronous domains is given on Fig. 6. The designed FIFO is able to transfer 34-bit data. The depth of the FIFO is 8. Therefore the address field for RAMs used in FIFO is 3-bit. Write side of the FIFO is driven from asynchronous control logic. However it is needed to generate *write\_clock* that can be correlated to request to write data into FIFO.



**Figure 6: A-to-S Gray FIFO**

The read side of this FIFO is synchronous and therefore is needed to synchronize write pointers needed to generate the empty signal. For synchronization double flip-flop is used as given in the following equations.

$@(read\_clock) Wadd1 <= Wadd;$

$@(read\_clock) Wadd2 <= Wadd1;$

$empty <= 1 \text{ when } Wadd2[2:0] = Radd[2:0];$

When synchronized write pointer is equal to read pointer that indicates that FIFO is empty.

Full signal is generated using the asymmetric C-elements. The production rules are as following:

$Full_b \wedge CKW_b \wedge (Radd = Wadd + 1) \Rightarrow Set\_Full+$

$\neg Full_b \Rightarrow Set\_Full-$

$Full \wedge CKR_b \wedge (Radd \neq Wadd + 1) \Rightarrow Reset\_Full+$

$\neg Full \Rightarrow Reset\_Full-$

$\neg Set\_Full \wedge \neg Reset\_Full_b \Rightarrow Full-$

$Set\_Full \Rightarrow Full+$

Comparison between read and write pointers is done just on 2 most significant bits in order to preserve some extra spare place in FIFO. Actually what we get is almost full signal.

### S-A FIFO GALS interface

Block diagram of the gray FIFO for data transfer from synchronous to asynchronous domain is given on Fig. 7. The designed FIFO is also able to transfer 34-bit data. The depth of the FIFO is 8 and the address field for RAMs used in FIFO is 3-bit. Read side of the FIFO is driven from asynchronous control logic. However it is needed to generate *read\_clock* that can be correlated to read acknowledge.



**Figure 7: S-to-A Gray FIFO**



The write side of this FIFO is synchronous and therefore is needed to synchronize read pointers needed to generate the full signal. For synchronization double flip-flop is used as given in the following equations.

```
@(write_clock) Radd1 <= Radd;  
  
@(write_clock) Radd2 <= Radd1;  
  
full<=1 when Radd2[2:1]=(Wadd[2:1])+1;
```

Similar to A-S FIFO for comparison are used only two most significant bits in order to generate almost full signal.

Empty signal is generated using the asymmetric C-elements. The production rules are as following:

```
Empty_b ^ CKR_b ^ (Radd=Wadd) =>Set_Empty+  
  
¬Empty_b =>Set_Empty-  
  
Empty ^ CKW_b ^ (Radd≠Wadd) =>Reset_Empty+  
  
¬Empty =>Reset_Empty-  
  
¬Set_Empty ^ ¬Reset_Empty_b =>Empty-  
  
Set_Empty =>Empty+
```

## **Evaluation of FIFO GALS interfaces**

We have modelled and simulated both types of FIFO. One typical simulation run is given in Fig. 8.

After performing the modelling and simulation of these circuits we made an evaluation of the GALS methodology and architecture. We synthesized this FIFO GALS interface in IHP 0.25 um CMOS process. The hardware complexity of this circuit is around 3.67 kgates (inverter gates).

According to the simulation the throughput of this circuit is very high. The maximal throughput can be up to 1 data per clock cycle. However, the real result very much depends on clock frequencies in the system, architecture of the further blocks etc.

The latency is however increased with the introduction of FIFO interfaces and can be several clock cycles.

In addition to that for high-performance applications FIFO-based interfaces might not be the best choice due to the limited performance they offer. In principle the performance of FIFO-based GALS is defined with the cycle time of the used RAMs.

The FIFO interfaces described in this section can be also successfully used for bridging the long distances on chip, since the interface between the different FIFO interfaces is asynchronous. If one uses standard asynchronous FIFOs for connecting two GALS blocks (as it is in the state-of-the-art CMOS processes) this can result in the inability for long distance interconnects due to the clocked link.

Having in mind those facts the application scenario for FIFO GALS interfaces can be easily identified. The main application field can be the systems with high throughput requirements (demanding data transfer every cycle or in bursts). The target system and GALS blocks must be highly complex to pay-off investment in a big hardware overhead introduced with FIFO interfaces. Additionally, the target system must tolerate latency introduced by this approach. The application of such FIFO GALS circuitry can lead to the certain EMI reduction. However, much better results can be achieved with pausable clocking.



Those GALS architectures are already successfully applied for NoC interfaces in LETI's FAUST chip [BEI06]. However, the future use of the FIFO GALS interfaces for NoCs is questionable due to the large area overhead, latency, low performance in terms of the frequency, and problems in bridging long distances.

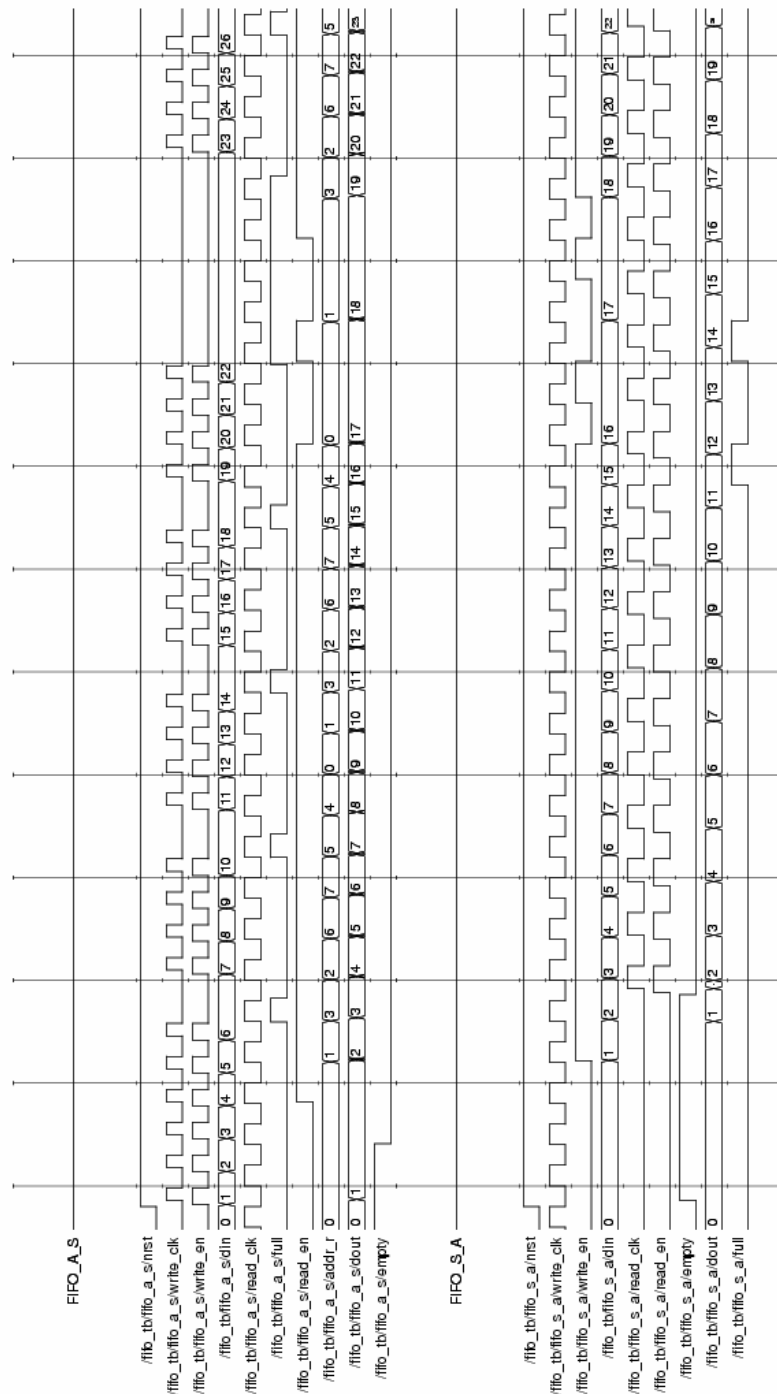


Figure 8: Simulation run of the FIFO GALS



### 3.3 GALS WRAPPER WITH PAUSIBLE CLOCKING

GALS interfaces based on pausable clocking were initially proposed with the first GALS proposal in [CH84]. Use of stretched clocks is the basic idea behind this work and it was widely used many years after. The concept, based on pausable clocks in order to prevent metastability, has been reactivated, and a working architecture is described in [YU96]. Further improvements of this solution were made in [BR97]. For the first time the term ‘asynchronous wrapper’ is used for an asynchronous interface surrounding locally synchronous modules as given on Fig. 9. The purpose of the asynchronous wrapper is to perform all operations for safe data transfer between locally synchronous modules. Accordingly, a locally synchronous module should just acquire the input data delivered from the asynchronous wrapper, process it and provide the processed data to the output stage (port) of the asynchronous wrapper. Every locally synchronous module can operate independently, minimising the problem of clock skew and clock tree generation. The basic idea for all those proposals is similar - transfer data between wrapper when both clocks of data transmitter and data receiver are stopped. Therefore, the problems of synchronization between two clock domains are elegantly solved. An asynchronous wrapper consists of input and output ports and local clock generation circuitry. The locally synchronous part can be designed in standard fashion. This concept is further elaborated in [MT00], and better arbitration of concurrent requests is achieved. The proposed architecture is very general and allows multiport structures of the GALS wrapper, as well as an integration of the GALS system into different interconnect topologies like point-to-point bus, tree, ring, star. Additionally, this proposal is applicable even for data-transfer intensive systems because the data theoretically could be transferred with every clock cycle of the locally synchronous (LS) module.

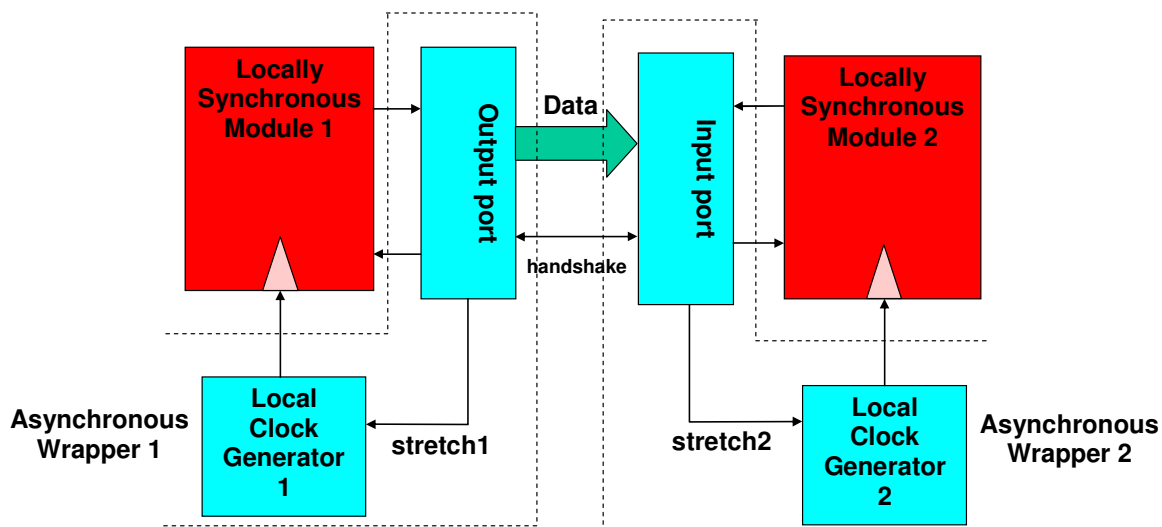
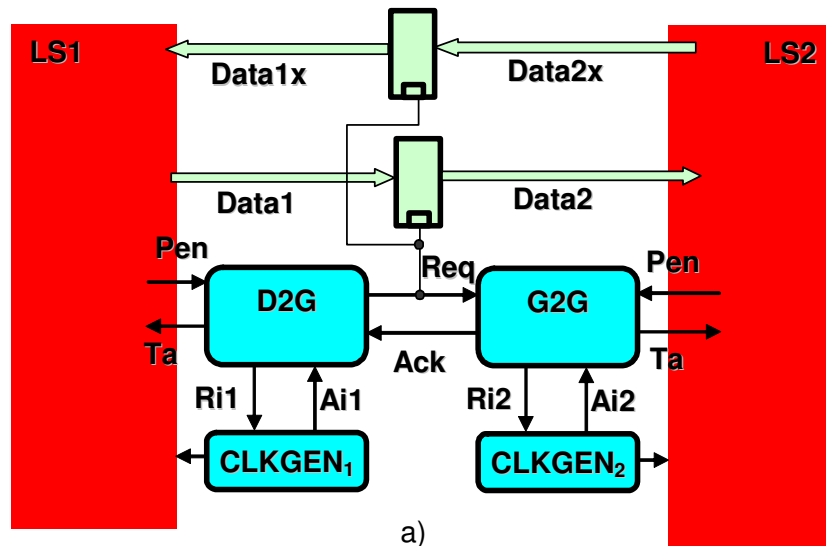
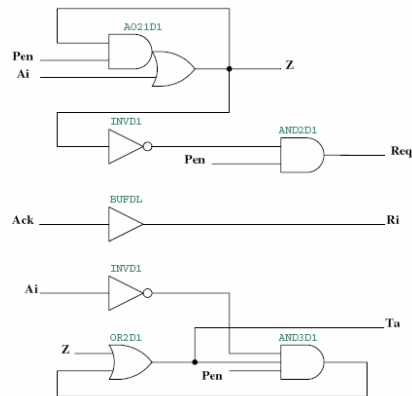
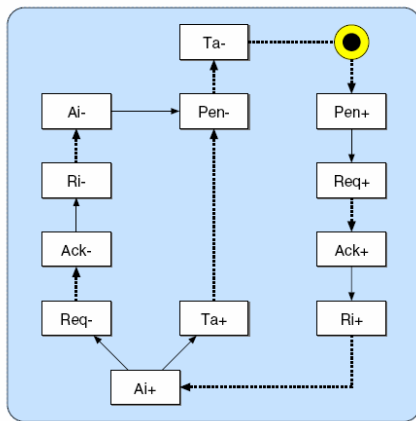


Figure 9: Block diagram of a GALS system with pausable clocking

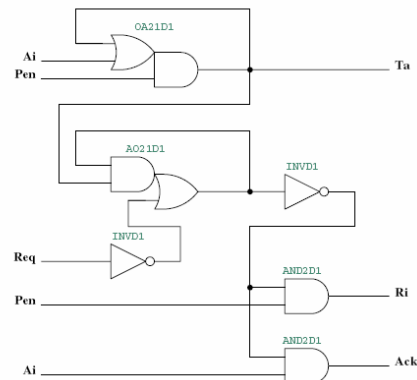
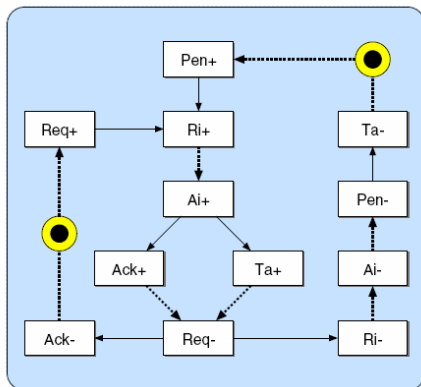
This architecture was to the small extent modified in [GU06] with bidirectional ports. Structure of the proposed system and specifications of the controllers are shown in Figure 10. The port, once activated by *Pen* (A), immediately activates the *Req* signal (B) and waits for *Ack+* (C) (see Fig 11 as taken from [GU06]). The local clock is only paused after the *Ack* signal is received. After the clock is paused (D), data can be safely sampled. At this point, the data transfer is effectively concluded and the *Ta* signal is activated (E). Afterwards, the handshake signals are returned to their idle states and the clock pause request signal *Ri* is deactivated (F). The *Ta* signal remains active (G) as long as the *Pen* signal remains active (H). This is an important change from older port controllers designed by Muttersbach, where the *Ta* signal was only available at the first active clock edge. In [GU06] the operation of the GALS wrapper is much more reliable and *Ta* signal is much more predictable and useful.



a)



b)

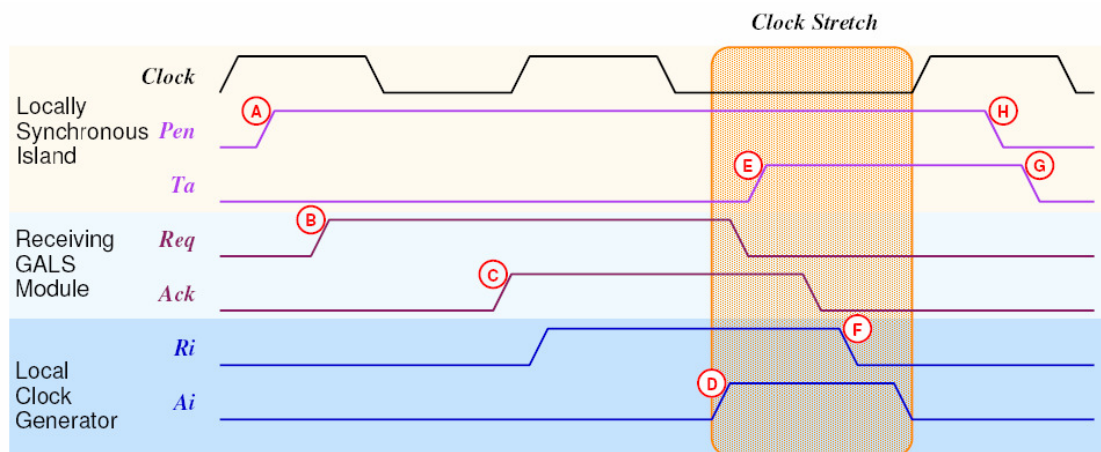


c)

**Figure 10: GALS architecture for high-speed data communications (a), d2g (b), and g2g (c), port specifications and implementation**



The g2d port controller is very similar to the d2g. The  $Ri+$  transition, that will instruct the local clock generator to pause the clock, can only fire after both  $Pen+$  (coming from d2g) and  $Req+$  (coming from Goliath) are received. At this point, LS (locally synchronous) block is ready to transfer data, the local clock is paused immediately, and the  $Ack$  signal is generated. Once the  $Req-$  signal is received, data transfer has been completed and the local clock is released again. Similar to d2g, the  $Ta$  signal is set immediately after pausing the clock and remains active until  $Pen-$  is received. From the port specifications it is clear that the complexity of the controllers is quite low and that they can be realized with a couple of tens of logic gates.



**Figure 11: Timing diagram for d2g port**

This solution was extended with a test methodology based on functional testing of the asynchronous wrapper [GU06]. Another important point is implementation of the adjustable clock generator with the possibility to tune the frequency over a broad range and to self-calibrate clocks. In [MO00] this problem is tackled and the generation of stable local clocks with a fixed frequency is provided. It is suggested to self-calibrate local clocks by introducing an additional low frequency stable clock as a reference. Some similar solutions based on the pausable clocking concept were presented in [MO02, ZH02, KR06].

In general, the proposed pausable clocking scheme offers good data throughput paid with relatively small hardware overhead. Additional complexity is introduced with the ring oscillator and data latch for avoiding the metastability. In general, GALS with pausable clocking adds more hardware overhead than a solution based on the synchronizers but it is still far below the FIFO GALS solution. In general, the throughput achieved with such a solution is quite good (up to 1 data item per clock cycle for the solution given by Muttersbach and 2 data item per clock for Gürkaynak solution). In [GU06] it is noted that throughput reduction in comparison with the synchronous solution was around 15% and energy consumption was increased 20% per data item. The latency introduced with such type of interfaces is also relatively low and originated mainly from additional latching needed between the two domains and time needed for the clock synchronization. The main obstacles for this solution are limited capabilities of the currently available pausable clock generators. Although the solution provided in [MO00] is applicable, the self-calibration mechanism given there is relatively complex and requires additional low-frequency clock source. In addition to that, for higher clock frequencies the precision of clock setting is relatively poor and it is defined and limited to the delay of inverter gate. On the other hand, the application of tuneable clock generators generates an important advantage in respect to possibilities for EMI reduction and increased security against DPA attack. For example, with this solution GALS system may use plesiochronous clocks instead of usual synchronous solution. With this EMI generated from clock source can be decreased up to 20 dB [KR06].

GALS interfaces with pausable clocking have wide application area. Due to the simple structure (critical path of only several gates) they can be also used for high performance applications. This type of interfaces can be also used when high data throughput is needed. Finally, the major application field can be the applications where EMI has to be lowered (mixed-signal designs) or for secure application (crypto-chips etc.). An additional interesting area for implementation of pausable clocking is for systems



that require or where it is optimal to use several different clock frequencies instead of a single clock frequency. With such a solution crystal oscillators on board are not needed any more, and there is a great scope for power saving. This type of the GALS circuits in general shows a great potential for power saving and it can be successfully used in combination with dynamic frequency and voltage scaling and power gating.

Finally, this method for synchronizing the clocks between two locally synchronous islands is very atypical and can be unattractive to many synchronous designers because of its nature that directly affects the clocking of the locally synchronous clocks. In this case, synchronization affects the final performances of the synchronous system and can cause some performance degradation (depending on the clock relations and frequency of data transfer but usually in the order of 10%). The level of this degradation is also not fixed and may vary during the operation of the system. Higher performance degradation is noticeable with more intensive or bursty data transfer. Better results are expected for not so intensive data transfers. Also this methodology is not design-friendly for hard macros and cores that require specific clocking, application of PLLs and complex clocking circuits. However, for complex embedded systems without extreme performance requirements, these techniques can be applied successfully.

Those GALS architectures have been recently successfully applied for NoC interfaces in LETI's telecom chip [BEI08]. In principle, it can be expected that those interfaces will be also used in the future NoC systems especially for ones where the focus is on low-EMI and low-power properties.

### 3.4 CROSS-COMPARISON OF DIFFERENT GALS INTERFACES AND INDUSTRIAL REQUIREMENTS

On Table 2 a summary of different GALS interfaces and their properties is provided. It is quite clear that the decision which interface should be used very much depend on the target platform.

<i>Method</i>	<b>Pausible clocking</b>	<b>FIFO-based</b>	<b>Synchronizer-based</b>
<i>Area overhead</i>	Low	Medium – High	Low
<i>Latency</i>	Low	High	Medium
<i>Throughput</i>	Lowered according to clock pause rate	High	Medium
<i>Power Consumption</i>	Low	High	Medium
<i>Additional Cells</i>	Mutex, Delayline, Muller-C	Empty/Full flag	Muller-C, Mutex
<i>Advantages</i>	No metastability, EMI profile improvement, easy to be integrated with other low-power methods, reduces number of crystal oscillators on board	Simple solution, throughput one data per clock cycle	Simplicity, low overhead
<i>Disadvantages</i>	Local clock generators, throughput, performance degradation, friendliness to standard SoC design, nondeterminism is still present, problems with data bursts and intensive data transfer	Area overhead, latency, low performance in terms of the frequency, problems in bridging long distances	Requires verification, throughput, it can be problematic for mesochronous clocking

**Table 2: Summary of different GALS interfaces**



# GALAXY

GALS InterfAce for CompleX Digital  
SYstem Integration

Confid. Level: Public  
Date : 24/12/2008  
Issue: 2

One additional parameter that is of crucial importance is the alignment of the different techniques to the industry standards and requirements. According to our industrial partner Infineon Technologies, especially from the view of MPSoC system implementation, the priority of the applied interfaces is as given in Table 2.

<b>Synchronization Method</b>	<b>Pausable clocking</b>	<b>FIFO-based</b>	<b>Synchronizer-based</b>
<i>Priority</i>	Medium/Low	Low	High
<i>Comment</i>	Might be difficult to implement on CPUs with several synchronous interfaces.  Performance degradation expected, especially with multi-threaded CPUs.	State of art in current CMOS processes.  Low performance.  Link is clocked → cannot bridge long distances.  FIFOs needed on each side of the link (poor area, latency).	The synchronizers based on the self-time logic are interesting.  It would be good to have the full design flow with automatic insertion.

**Table 3: Industrial preferences**



---

## 4 DEFINITION OF STANDARDIZED GALS INTERFACES

---

During the last years several GALS architectures were proposed. Most of those interfaces are designed in a custom way and do not conform to any of the available standards. Each GALS interface has its own interfaces to the synchronous world that are not compliant to the existing standards.

On the other hand, the process of standardization for synchronous cores has been a lengthy story. There are many different standard interfaces for synchronous IP cores [MS06]. Many of them originated from the ARM processor bus architectures (AMBA APB, AMBA AHB, AMBA AXI) [AX04, AH06, AP04]. The other main standard is Open Core Protocol (OCP) [OC07] that has evolved and been constantly updated over the last several years. Finally, there are also some other standard interconnecting interfaces like IBM CoreConnect (bus based), Virtual Component Interface (VCI), Wishbone bus architecture, etc. From the industrial point of view currently the most interesting interface standards include AMBA AHB and AXI, OCP, SRAM, and DDR standards. One of the major disadvantages of GALS concept is the inability to easily connect any of the IPs that are compatible to the existing standards. This fact significantly lowers the chances of commercial application of GALS interfaces. Therefore, one of the important tasks for further development of GALS interfaces is to adapt them to correlate to the standards for IP cores. For some GALS NoC solutions some efforts are already made in this direction ([Bj05] with a GALS architecture based on simple synchronizers, CHAIN interface from Silistix [STX]). On the other hand, there are no provided solutions for usual GALS interfaces such as pausable clocking. We can consider adaptation of the GALS interfaces for any of those standards. However, we have to take into account the complexity of such an approach and the commercial value of each implementation.

While implementing standard interfaces on a GALS (or GALS NoC) platform the main problem is to enable transfer of read/write commands and data as defined by the standard (OCP, AMBA) to the other clock domain(s). This can be solved in different ways and it depends on the architecture of the GALS or GALS NoC interface. Some of the GALS (GALS NoC) interfaces include asynchronous links only implicitly. This is the case when the interface between two synchronous domains is resolved by some synchronizers or asynchronous FIFOs. Otherwise, there is an explicit asynchronous interconnect that performs the function of data transfer "backbone". In the first case, since almost all operations in the GALS or NoC interface are performed synchronously, the standardization of the external interfaces is not very complex. In the latter case, it is needed that at some point a sync-async and async-sync conversion is performed. This conversion can be done using synchronizers, FIFOs or pausable clocking. In this case, the task regarding interface standardization is more complex. In our opinion, the unresolved issue until now is how to tackle standardization on GALS based on the pausable clocking. Therefore in the further text we will mainly focus on this.

AMBA standards are generally focused on the AMBA bus as a means for transfer. APB (AMBA Peripheral Bus) is very simple and it is intended to be a peripheral bus for memories, and some slow peripheral interfaces etc. AMBA AHB is the specification for the main processor bus. Finally, AMBA AXI is the latest specification that supports concepts relatively similar to the OCP standard, with focus on point-to-point data transfer relations. Support for ARM standard is very common for IPs, even for the one that are not targeted to ARM processors, since having an AMBA compliant interface increases the market chances. On the other hand, AMBA is not really designed to be a general solution for SoCs. It was originally designed to support the bus of the ARM processor, and over time it has been expanded to support more peripherals and different data transfer types.

Open Core Protocol (OCP) delivers an openly licensed core-centric protocol that describes the system level integration requirements of intellectual property (IP) cores [OC07]. OCP standard doesn't define the end-interface (it can be a bus, point-to-point or some other). OCP specifications are generally targeting point-to-point communication links between a master and a slave. The bus specifications are here only implicitly present. Therefore, OCP standard can be successfully applied together with a GALS methodology. GALS methodologies are usually also based (without NoC transport layer) on point-to-point transfer links and the implementation of GALS buses is much more difficult [VL03]. The OCP



standard (the latest is 2.2 release) is a very complex proposal that assembles different types of data transfers but also sideband control and test signals. However, most of the specifications are not mandatory and support for the optional parts of the specifications depends on the targeted application.

The typical interface between the OCP master and slave is given on Fig. 12. The OCP standard supports two types of signals: dataflow and sideband signals. In order to have a minimum compatibility to the standard one has to implement at least basic dataflow signals (given in Fig. 12). Additionally, possible extensions include simple, burst, tag, and thread extensions. Depending on the application those extensions may or may not be implemented.

According to the previous discussion we have concluded that, in the first place, an OCP adapter for GALS interfaces should be considered, due to its point-to-point nature and independence from bus architectures. AMBA AXI is based on similar concepts. Based on the existing OCP adapter it would be relatively simple to implement an AMBA AXI GALS adapter. However, the OCP protocol is more general and independent from a target platform. The sizes of the control and data signals are not predefined as for AMBA protocols and there is more freedom to decide on the optimal data transfer mechanism.

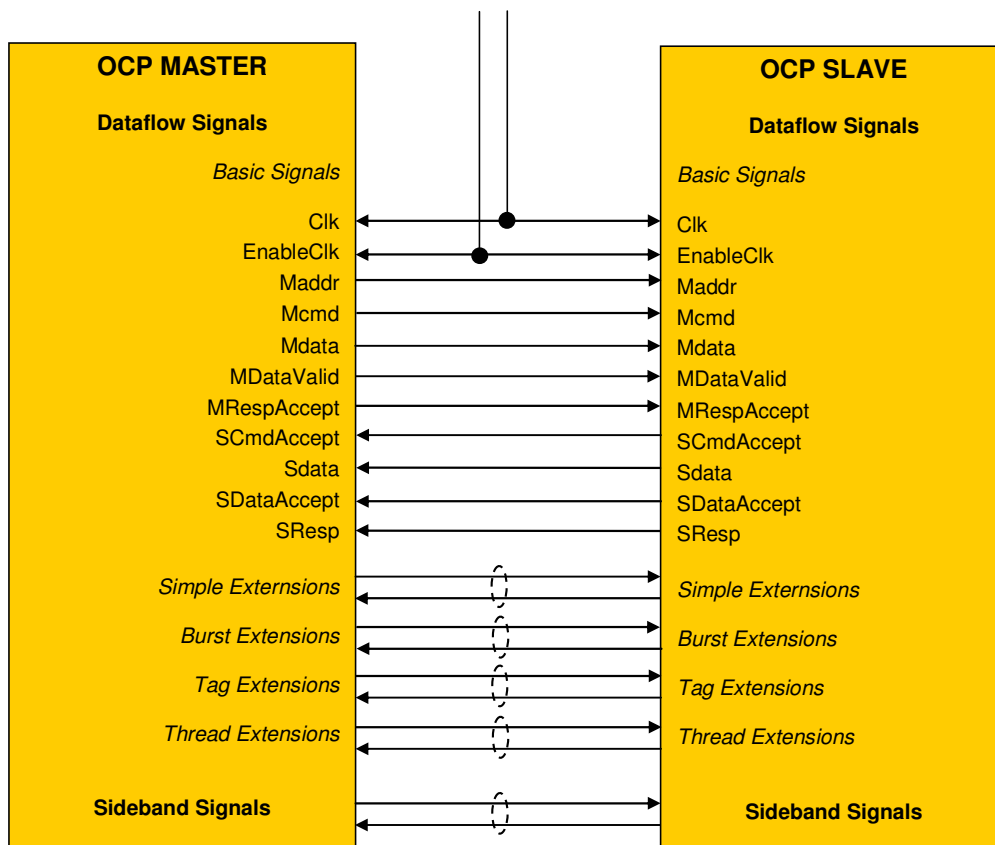


Figure 12: Typical OCP Configuration between the Master and Slave

### 4.1 OCP ADAPTER FOR GALS SYSTEMS

OCP protocol specifications are designed for the application in a purely synchronous world. Therefore, it is not possible to convert it straightforwardly into a GALS environment. GALS interfaces have different control signals and they use different protocols for data exchanges. Consequently, it is needed to design an adapter which will convert OCP signals into GALS signals and vice versa as illustrated in Fig. 13. The purpose of such an adapter would be to comply with the OCP standard interface on one side and on the other side to follow a GALS interface protocol and embed OCP signals in the GALS data bus.



In principle, this OCP adapter together with the GALS interface can be also just used as a front-end for NoC applications. However, the transport layer that is missing here has to be additionally implemented. In that case, some additional circuitry is needed that may include packetizing, parallel to serial conversion, serial to parallel conversion, and network switches. There are different possibilities for implementation of a GALS NoC system and the position of the GALS interface may be different for different architectures.

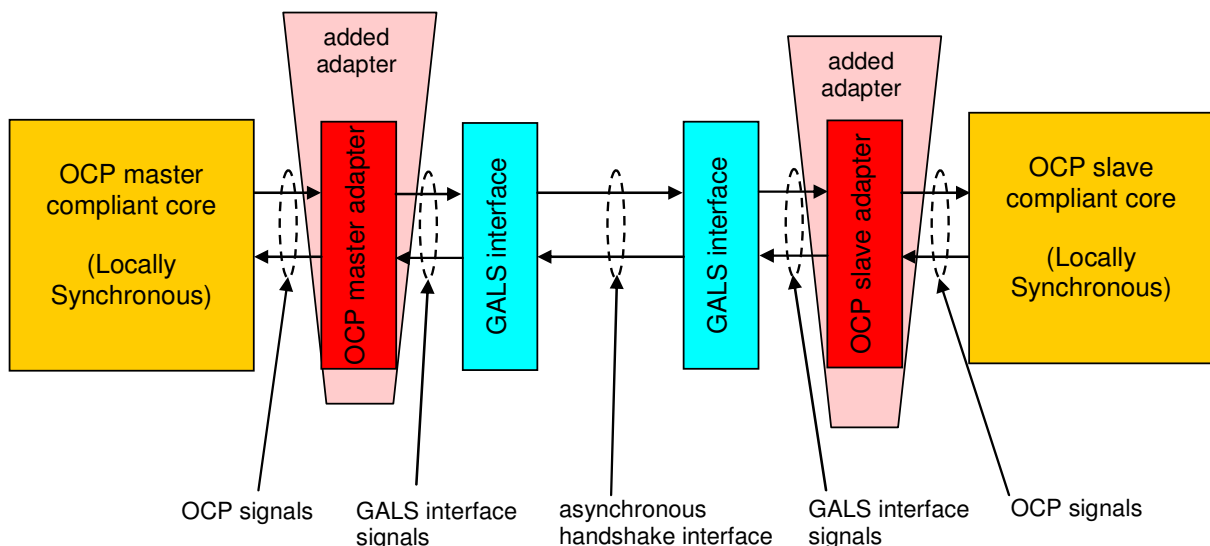
In principle, GALS interface may be placed directly next to the protocol adapter, and in this case P/S conversion, packetizing and switching operate completely asynchronously. However, in some case this may be suboptimal, due to the large number of data/control signals between protocol adapter and the other part of the network interface.

When the network interface is mapped next to the associated locally synchronous core it is convenient to place the conversion to the asynchronous world after the network interface . In this case, the operation of the network switches is asynchronous.

The most conservative solution is to implement network interfaces and switches synchronously and have just a synchronization between the switches. This solution may be interesting when switches are also mapped closely to particular network interfaces and their respective locally synchronous blocks.

If this topology has to be applied to systems that integrate cores complying with different protocols (for example OCP master and AMBA AXI slave), some cross-protocol adapters are needed to translate from OCP to the other protocol and vice versa. The implementation of such adapters can be a significant task. However, this problem is inherently independent from system GALSification, and must be solved in the same way even for completely synchronous systems.

For synchronization between the synchronous and asynchronous domains (or between two independent synchronous domains) any of the different GALS circuits defined in Section 3 can be used. The decision of which one will give the best results must be the result of careful analysis. During this analysis the designer has to take into account the nature and architecture of system, the required data throughput and latency between the different blocks, and (when applicable) the architecture of the applied NoC interface.



**Figure 13: Adaptation of the GALS interfaces to accommodate OCP compliant cores**



## 4.2 REQUIRED SUBSET OF OCP FUNCTIONS

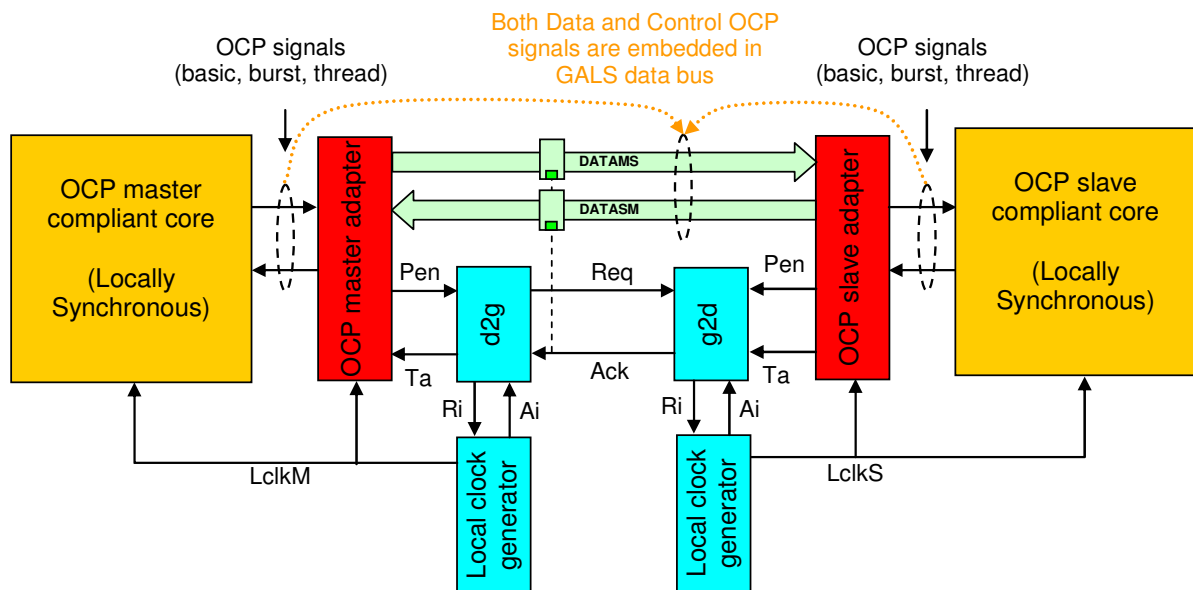
One additional question when implementing OCP compliant GALS (NoC) interface is determining which subset of OCP functions needs to be implemented. The OCP standard and its different releases are very complex and implementing the complete standard leads to suboptimal interconnect and large overheads with reduced performances. However, a minimal subset can be defined in order to provide successful data traffic and enable application of GALS and NoC approach. Our industrial partner Infineon Technologies has defined the following requirements:

- ◆ Mandatory functions
  - Point-to-Point
  - Standard command set (read/write, single/burst, no byte enable restrictions)
  - Outstanding transactions
  - Out-of-order transactions
  - Write response support (configurable)
  - Configurable amount of extension signals for
    - ◇ Command phase
    - ◇ Write data phase
    - ◇ Read data phase
  - Continue on error
- ◆ Optional functions
  - Break on error
- ◆ Nice to have
  - Multipoint
- ◆ Must not
  - Timing restriction (command/data phase)

## 4.3 OCP ADAPTER FOR GALS SYSTEMS BASED ON PAUSIBLE CLOCKING

As defined in the previous section, standardized GALS interfaces can be based on any type of GALS interface. However, in order to evaluate the concept proposed in Figure 13 we have used as a design case the GALS interfaces based on pausable clocking developed by Gürkaynak at ETHZ [GU06]. This GALS architecture is based on asynchronous controllers between the locally synchronous domains and represents the most difficult architecture for implementing an OCP adapter. The implementation of a similar OCP adapter for synchronizer-based GALS or FIFO-based GALS is simpler and requires mainly taking into account standard synchronous assumptions.

The architecture of pausable GALS interfaces including OCP adapters is shown in Fig 14. We have used d2g and g2d GALS ports for master and slave GALS interface that are described in previous section. The role of the OCP adapter is to support data transfer to the standard synchronous OCP master/slave. Additionally, they have to generate *Pen* control signal for GALS ports and respective data signal that will encapsulate data and control bus from OCP master/slave. This GALS interface is based on a synchronous protocol performed with just two signals *Pen* and response signal *Ta*. The protocol used for those signals is classical synchronous handshake. GALS port d2g is performs the function of master port and g2d is a slave port. OCP adapter has to interpret all different commands and types of the communications of the OCP master-slave communication using just those two synchronous signals.



**Figure 14: OCP Wrapper for GALS interfaces with pausable clocking**

This problem is solved in the following way: whenever the transfer between GALS master and slave is needed, the OCP adapter generates a *Pen* signal and starts the transfer between GALS interfaces. All OCP data and control signals therefore must be transferred over the data bus of the GALS interface. On both master and slave sides, OCP adapters should then decode the control signals transferred over GALS links and generate the respective OCP signals for synchronous cores.

Because of the sequential path between the OCP interface signals and the *Pen* signals, additional latency is introduced in the system. This latency would be avoided if there was a possibility to generate *Pen* using only combinational logic. However, this is not possible due to two important factors. One is that this combinational path will significantly delay *Pen* signal in respect to the clock, and the other is that it is very hard to achieve glitch free behaviour of such *Pen* signal and this can cause the failure of asynchronous FSMs. Therefore, the additional latency must be tolerated.

We have implemented first of all the simple OCP *write* and pipelined *read* command with response (Figure 15). Of course, we have also implemented the *idle* command. The major disadvantage introduced with the GALS solution is the additional latency during data transfers. Due to the nature of the used wrappers the maximum data transfer rate is 1 data per 2 cycles. The main issue is to embed independent behaviour of *SCmdAccept* and *SResp* signal and handshakes.

### **Write command**

A simple write command is one of the basic operations of the OCP protocol. With this operation the master is able to write some data information to the slave. This command is started when *MCmd* is driven with write command and valid address and data is set. The slave acknowledges the command with *SCmdAccept* and the data transfer is acknowledged with *Sresp*. The implementation of the write command is much simpler than the implementation of the read command. Write commands can be immediately acknowledged by the OCP adapter and write data and control can eventually be transferred over the GALS link.

Non-posted write commands can be also implemented. In this case the *Sresp* signal can be significantly delayed with decreased data throughput and increased system reliability.



## ***Read command***

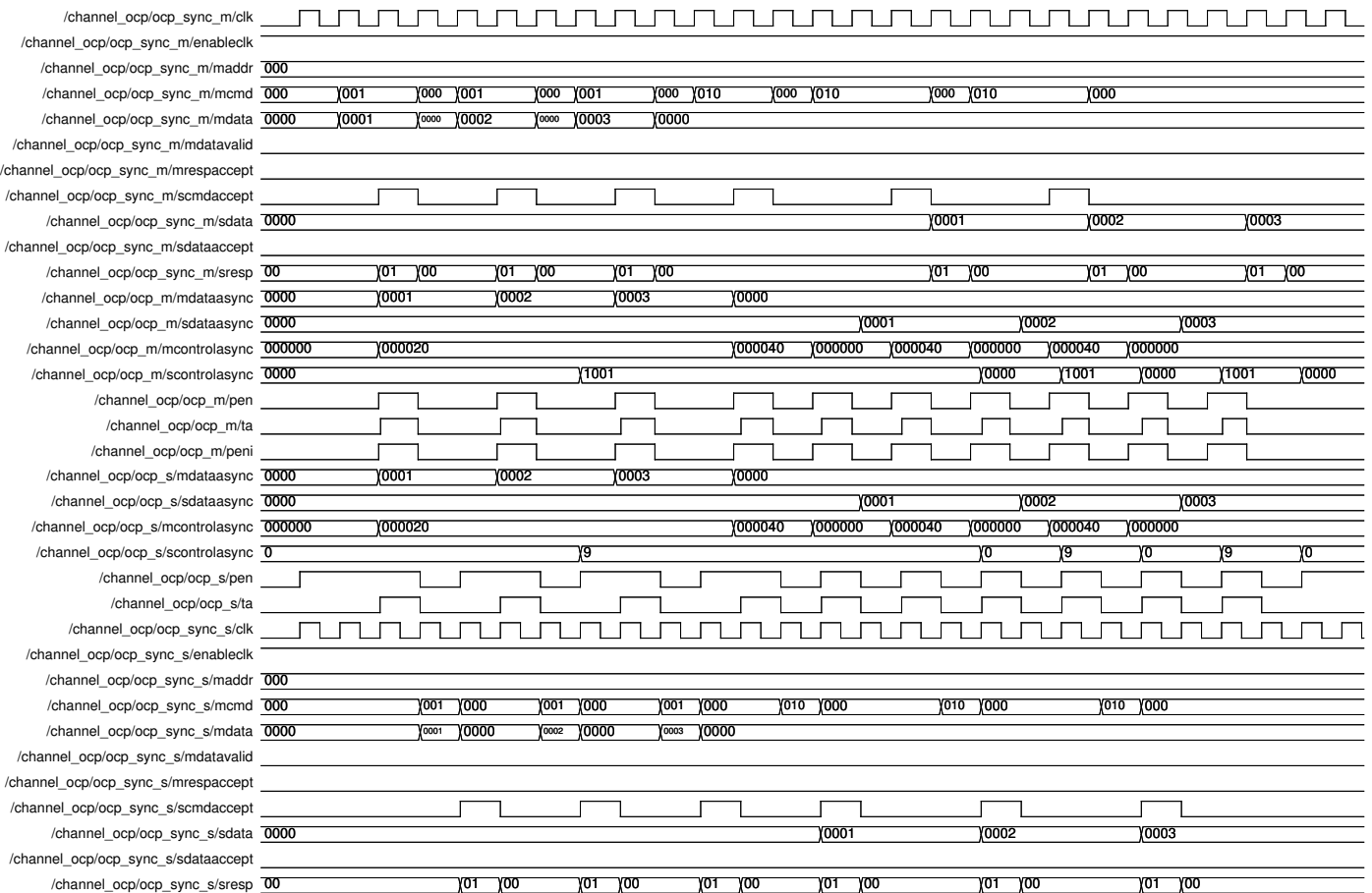
The read command is the second important command of the OCP standard. Due to the nature of the GALS interfaces it is best to implement pipelined read command. Otherwise the throughput will be significantly reduced. With this command the master is able to read some data from the slave. A read command is initiated with correct setting of *MCmd* and a valid address in *MAddr*. The slave acknowledges the command by setting *MCmd*. However, this acknowledge is not necessarily followed by read data. The data may follow even after some cycles. The presence of data is indicated by the activation of the *SResp* signal. The read command is more complex to be implemented since it requires a response from the back side of the GALS link. In practice with our OCP adapter the read command is immediately acknowledged with the *SCmdAccept* signal. Backward handshake (implemented through the *Sresp* signal) followed with read data will appear on the master side after a few cycles. In the meantime further read commands may be generated over the pipelined forward link.

Usually the *idle* command is not transferred over the GALS link in order to decouple master and slave GALS blocks when data transfer is not needed. However, in some cases we have to initiate the transfer between the GALS domains even when the *idle* command is generated from OCP master. The condition for this is a pending *Sresp* signal from the slave. The slave is not able to initiate the communication with the master and it can just react to a request from the master GALS block. Therefore, the master OCP adapter has to initiate the handshake when the *Sresp* signal is expected even when the *idle* command is present at the input of the master OCP adapter.

The other important feature of the slave OCP adapter is embedding the specific counters for counting the number of handshakes performed in forward and backward directions and the number of *Sresp* signals transferred over the GALS link. These two numbers must match and the OCP slave adapter must be able to store all important response actions from the synchronous OCP slave even during the clock cycles when data transfer between the GALS master and slave is not enabled.

The OCP master adapter is in principle easier to be implemented since it also controls the activation of GALS link. However, even there we have to implement the counters for counting the backward and forward handshake transfers and to be able to observe the pending response signals and consequently enable further GALS communication.

With this solution we are able to perform the basic data transfer actions and to preserve a relatively high throughput level. The main limitation is related to the limitation of the GALS interface to transfer data every other cycle. However, it is in any case very difficult to use such a pausable interface when the requested throughput is over this limit. Therefore, the application field of this OCP adapted GALS interface has to be limited to such applications where the requested throughput is not very high. Additionally a price has to be paid with introduced latency, which can be of several clock cycles depending on the clock relations between the master and slave GALS blocks.



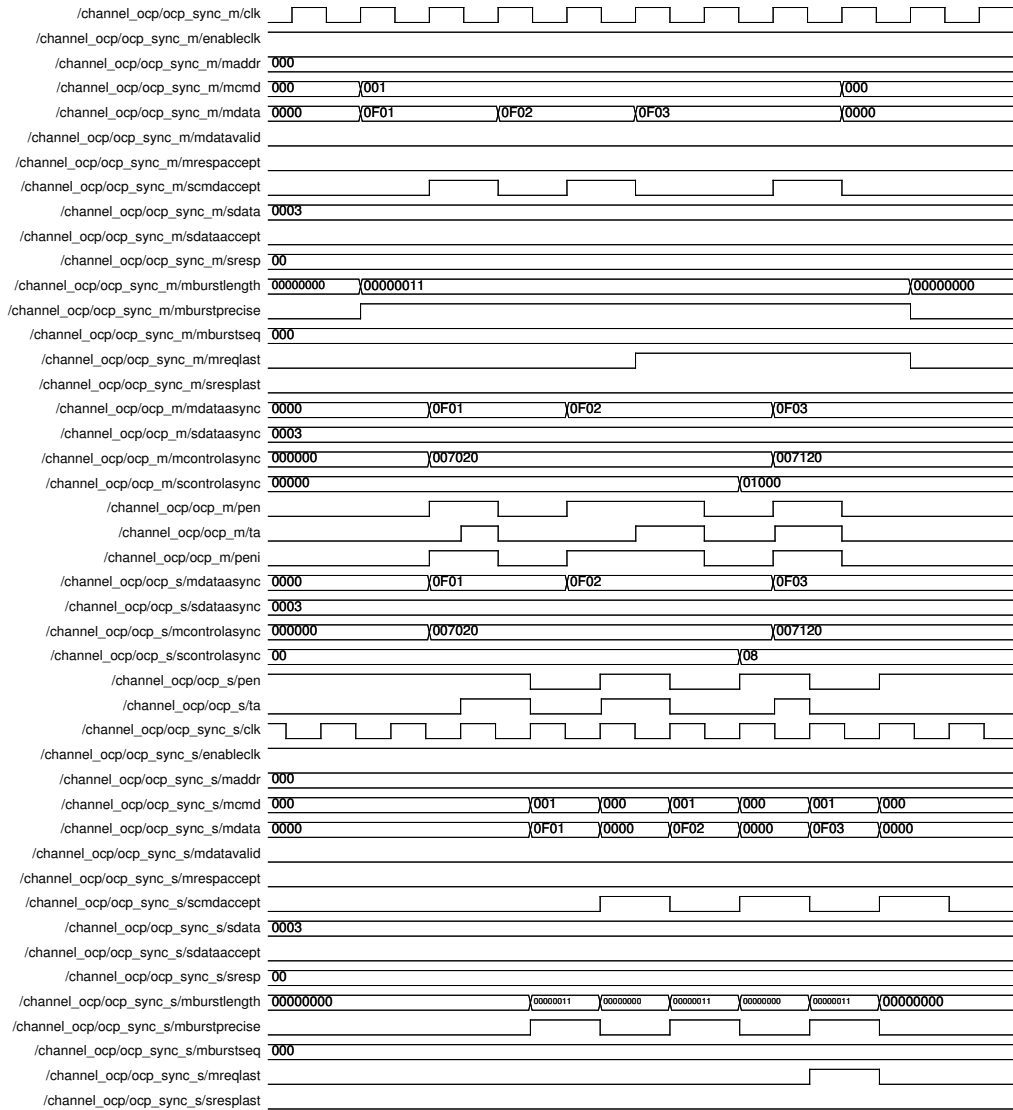
**Figure 15: Simulation run of write and read commands over OCP adapter**

**Burst write**

The OCP standard also defines the possibility to write data from master to slave in bursts. In this case, in addition to the control signals that are set for the normal write command (*MCmd*, *MAddr*, *MData*), an additional set of burst control signals is activated. This includes *MBurstLength* (indicates number of data in burst), *MBurstSeq* (indicates sequence of addresses), *MBurstPrecise* (indicates whether the initial estimation of the burst length is also the final estimation) and *MReqLast* (indicates



last data in burst). Acceptance of the write burst is usually acknowledged with *SCmdAccept* and *SResp* is not mandatory.



**Figure 16: Simulation run of burst write**

In principle the implementation of burst writes in OCP GALS adapter is very similar to the implementation of normal writes. The major difference is omitting the activity of *Sresp* signal. It is obvious that burst operation due to the nature of applied GALS circuit can be executed only using every other cycle of the clock. In addition to that, the latency is somewhat increased. The simulation run is shown in Fig. 16. In this simulation we are writing a burst with a length of 3 words.

### **Burst read (precise)**

Burst operations within the OCP standard are not limited to the write command, i.e. burst read is also possible. With burst read the same additional set of commands is set by the master (*MBurstLength*, *MBurstSeq*, *MBurstPrecise*, *MReqLast*). The slave acknowledges the commands with *SCmdAccept*. However, the data is acknowledged with *SResp*. The last data in burst is additionally acknowledged with *SRespLast*.

The OCP adapter implementation is relatively simple for this operation. There is no major difference to the pipelined read in respect to the adapter implementation. In addition to the burst signals used in









In the architecture of the wrapper adapter the introduction of threads causes not too complex modifications. The major addition to the existing adapter is the addition of the *MThreadID* signal in the forward handshake path and of the *SThreadID* signal in the backward handshake path. Fig. 20 shows one example of threaded read. In this example we have generated on the master side three read commands. First and third are belonging to one thread and second to the other. On the other hand the slave first answers to the first thread and then to the second. The OCP GALS adapter is able to pass the control and data signals correctly over the asynchronous link. Additionally, our adapter can pass and can react to the thread busy signal.

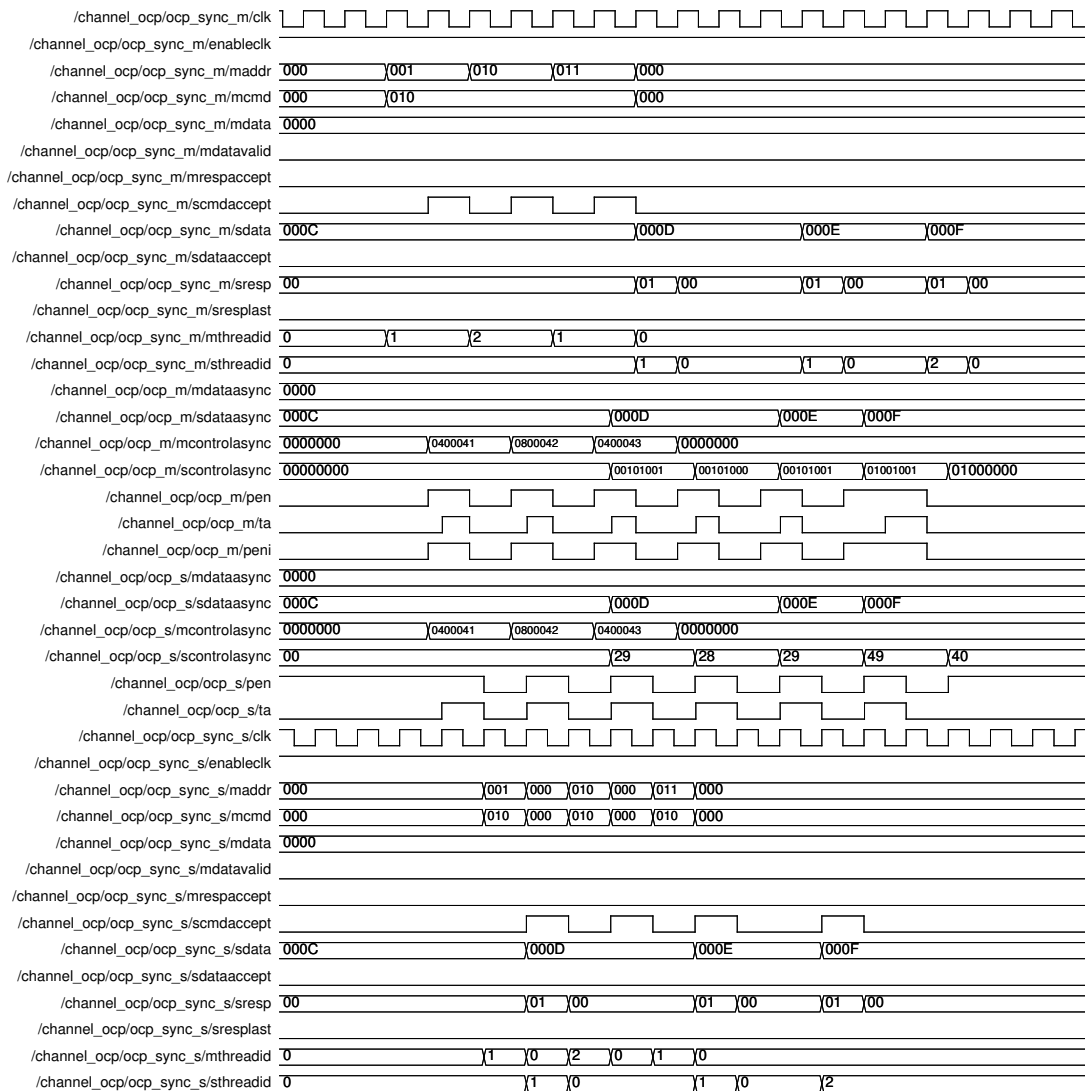


Figure 20: Simulation run of threaded read

### 4.4 COMPLEXITY OF THE OCP ADAPTER

In order to evaluate the complexity of building the OCP adapter for GALS wrapper we have performed the trial logical synthesis of the designed circuitry described in the previous section. The synthesis is performed using a library of IHP's internal 0.25 um CMOS process. According to the results the complexity of the master OCP wrapper is around 820 inverter gates and the slave OCP wrapper ends up with 950 inverter gates.



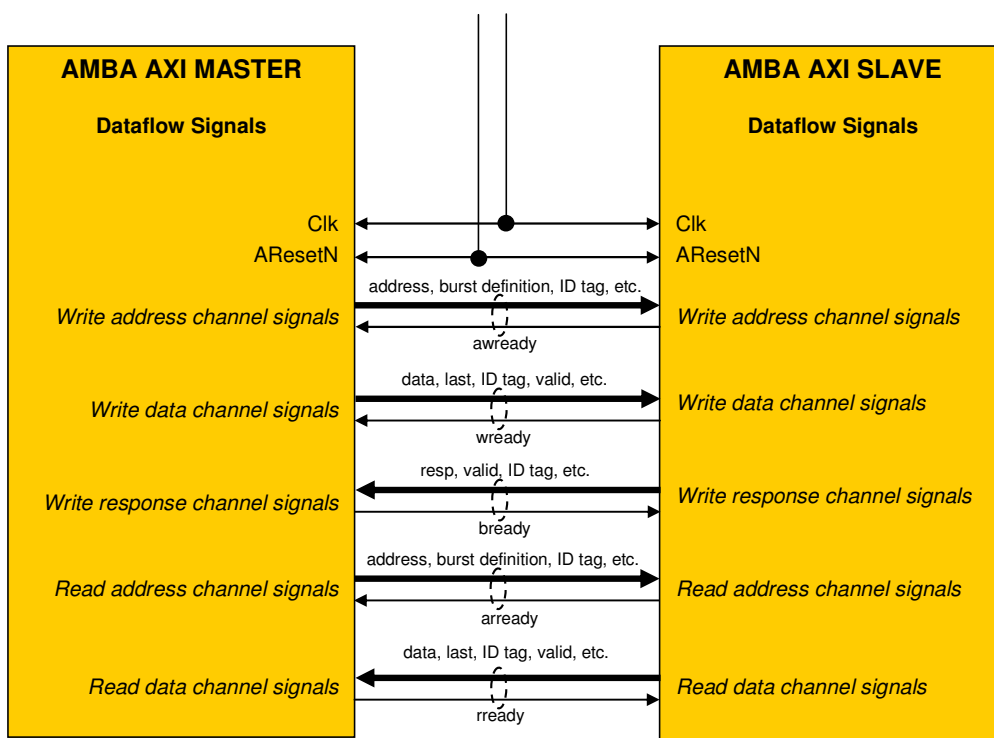
Based on this result we can draw a few conclusions. The hardware complexity of OCP wrappers is not high and is comparable to the complexity of GALS controllers including ring oscillators. However, to justify the investment in these additional hardware structures the complexity of the locally synchronous block must be significant. The reasonable measure for overhead introduced by GALSification should be up to 5% of the total area.

### 4.5 AMBA AXI GALS INTERFACE

Based on similar principles as the OCP adapter it is possible to implement an AMBA AXI adapter for GALS interfaces. The AMBA AXI interface also supports read and write data burst, out-of order completion of the instructions and possibility of a command issued ahead of the actual data transfer.

Similarly to the OCP standard the basis for data exchange in AMBA is a synchronous handshake protocol. Two main signals are used: *Valid* and *Ready*. High value on *Valid* signal indicates the start of the handshake process and *Ready* indicates acceptance of data or control information. This protocol is identical to the handshake process defined by OCP standard.

AMBA AXI is also defined for master and slave configuration. The address and command information is always defined by the master. Also this protocol defines separate read and write channels for data exchange. The architecture of a typical AMBA AXI system is given on Fig. 21.



**Figure 21: AMBA AXI Master and Slave**

As opposed to OCP, the AMBA protocol clearly defines the bit width for all data and control signals. For example data bus can be 8, 16, 32, 64, 128, 256, 512 or 1024 bits wide. This creates a simpler situation for designers but reduces the flexibility and optimality of interfaces.

AMBA AXI also allows burst read and write transfers. However, the length of the data bursts is limited to 16 transactions. In this protocol the last data in burst is indicated with the signal *WLast* for write burst or *RLast* for read burst. With the AXI protocol it is not possible to implement imprecise burst operations and the length of the burst must be fixed. In principle, all bursts implemented with the AMBA AXI protocol are executed as single request bursts.



# GALAXY

GALS InterfAce for CompleX Digital  
SYstem Integration

Confid. Level: Public  
Date : 24/12/2008  
Issue: 2

---

Similarly to threads with OCP, it is also possible to execute transactions out of order. In this case, every transaction becomes an ID tag. Transactions with the same ID tag are completed in order but transactions with different ID tags can be completed out of order.

From all those facts it is easy to conclude that AMBA AXI protocol is based on the same principles as OCP. The complexity of AMBA AXI is however lower than the full OCP protocol that assembles many additional operations and features. Of course, there are some differences in the naming of the signals, bit width of the signals, and ordering of the signals. However, it is relatively simple to generate the similar AMBA AXI adapter for GALS interfaces. This adapter would be based on the same principles and methodology for embedding the standard signals over the asynchronous channel between GALS domains.

## 4.6 AMBA AHP AND AMBA APB GALS INTERFACE

Similarly to AMBA AXI, it is possible to implement an adapter that will be compatible with AMBA AHB and AMBA APB. Those standards are in principle subsets of and similar to the AMBA AXI standard. The implementation of an APB adapter is very simple since these standard supports just basic read and write operations and can make the transfer every other cycle. An AHB adapter however is much more complex, with its support for burst operations and multilayer interconnect structure. Nevertheless, the implementation of those adapters will lead to inefficient and suboptimal operation of the standardized GALS interfaces due to the incompatibility of the GALS approach with bus structures that are inherent to the AMBA APB and AHB standards.



---

## 5 OPTIMIZATION OF THE GALS INTERFACES

---

A more than twenty years long history [CH84] of GALS solutions created the large variety of different GALS methods and interfaces. Some of them are very mature and evaluated over practical implementations [BEI06, Bj05, MT00]. However, there is still a scope for further optimization and improvement especially to improve performance of the GALS solution.

As already described a GALS methodology can be implemented with pausable clocking, with asynchronous FIFOs, and with synchronizers. FIFOs and synchronizers have been developed for many years and we don't see too much scope for further improvements. However, the GALS techniques based on pausable clocking seem to be not entirely optimized. On the other hand, this methodology potentially offers great potential especially in the direction of EMI and power reduction. In the following sections we will present our results of the GALS interfaces improvement.

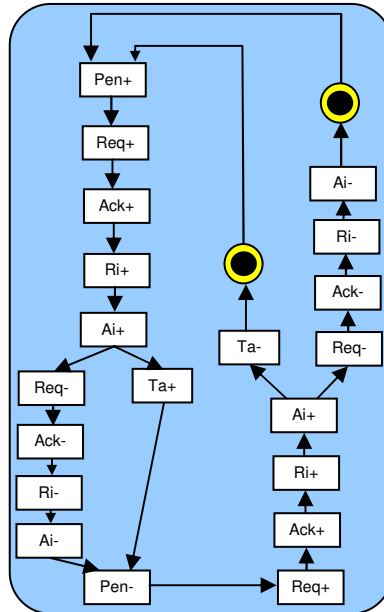
### 5.1 IMPROVING GALS BASED ON THE PAUSIBLE CLOCKING

GALS interfaces based on pausable clocking have been developed for many years. Several solutions have been proposed by different research groups [BR97, GU06, KR06, MO02, MT00, YU96, ZH02]. One of the main challenges of this approach is to achieve the throughput needed for some applications. Many applications need to transfer data bursts and consequently enable data transfer with every clock cycle. One of the most referenced solutions, described in [MT00], actually enables data transfer with every clock cycle of the local clock. This solution is based on extended burst mode (XBM) specifications of the asynchronous controllers and operates correctly and reliably.

However, there are some issues that may create problems. One of the issues is that XBM specifications were only supported by one single academic tool, 3D, which is not available anymore even on the internet. Changing anything to the original specifications as provided in the literature would cause problems. Another issue is the generation of a transfer acknowledge (Ta) signal as described in [MT00]. This signal is provided to the synchronous island both for push and pulls ports. However, the proposed solution is in principle a hack that uses self-resetting RS flip-flops that are rarely provided in standard libraries and where the timing analysis for a complex GALS chip with many clock domains can be very difficult. An additional issue is that for provided GALS ports Ta signal is not even needed since the push and pull ports are completely coupled. That means that both master (push) and slave (pull) ports are pausing the clock whenever they receive Pen (port enable) or Den (data enable) signal from the synchronous part. This results in the fact that a slave port can be paused for a very long time until the master pushes the data or enables a data transfer. Consequently, for many applications this can not be the desired mode of operations and it would be better that the slave continues its operation and data processing until it actually receives some data (indicated by Ta activation).

This solution was carefully redesigned in [GU06] and port controllers based on the signal transition graphs (STGs) are provided. STGs can be successfully synthesized with several tools including Petrify, that has been constantly updated. In addition to that in this solution master and slave ports are decoupled and slave ports are not just waiting for data transfer with paused clock. The problem with this solution is its inability to transfer data with every clock cycle. The maximal rate is data transfer every other cycle. This disables the applicability of this solution for high throughput applications.

We have considered the update of this solution in order to support data transfers with every clock cycle. The STGs for new versions of the d2g and g2d controllers are given in Fig. 22 and 23. It is easy to observe that this solution is very similar to the one provided from [GU06] and just extended from four-phase synchronous handshaking to two-phase synchronous handshaking. In this port specification the Ta signal is directly included in the port specification and is not generated by some hack as in [MT00]. With this solution we can achieve one data transfer with every data cycle.



**Figure 22: 2-phase d2g controller**

Here is the result of the logic synthesis of the controller shown in Fig. 22:

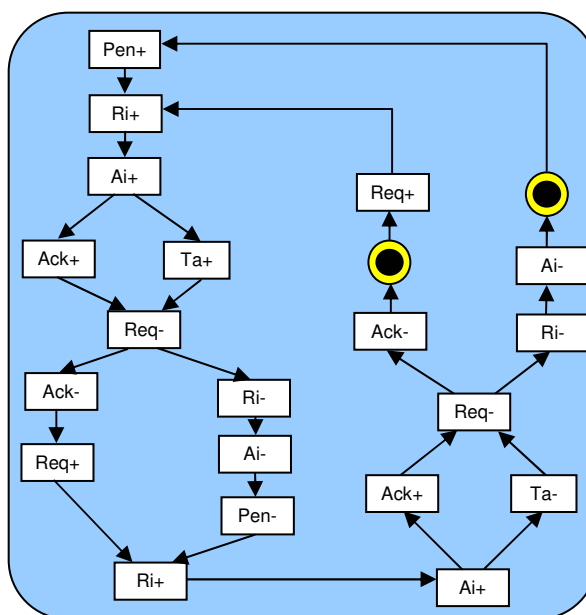
$$Ta = csc0$$

$$Ri = Ack$$

$$Req = Pen\ csc0' + Pen'\ csc0$$

$$csc0 = Ai'\ csc0 + Pen\ Ai$$

where *csc0* is an internally generated state-holding signal.



**Figure 23: 2-phase g2d controller**



Here is the result of the logic synthesis of the controller shown in Fig. 23:

$$Ta = Ta Ai' + Pen Ai$$

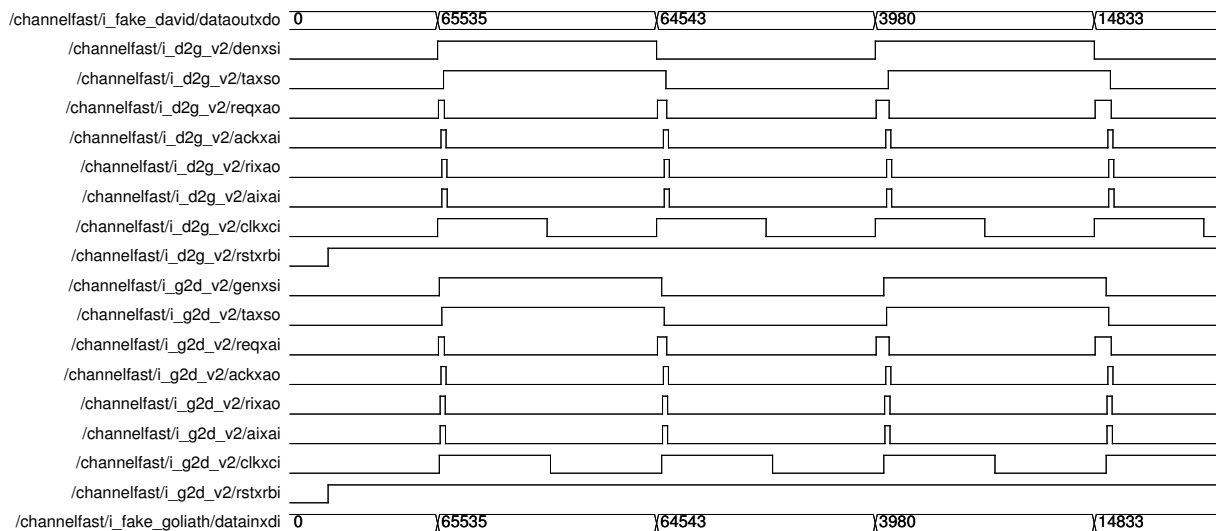
$$Ri = Req (Pen csc0' + Pen' csc0)$$

$$Ack = Ai (Ta Pen' + Ta' Pen) + Ta csc0' + Ta' csc0$$

$$csc0 = Req csc0 + Ta Req'$$

where csc0 is internally generated state-holding signal.

We have modelled in VHDL the complete system including the test master and slave GALS ports. The results of the simulation are given in Fig. 24. From this simulation one can observe that with every clock cycle one data transfer is initiated and clocks are paused.



**Figure 24: Simulation run of the data transfer over 2-phase GALS controller**

The main problem of the proposed solution is the fact that the *Ta* signal has to be evaluated in the same cycle as when the new data transfer is initiated. The only solution for this is that the *Ta* signal is evaluated by a very short combinational path. We have to define some timing window before the rising clock edge where we can safely evaluate the *Ta* signal. This, however, complicates the timing analysis of the ports and creates additional difficulties in the layout process.

## 5.2 GALS FOR BURSTY DATA TRANSFER BASED ON THE CLOCK COUPLING

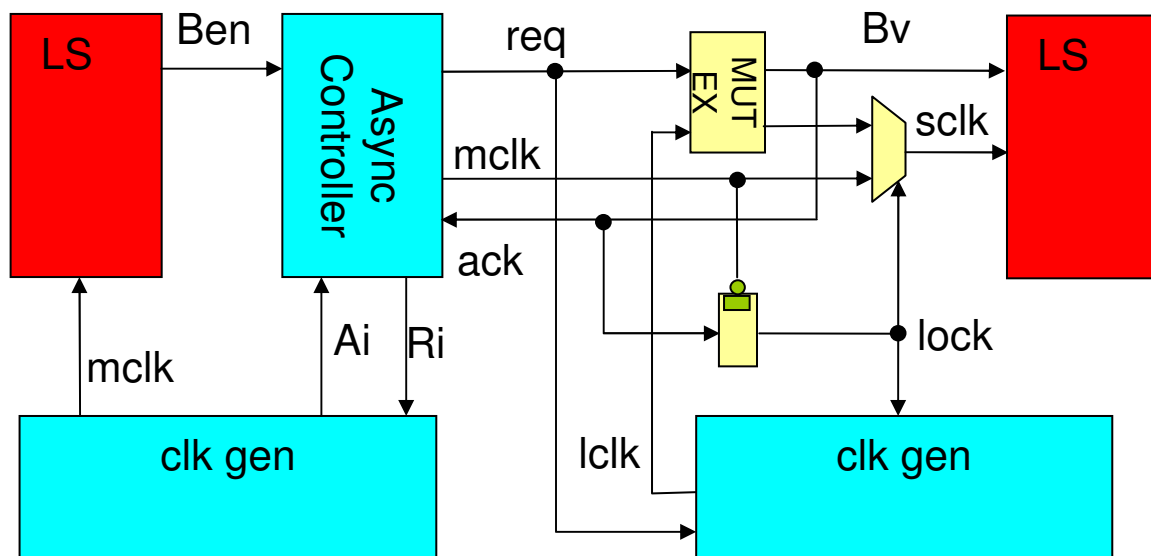
In the previous section we have discussed the possibilities to improve the features of the usually applied GALS interfaces based on the pausable clocking. The main task was improvement of those interfaces for bursty data transfer with support of one data transfer per one clock cycle. However, there is a general problem of the standard pausable clocking interface for bursty data transfer. This problem is intensive stretching of data with every clock cycle that deminishes performances. In addition, to that it looks unnecessary to synchronize the clocks for each clock cycle in the burst mode.

The better idea would be to utilize the single clock during the burst (normally the one with lower clock frequency) for both clock domains or to lock both clock sources to the same clock frequency. In this case during the burst transfer both locally synchronous (LS) domains would use the clock with same frequency. When burst is finally transferred, those LS domains can be again triggered with their own independent clock sources. With such solution we can achieve independent operation of the locally synchronous blocks when data is not being transfered and locked operation in data transfer mode. The



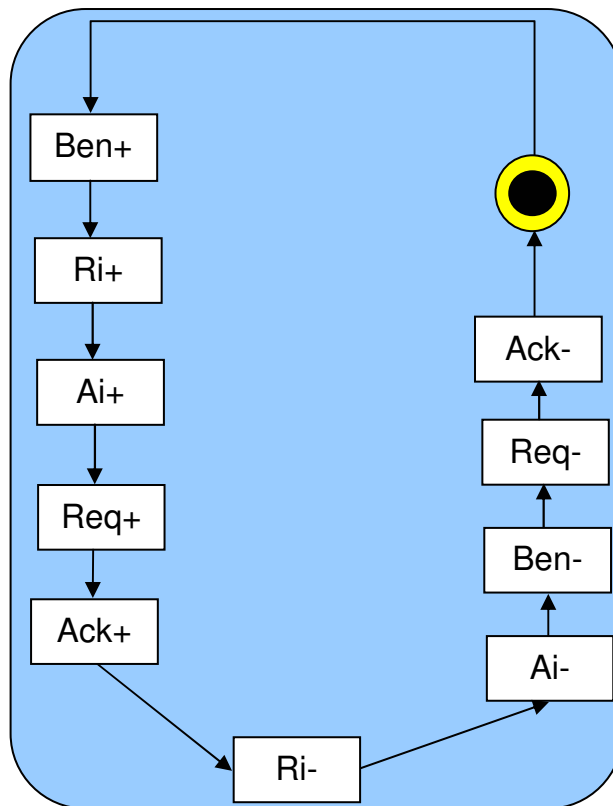
clock synchronization is then needed only once per data burst. Another positive aspect of this technique is the ability to use a separate operation speed during data transfer and operation modes. Similar idea is used in [KR06] with so called "request-driven approach". However, the solution presented there is very complex.

One possible solution based on the described idea is illustrated on Fig. 25. In this figure a typical configuration of two independent LS block are given. Those independent blocks are triggered with different clock generation units. Data transfer is controlled over master and slave port controllers. Burst data transfer is initiated when LS block activates burst enable (*Ben*) signal to master control port. When burst transfer starts, the slave clock control is transferred from its own clock generator to master clock. This is indicated with activation of lock signal. When the burst is being transferred this is indicated to slave with the activation of the burst valid signal (*Bv*). The main point is that when the burst transfer is initiated the clocks are locked, i.e. both master and slave use the same clock source. Therefore, we are indicating with lock signal necessity that one of the clock sources has to be locked to the other. In the configuration provided in Fig. 25 slave clock has to be locked to the master clock. *Lock* signal generated from based on the arbitrated *req* signal and must be sampled with one additional latch to avoid the clock locking when master clock is on high. Burst request (*req*) is arbitrated with locally generated clock. When local clock is not active request will propagate and generate burst acknowledge (*ack*). *Lock* signal is further used to control the clock multiplexer. This operation is performed safely when both clocks are low.



**Figure 25: Coupled GALS controllers for bursty data transfer**

The specification of the master controller is given on Fig. 26. In principle, this is a very simple asynchronous controller which pauses the clock when burst enable arrives and activates request for burst (*Req* $\uparrow$ ). When the burst is acknowledged (*Ack* $\uparrow$ ) clock control is released and burst transfer starts. When all data are transferred *Ben* is deactivated (*Ben* $\downarrow$ ) and port controller then deactivates handshake signal between GALS blocks (*Req* $\downarrow$ ). When slave port deactivates acknowledge (*Ack* $\downarrow$ ), the master port is ready to start with another burst.



**Figure 26: Specifications of the master GALS controller**

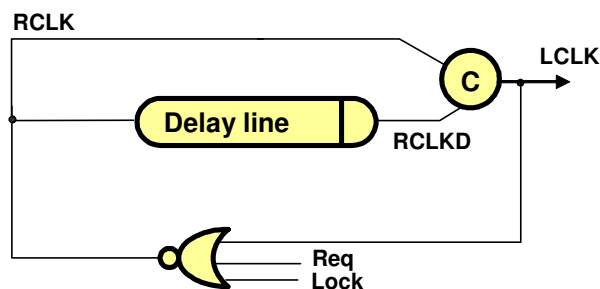
Here is the result of the logic synthesis of the controller shown in Fig. 26:

$$Ri = Ben Ack'$$

$$Req = Ben Ack + Ai$$

From those equations it is clear that the implementation of such controller would be very simple and end-up with just couple of gates.

In order to apply those interfaces some changes of the clock generator must be performed in comparison to the one used in the standard GALS solutions. The block diagram of the modified clock generator is given in Fig. 27. In the normal (locally driven) mode this circuit behaves as a usual ring oscillator block. In locked mode (burst arrives, *req* or *lock* on high) this circuit behaves as stoppable clock. In this way we are disabling the positive clock edge on *Lclk* output whenever *req* arrives and until *lock* is high.



**Figure 27: Modifications of the clock generator**



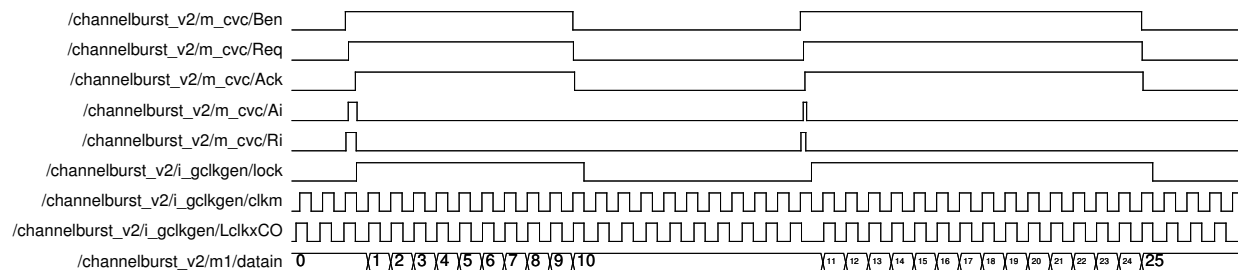
In principle, it is not necessary that slave clock is always locked to the master clock. The other configuration is also possible (when master clock is locked to the slave clock). The rule is usually the following: the clocks have to be locked to the one with lower clock frequency. The reason for that is simple. We are always optimizing some synchronous block for some frequency but this circuit can always operate at the lower frequency. The reciprocal configuration where the master clock is locked to the slave clock is also possible.

The configuration where master clock has to be locked to the slave is similar to the one shown at Fig. 25. The only point is that lock signal is generated from master port and that master clock generator has to be modified instead of the slave generator.

The proposed GALS technique can be also applied in combination to the classical pausable GALS technique. In principle, it is sufficient that the timing critical interconnects with the long bursty data transfers are covered with these type of interfaces.

In order to evaluate the proposed burst mode GALS scheme we have performed the modelling of the simple GALS point-to-point system that is used as a verification vehicle of the concept. On the Fig. 28 a simulation run of two burst transfers over this simple GALS link is given. In the first burst 10 data symbols are transferred, and in the second additional 15 data. The synchronization between the two modules is performed just once at the beginning of the burst. Some time is used to perform the locking the system. This is actually the only time interval where we are introducing additional time loss and where performance is reduced. The complete data transfer is performed afterwards without any loss and additional synchronization. During data transfer the locally synchronous modules are locked and are using the same clock source.

We have synthesized the complete burst GALS interfaces using 0.13 um CMOS process provided from IHP. As already explained the complexity of the controllers is very low. The master controller is of equivalent complexity of 4 inverter gates. More complexity is added by the clock generators. A typical clock generator has complexity of about 600 gate equivalents.



**Figure 28: Transfer of one burst over GALS link**

The throughput simulation gives us the maximum frequency of the controller limited to 609 MHz for the typical PVT conditions (this is approx. 15 FO4 delays). However, the maximum throughput can be even higher since during the burst transfer there is no synchronization and clock is directly transferred from one block to the other. Therefore, during the transfer clock frequency can be higher then this limit. On the other hand, it is very unlikely for this process that we will need the clock frequency higher then this. Of course in the first synchronization cycle the clock will be stopped and the timing interval needed to perform data transfer will be increased for this time. The maximum frequencies of the clock generators are 763 MHz (master) and 581 MHz (slave).

Regardless of the implementation, GALS implementations will always introduce some overhead due to the interface circuitry. However, typical implementations of GALS favor coarse functional blocks which frequently exceed 100 K gate complexity, where the area overhead of few hundred gates will not be a major factor. Also, the frequency limit of the proposed circuitry is in accordance with utilized process for synthesis and it should not normally impose any limitation for locally synchronous blocks. It is plausible to expect that the performance of the proposed interfaces will scale very well with smaller device sizes. Finally, the performance loss introduced over the synchronization period that precedes

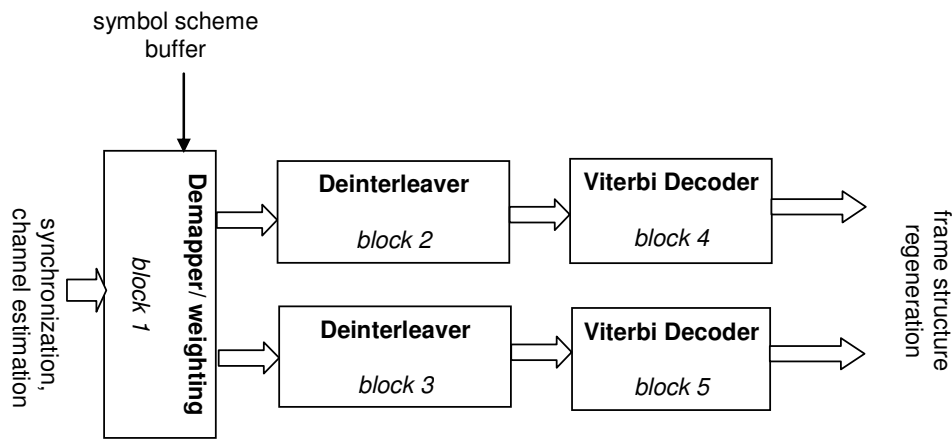


each data burst is usually limited to maximum one clock cycle. However, this loss can be tolerated for longer data bursts.

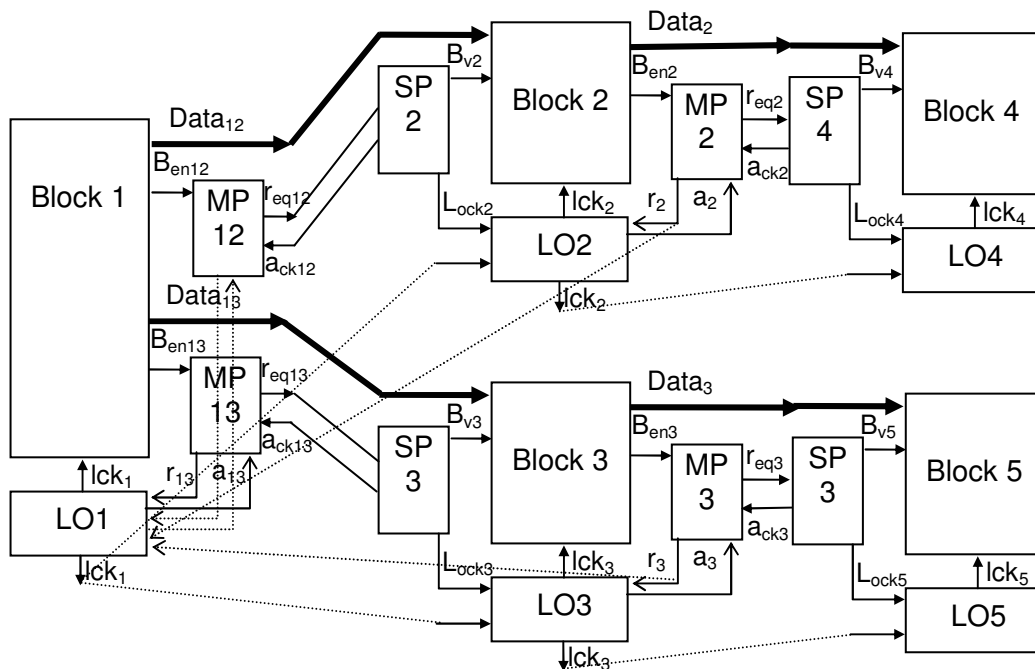
### Experimental design

The more complex but also more realistic case is to have more than one master and slave in the system. However, burst mode GALS scales well also for more involved systems. However, the clock generator for slave block has to be also pausable if this block is also master block for some other GALS module.

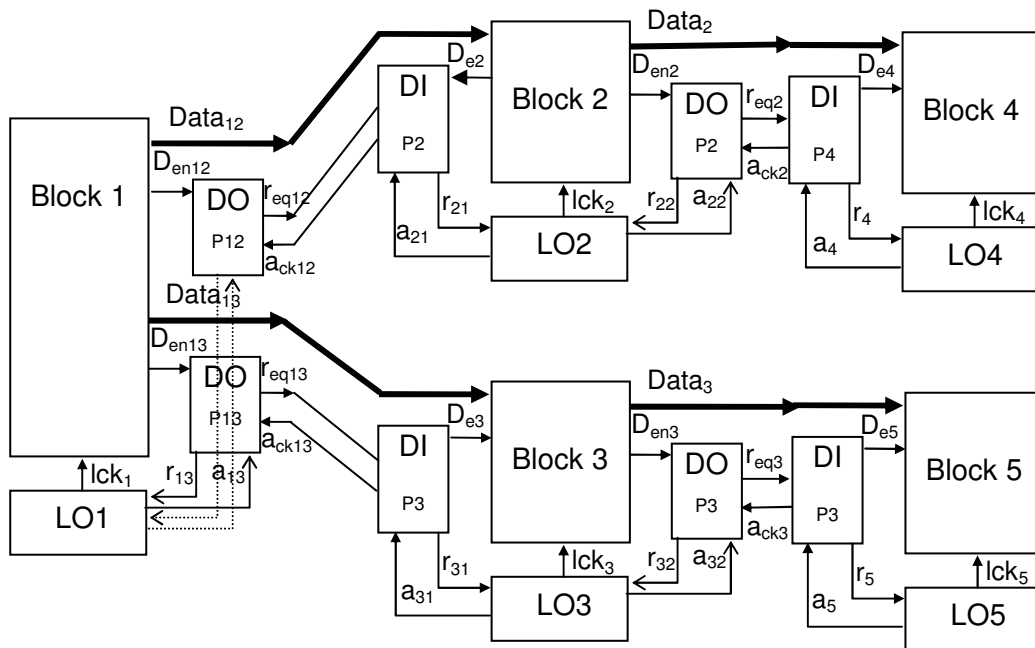
In order to test burst mode GALS wrappers we have built a realistic hardware system consisting of 5 GALS modules. This system is part of the 60 GHz OFDM baseband processor and can be used as a hardware accelerator for such processor [KR08]. This processor has innovative streaming architecture that can meet the throughput of 1 Gbps operating with only 100 MHz clock rate. This hardware accelerator is shown on Fig. 29a.



a)



b)



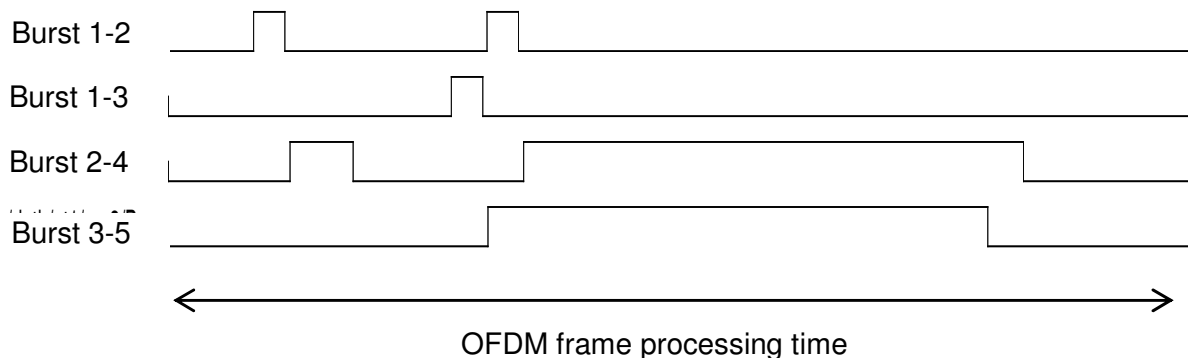
c)

**Figure 29: Hardware accelerator for 60 GHz OFDM baseband processor (a), and GALS partitioning for burst mode GALS (MP - master port, SP - slave port, LO - local oscillator) (b) and classical passible clocking (DI - demand input port, DO - demand output port, LO - local oscillator) (c)**

First block in this scheme contains soft demapper/power weighting block. In this case the system supports two different streams. Therefore, we have placed a set of two deinterleavers/viterbi decoders. Demapper/weighting block in this system distributes data for different stream into one of two available datapaths. With such parallel approach it is possible to double the throughput.

Such system we have implemented two different GALS approaches: burst mode GALS (Fig. 29b) and classical passible clocking (Fig. 29c) as described in [MT01].

In Figure 29b we have shown a relatively complex system of 5 GALS blocks and 8 different ports. Block one acts as a master for Blocks 2 and Block 3. However, since the activation signal (Ben) is different for those two blocks we have in fact two master ports (MP12 and MP23). Blocks 2 and 3 act as a master for blocks 4 and 5, respectively. Therefore, their local oscillator has to be passible to be able to stretch its clock during burst initiation between B2-4 and B3-5. In addition to that, whenever local clock 2 or 3 are stretched ( $r_2 \uparrow$  or  $r_3 \uparrow$ ) we also have to stretch master clock 1 because the slave clocks maybe are driven from the master clock in this moment. Stretch acknowledge signals for master-slave block ( $a_2$ ,  $a_3$ ) are then constituted of a join (using C-element) of the acknowledge signals coming from LO2(or LO3) and master LO1. This is not shown on this figure in order to reduce the complexity of the diagram. This system is successfully implemented and simulated. This implementation confirms the ability of burst GALS scheme to deal also with more complex designs.



**Figure 30: Burst activity of designed system**

The designed hardware accelerator is very well suited for application of burst mode GALS wrappers since the complete data transfer activity is performed in bursts of data as shown in Fig. 30. In this figure the length of each burst for bursts 1-2 and 1-3 is 49 data transfers. Bursts 2-4 and 3-5 have the variable size of the data transfers starting from 97 and up to 769 consecutive data exchanges. For comparison, same system is implemented using classical GALS concept [MT01] (Fig. 29c). In this case we have used demand type input and output ports order to fulfil the requested timing of the system. For both GALS implementations we have used behavioral description of synchronous blocks and synthesized netlists for asynchronous wrappers. Performance analysis has shown that for plesiochronous system (frequencies of the local clocks for the synchronous blocks  $f_{\text{clocki}}$  are similar and almost identical) gain is very low. For example, if the difference between the highest and lowest clock frequency is 3.6% the performance gain of burst mode circuits is only 1.6%. However, if difference is higher performance gain also raises. If frequency difference is 9.6%, burst mode GALS reaches 6.34% higher performance.

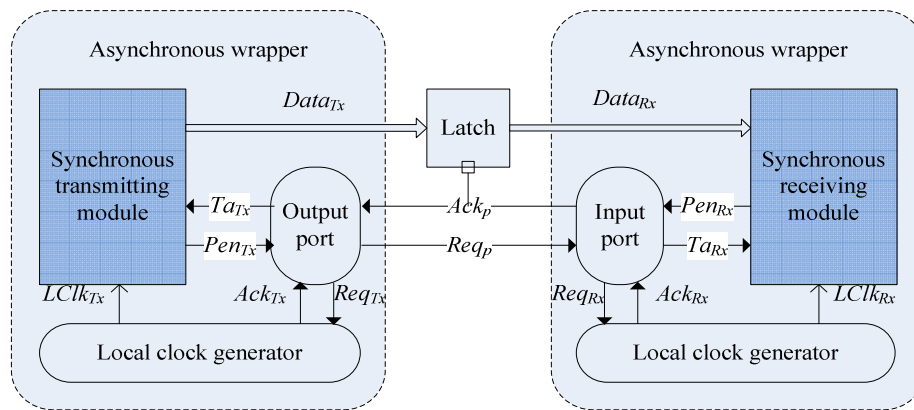
The proposed burst GALS interface represents one step a head in comparison to the existing solutions. For bursty data transfer with this solution we just perform the synchronization once at the beginning of the burst and from this moment no synchronization is performed. This is a significant advantage in comparison with classical pausable clocking interfaces. The applied solution is by nature similar to the one proposed in [KR06]. However, implementation is much simpler and the throughput is much higher. The critical path of the interfaces given in [KR06] were around 45 FO4 delays and the critical path of the burst mode GALS interface is around 15 FO4 delays.

For the systems that do not need only bursty data transfer it is still possible to use the proposed scheme. However, due to initial synchronization delay the results may be suboptimal for the frequent single data transfer commands. For such system it is possible to use different ports to decouple single data transfer commands and bursty data transfer commands. For single data transfer classical pausable clocking interface may be used and for bursts we can apply the proposed scheme. Alternatively, it is possible to modify the burst interfaces to support the special mode of the single data transfers. For such command we will not perform the frequency locking and we will perform the usual pausable clocking scenario. However, the introduction of such mode needs certain modification of the port specification (differentiating a single and the burst transfer request, disabling the lock for single transfer). Such modification will lead to higher complexity of the burst port controllers and possibly also to the lower maximal throughput of them. Therefore, it is best to use the burst controllers for systems with predominantly bursty data transfer and if necessary add the special additional classical GALS controllers for single data transfers.

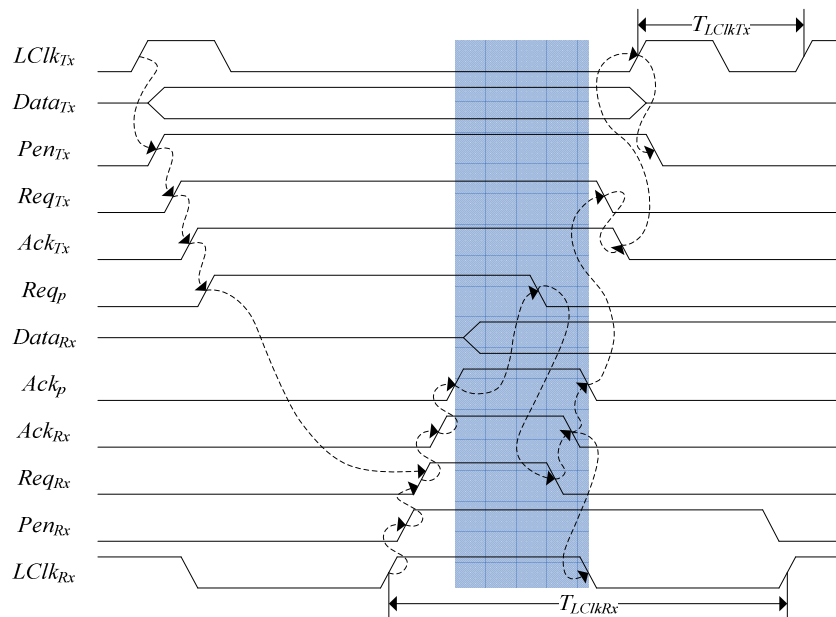


## 5.3 ANALYSIS AND MODIFICATION OF THE PAUSIBLE CLOCK GENERATOR FOR GALS SYSTEMS FOR MULTIPOINT APPLICATIONS

In the recent years, an alternative method for clock synchronization based on pausable local clock is developed for high performance GALS system [YU96, MT00]. As an example, Figure 31 shows a point to point GALS system based on pausable clock and its waveform of handshake signals. Each locally synchronous module is surrounded by an asynchronous wrapper, which mainly consists of a local clock generator and a number of asynchronous I/O ports. Communication between synchronous modules is via asynchronous ports, which generate request-acknowledge handshake signals bundled with data to be transferred. Once the data and the rising edge of clock are too close to each other, the clock will be paused to avoid metastability. The design of the asynchronous wrapper is described in detail in [MT01].



(a)



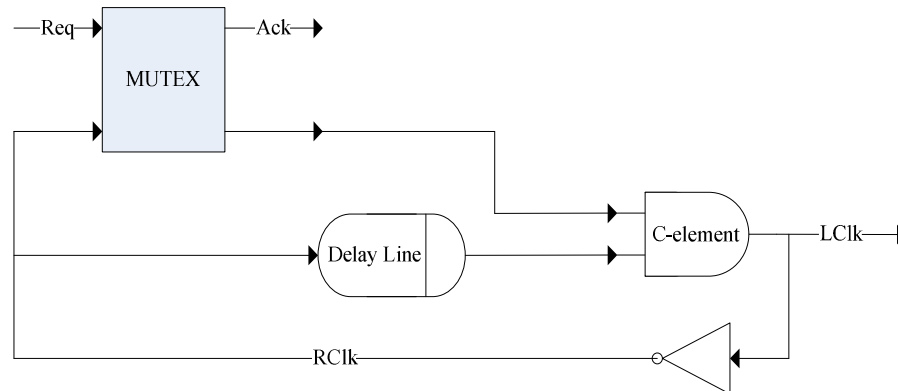
(b)

**Figure 31: An example of GALS system (a) & waveform (b)**



### Analysis of the Existing Clock Generators

A well documented pausable clock generator based on ring oscillator is presented in Figure 32 [YU96, MT01]. A MUTEX element is utilized to safely arbitrate between port request *Req* and the rising edge of request clock signal *RClk*, which is the phase inversion of local clock signal *LClk*. The local clock *LClk* will not issue any new rising edge if *Req* is granted by the MUTEX element.



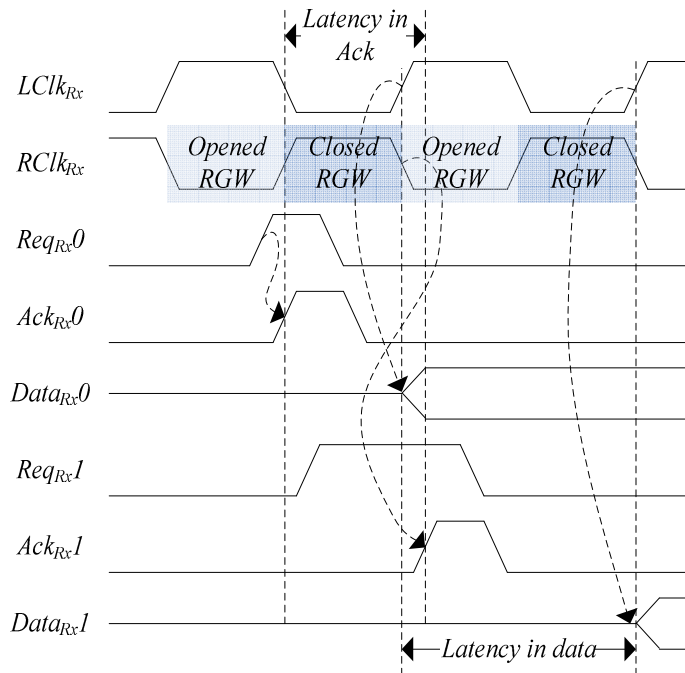
**Figure 32: Pausible clock generator**

At a time only one of two incoming events *Req+* and *RClk+* is allowed to pass by a MUTEX element on first come first serve basis. Any request arriving behind *RClk+* can not be granted until *RClk-* occurs. If *Req+* and *RClk+* arrive simultaneously, the MUTEX element decides at random which one can pass. Therefore the Request Granted Window (RGW), which represents the duration within each *LCLK* cycle at that time *Req* can be granted by the MUTEX element, is the off-phase period of *RClk*, which corresponds to the on-phase of *LClk*. Since the 50% duty cycle in *LClk*, the RGW of the pausable clock generator in Figure 32 is half cycle of *LClk*. Any *Req+* occurrence outside of this RGW results in additional latency.

The potential reduction in data throughput caused by the latency of pausable clock generator will be analyzed as below. For the simplicity, communication between a demand-type output port in the transmitting module and a poll-type input port in the receiving module are described.

### Influences in Receiving Module

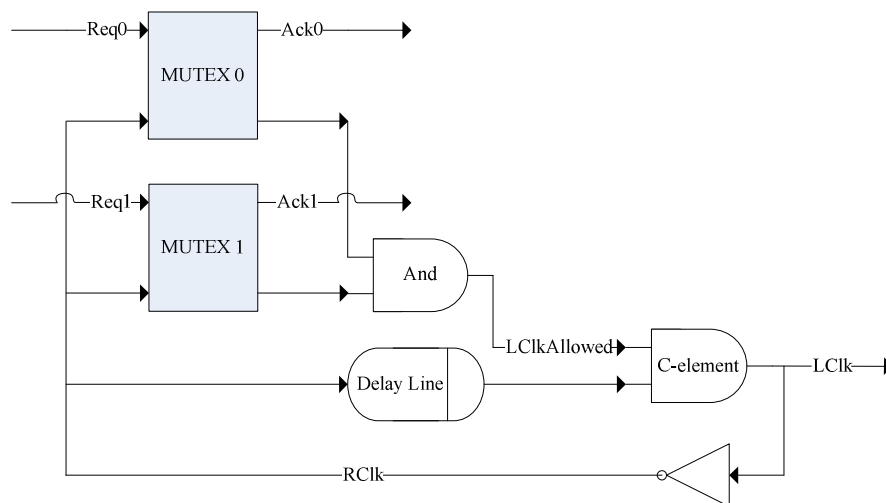
For the receiving module equipped with a poll-type input port, its local clock *LClk* will be paused after *Req+* occurs from the transmitting module [MT01]. Once *Req+* arrives outside of the RGW of current cycle, according to above analysis, it will wait for the RGW of the next cycle to be granted. The maximal latency in responding *Req* is half period of *LClk*, however, since the data is sampled by the receiving module at the next rising edge of *LClk*, the maximal latency in data path is as long as the whole *LClk* period shown in Figure 33.



**Figure 33: Latency in Ack and data caused by RGW**

Since  $Req$  is generated by the transmitting module which runs at an independent clock in frequency from  $LClk$ , the arrival time of  $Req+$  in a period of  $LClk$  can be modelled without loss of generality as a uniformly distributed random variable. Considering the half cycle RGW in above pausable clock generator, there is in theory 50% possibility that  $Req$  is extended and responded in the next cycle.

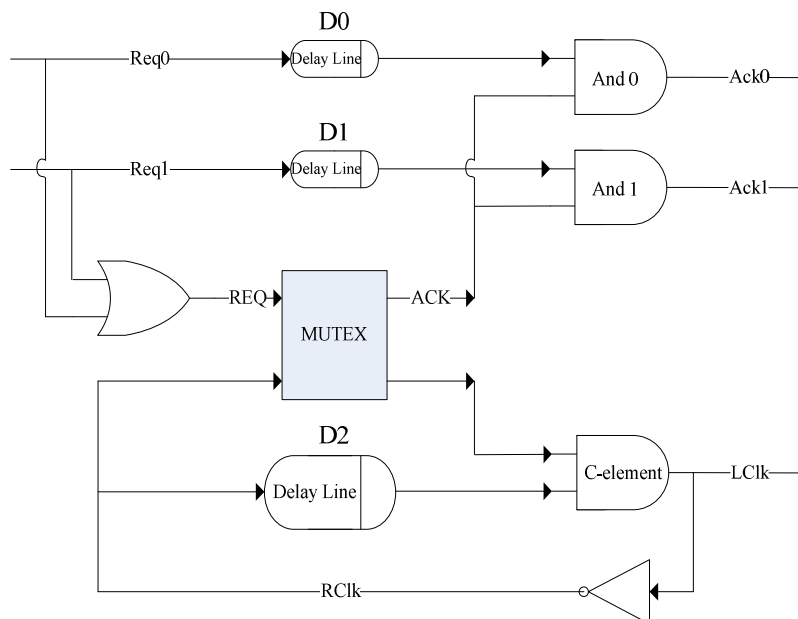
When applying on multi-port communication systems, such latency may result in much heavier degradation in data throughput. An extended multi-port pausable clock generator based on above scheme is shown in Figure 34, where the single MUX element is replaced with an array of MUX elements to deal with concurrent requests [MT01, MO02, GU06]. Since all the requests from different transmitting modules arrive at random, the latency due to the closed RGW will occur more frequently, and there could be a number of requests delayed at the same time.



**Figure 34: Multi-port pausable clock generator**



Another multi-port pausable clock generator is reported in [MT00] as shown in Figure 35. It bundles port requests with an OR-gate to generate a combined port request *REQ* to the single MUTEX element. The *RGW* for *REQ* remains half cycle of *LClk*, however, after *REQ* is granted, it can be extended by all the requests from different ports. Even through the *RGW* of current cycle is over, port requests also can be responded provided *REQ* could be held high. By this means the *RGW* for each port request is actually extended. Of course it contributes to increase data throughput.



**Figure 35: Another multi-port pausable clock generator**

A key problem coming from this solution is to guarantee output acknowledge signals *Ack* free of glitches. In [MT00] this is achieved by employing delay-lines which need to be larger than the sum of propagation delays of the OR-gate and the MUTEX element. In fact, this approach will never eliminate glitches in acknowledge signals, since there is no superior limit in the resolution time of a MUTEX element [KI02]. Increasing the length of delay-line could only lower the probability of occurring glitch. Considering that any glitch in handshake signal *Ack* may result in a disaster in the whole system, the delay-lines have to be long enough to ensure this probability sufficiently lower.

Another issue of the scheme is short low-pulses on *REQ*. Low pulses shorter than the reaction time of the MUTEX element would result in metastability. The solution provided in [3] is to impose some restrictions on the arrival time of port requests. This is less flexible and introduces other limitations on data throughput.

### ***Influences in Transmitting Module***

For the transmitting module equipped with a demand-type output port, its local clock *LClk* need to be paused before asserting a request to the receiving module, and *LClk* will be released after the request is responded by the receiving module [MT01]. That means the latency of pausable clock generator in the receiving module will be finally propagated into the transmitting module. As given in above analysis, the maximal latency to respond requests is half cycle of *LClk* of the receiving module. If the clock period of the receiving module is longer than that of the transmitting module, this latency may result in a multi-cycle delay in the transmitting module. In some circumstances this delay will accumulate and introduce significant reduction in system data throughput.

For instance, let's construct a point to point GALS system, where the periods of two local clocks *LClkTx* and *LClkRx* satisfy condition (1):



$$T_{ReqTx} = N \cdot T_{LCLKTx} , \quad (1)$$

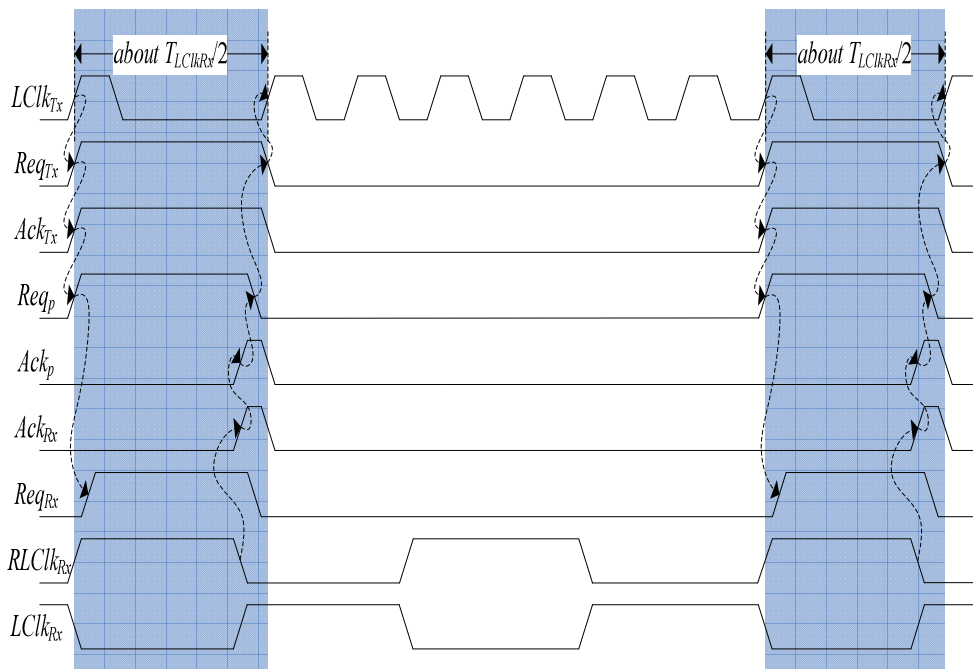
and the requests are asserted in the transmitting module every  $N$  cycles of  $LCLKTx$  as shown in the condition (2):

$$T_{ReqTx} = N \cdot T_{LCLKTx} . \quad (2)$$

Since  $Req$  is asserted in the transmitting module running at an independent clock  $LCLKTx$ , its arrival time in each cycle of  $LCLKRx$  distributes randomly. If a  $Req+$  occurs simultaneously with  $RClk+$  in the receiving module, which is of cause possible, it needs to wait for half cycle of  $LCLKRx$  to be granted and during that period  $LCLKTx$  is paused and stretched in off-phase. When  $RClk-$  occurs in half cycle of  $LCLKRx$ ,  $Req$  is granted and  $LCLKTx$  is also released. Consequently the next rising edge of  $LCLKTx$  and  $LCLKRx$  are synchronized to occur at almost the same time. On one side the next  $Req+$  will be asserted in  $(N-1)$  cycles of  $LCLKTx$  in the transmitting module. However, on the other side, according to condition (1), the next  $RClk+$  will also be asserted in one and a half cycles of  $LCLKRx$  in the receiving module. Then the extension in  $Req$  as well as the pause in  $LCLKTx$  will occur again.

That means in any systems satisfying conditions (1) and (2), once a  $Req+$  occurs simultaneously with  $RClk+$  in the receiving module, all the following  $Req+$  will be synchronized to occur at the same time with  $RClk+$ . So each  $Req$  will be extended for half cycle of  $LCLKRx$  due to the closed RGW of  $LCLKRx$ , and  $LCLKTx$  will also be paused in off-phase for half cycle of  $LCLKRx$  periodically.

Each time when  $LCLKTx$  is paused, its off-phase will be stretched for a period of  $(T_{LCLKRx}/2 - T_{LCLKTx})$ . In fact it is an additional delay for data transmission, because the data is always delayed the same period as  $LCLKTx$ . With the increase of the amount of data transferred, all the delay will be accumulated and finally lead to a large latency in the transmitting module. This is the worst case caused by the closed RGW of  $LCLKRx$  in data transmission. As an example, a sample waveform under this circumstance is depicted in Figure 36, where  $N = 7$ .



**Figure 36: Waveform for the system satisfying (1) & (2)**

Based on conditions (1) and (2), the reduction in data transmission result from the latency of paused clock is deduced as below:



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

Equation (3) shows that the exact percentage is determined by the value of  $N$ , and the limit in  $RTx$  with the increase in the value of  $N$  is:

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

Equation (4) illustrates that the latency of pausable clock generator in the receiving module could lead to up to one-third reduction in data transmission in the worst case.

Similar to the discussion on the receiving module, the multi-port schemes shown in Figure 34 and Figure 35 may cause higher reduction in data transmission, and the percentage depends on the maximal ratio of  $TLCIkRx$  to  $TLCIkTx$  which satisfy above condition (1).

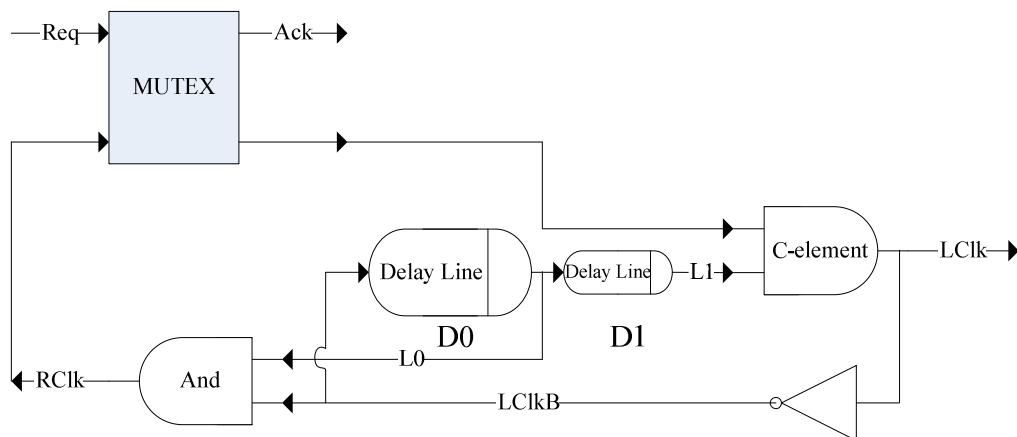
### Modification on Clock Generator

In fact, the reduction in data-throughput result from pausable clock generators does not only happen in communication between a poll-type output port and a demand-type input port, similar analysis can be easily applied on the other port configurations, such as a poll-type output port with a demand-type input port or a poll-type output port with a poll-type input port. The only exception occurs when both input and output ports are demand-type. In that situation  $LClkTx$  and  $LClkRx$  is paused before data transmission, so the latency to pause clocks has no influence on data throughput.

The reduction can be minimized by careful co-design on both sides of the transmitting module and the receiving module, as what have been done in [MT01, GU06]. However, for GALS systems, the modularity design property to integrate numerous kinds of pre-designed IP blocks is very important. So any potential reduction introduced by pausable clock generators need to be avoided.

### Proposed Scheme

To minimize data-throughput reduction caused by pausable clock generators, a simple solution is to widen RGW within each clock cycle. Figure 37 presents a modified scheme in this direction.



**Figure 37: Modified pausable clock generator**

Different from the scheme in Figure 32, the single delay-line is divided into two cascaded ones  $D0$  and  $D1$ , and request clock  $RCIk$  is now generated by an AND operation between  $LClkB$ , the phase

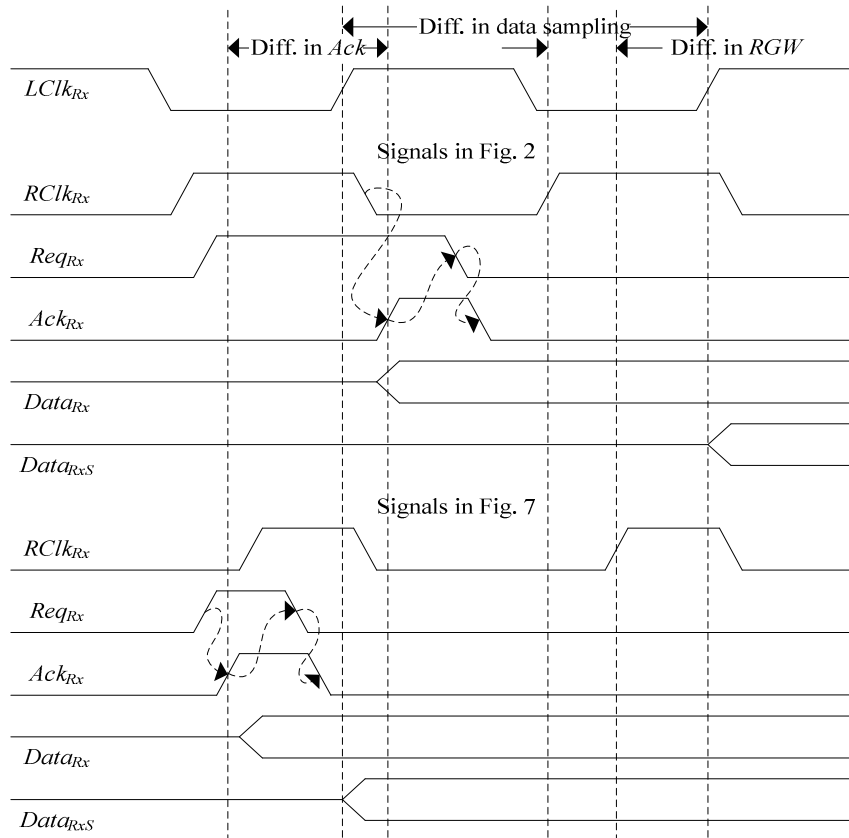


inversion of  $LClk$ , and  $L0$ , the output signal of  $D0$ . Moreover, the delay times of  $D1$  and  $D0$  are configured to satisfy conditions (5) and (6):

$$d_{D0} = T_{LCIk} / 2 - (d_{D1} + d_{C-element} + d_{Invertor}) , \quad (5)$$

$$d_{D1} = d_{And} + d_{MUTEX} . \quad (6)$$

Let's analyze the modified scheme of pausable clock in detail. Since  $RClk$  is asserted after both  $LCIkB$  and  $L0$  are high and is de-asserted as soon as  $LCIkB$  turns low, its on-phase period within each cycle of  $LCIk$  is the sum of propagation delays of the MUTEX element, the Muller-C element, the inverter and the AND-gate. If such a delay is shorter than the half period of  $LCIk$ , the RGW in this modified scheme will be wider than that in Figure 32. Considering the delay of logic gates in typical CMOS processes is on a picoseconds time scale, this requirement is normally easy to achieve. For example, if the period of  $LCIk$  is 10ns and the summation of above delay is 1.5ns, the RGW is 8.5ns in Figure 36 while only 5ns in Figure 31. Then the probability for each  $Req$  to introduce one- $LCIkRx$ -cycle latency in data path drops from 50% to 15%, providing a uniform distribution of the arrival time of  $Req$  within each cycle of  $LCIk$ . Figure 38 illustrates a comparison between above schemes in RGW,  $Ack$  latency and data sampling.



**Figure 38: Comparison in Ack latency and data sampling**

Delay-line  $D1$  is required in Figure 36 between delay line  $D0$  and the Muller-C element to remain  $LCIk$  with 50% duty cycle. The maximal delay path from  $LCIk+$  to  $LCIk-$  is

$$Inv- \rightarrow D_0- \rightarrow D_1- \rightarrow MullerC$$

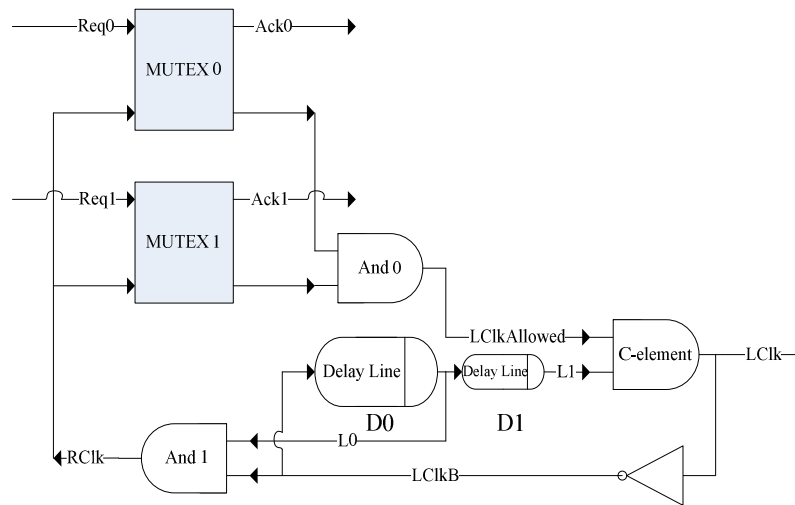
And the maximal delay path of  $LCIk-$  to  $LCIk+$  is



$$Inv \rightarrow D_0 \rightarrow D_1 \rightarrow MullerC$$

As shown in Equation 6, the delay of  $D_1$  matches the total delay of the AND-gate and the MUTEX element; therefore both of the paths are balanced.

As shown in Figure 39, it is simple to extend above pausable clock generator to support multi-port communications, where an array of MUTEX elements replaces the single MUTEX element to process concurrent requests. Ascribing to the wider RGW, the probability that several requests are extended in this scheme will be much lower than that in the previous schemes in Figure 34 & Figure 35.

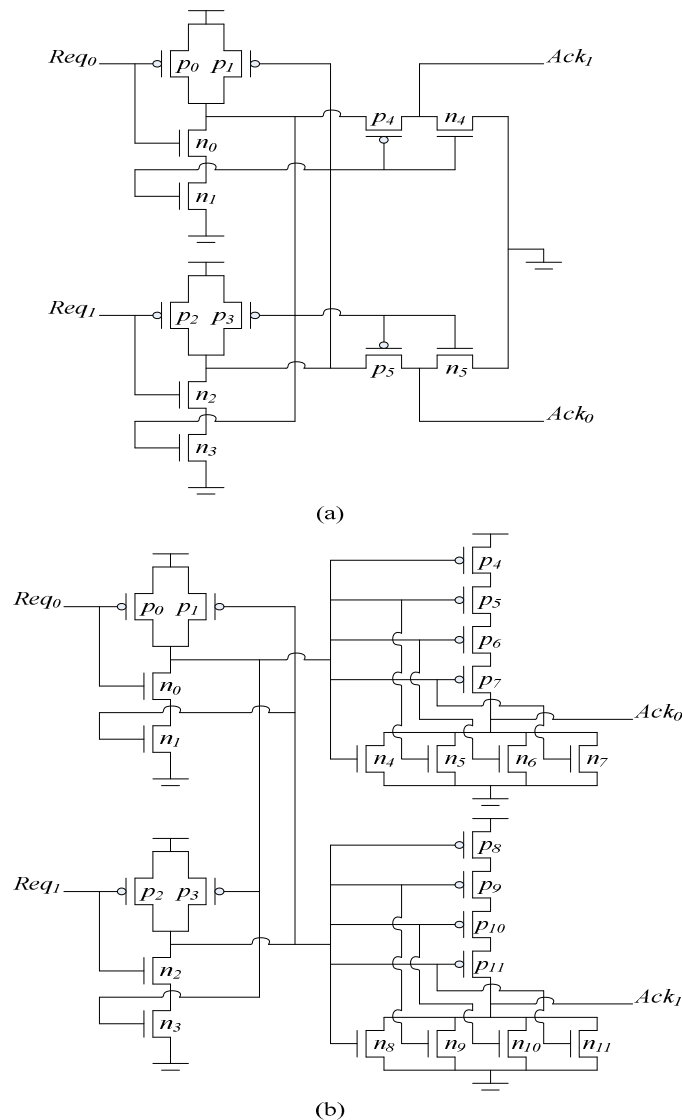


**Figure 39: Modified multi-port pausable clock generator**

### Optimization in Delay-Lines

Equation (6) shows that the length of  $D_1$  should match the delay time of the MUTEX element. For a MUTEX element, such a resolution time varies with the difference in arrival times of  $Req_+$  and  $LClk_+$ . Therefore, the maximal resolution time in theory should be taken into consideration for  $D_1$ . Unfortunately research in [KI02] illustrates that the resolution time of MUTEX element increases with the decreasing interval of  $Req_+$  and  $LClk_+$ , and no upper bound exists in its value. A practical solution is to use the accurate enough interval for normal systems to calculate the resolution time of MUTEX.

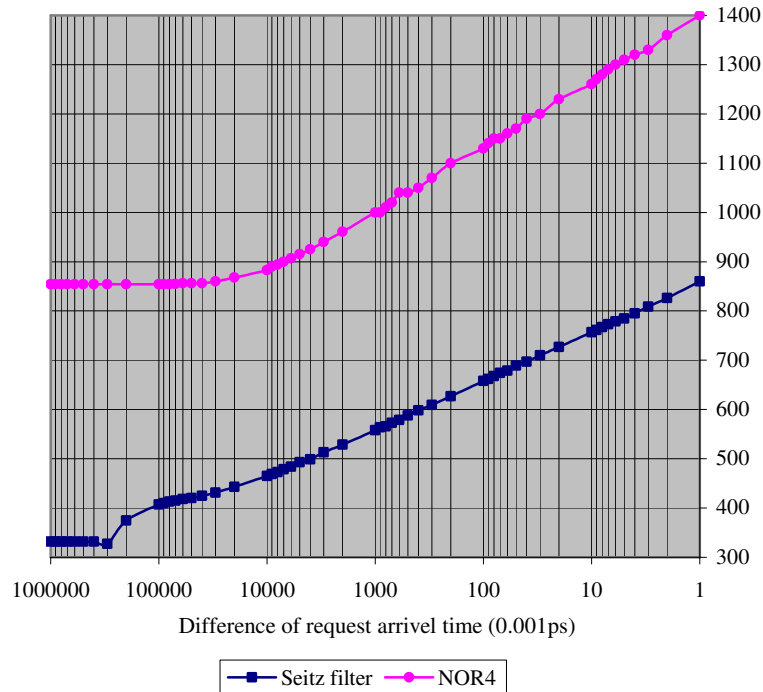
To investigate the property of resolution time of MUTEX element in mainstream CMOS processes, two different structures shown in Figure 40 are implemented and simulated in transistor level in IHP 0.13- $\mu\text{m}$  CMOS process with Cadence Spectre. Figure 40(a) is the widely utilized Seitz structure, and Figure 40(b) is an alternative one reported in [KE97] which can be implemented in standard cell.



**Figure 40: Two structures of MUTEX element**

All PMOS transistors have  $W/L = 12\lambda/2\lambda$  and all NMOS transistors have  $W/L = 6\lambda/2\lambda$  in above circuits. A 12fF capacitance is used as loading condition for the simulation, which is equivalent to a fanout of four inverters (FO4) in IHP 0.13- $\mu\text{m}$  CMOS process. The rising time of both inputs is set to 20ps, the typical value for 0.13- $\mu\text{m}$  CMOS process. Additionally, the resolution time is measured from the arrival time of the first signal to output 90% of peak amplitude. Finally, the simulation results of the resolution times of MUTEX implemented in two structures versus the interval of  $Req_+$  and  $LClk_+$  from 10-3ps to 103ps are shown in Figure 41.

It can be seen that for the interval of 10-3ps, which is an accurate enough time scale for normal systems, the resolution time of the structure in Figure 40(a) is measured as about 0.85ns. Therefore to match the timing requirement in Equation (6), the propagation delay of  $D1$  can be configured as 1ns. On one side this delay-line minimizes the probability that the MUTEX element introduces a larger latency than its propagation delay, and on the other side it maximizes the RGW in the modified scheme of pausable clock. From this point of view, it represents the optimal length of  $D1$  for the Seitz implementation of MUTEX element in 0.13- $\mu\text{m}$  CMOS process. Similarly for the MUTEX structure in Figure 40(b), the resolution time corresponding to 10-3ps is about 1.4ns as shown in Figure 41, and the optimal propagation delay of  $D1$  can be configured as long as 1.5ns.



**Figure 41: Resolution times of MUTEX in two structures**

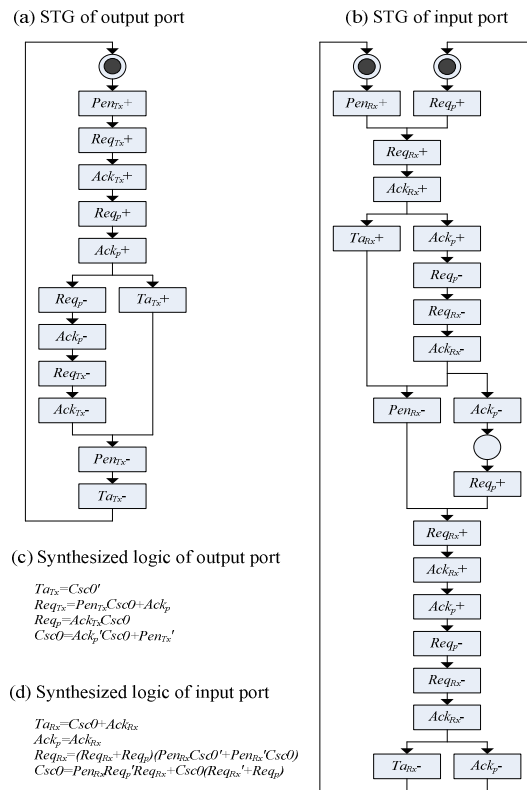
Figure 40(b) employs NOR-gates to construct inverters with low threshold as metastability filter. Simulation in IHP 0.13- $\mu\text{m}$  1.2v CMOS process demonstrates that the threshold value of inverter-connected NOR-gate is 0.47v (40%VDD), and the metastability level of bi-stable latch is about 0.61v (50%VDD). Experiments show that this approach can avoid meta-stable output safely. However, a weakness of the use of NOR-gate as metastability filter lies in its longer resolution time comparing to that with Seitz filter. The heavy loading capacitance from 4-input NOR-gates make it much slower for bi-stable latch to recover from meta-stable state. Moreover, there are 4 cascaded PMOS transistors in the NOR-gate filter (p4, p5, p6, p7 for Ack0 and p8, p9, p10, p11 for Ack1) while only 2 cascaded ones in Seitz filter (p3, p5 for Ack0 and p1, p4 for Ack1), therefore the charge current in NOR-gate filter is much smaller than that in Seitz filter. It also contributes to the longer resolution time.

When  $Req+$  and  $LClk+$  arrive so close with each other that the MUTEX element leads to longer latency than the propagation delay of  $D1$ , the off-phase of  $LClk$  will be stretched. This circumstance also occur in the scheme in Figure 32, while, due to the minimized length in  $D1$ , it may happen in the modified scheme in a higher probability. However, the stretched clock signal results in no mal-function in system, and in this aspect this scheme is more reliable than that shown in Figure 35.

Finally, as to the length of  $D0$ , it is easy to be optimized based on the local clock period and the optimal length of  $D1$  using Equation (5).

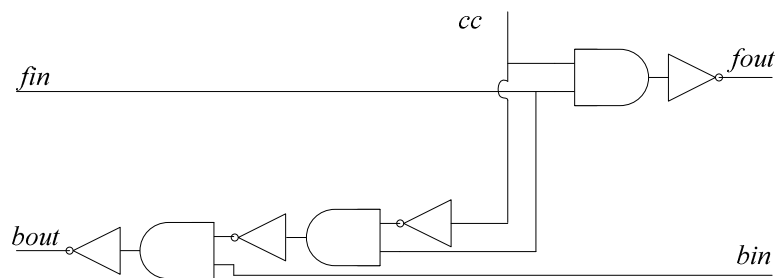
### Experimental Results

To illustrate the reduction on data throughput caused by pausable clock generator, the point to point GALS system described previously is designed, synthesized and simulated in gate level in IHP 0.13- $\mu\text{m}$  CMOS library. The signal transition graphs (STG) for the demand-type output port and poll-type input port and the synthesized results by Petrify are presented in Figure 42.



**Figure 42: STG and synthesized logic of I/O ports**

The simplified programmable delay slice shown in Figure 43 is used to construct delay-lines, and its propagation delay is measured as about 0.259ns. In the receiving module the delay line is made up of 32 cascaded delay slices, and its clock period  $TLCIkRx$  is fixed to 16.57ns. In the transmitting module, the delay line is programmed to generate a serial of different clock period  $TLCIkTx$ .



**Figure 43: Delay slice implementation**

The exact value of  $TLCIkTx$  is programmed to meet the Equation (1), and it varies with the value of parameter  $N$ . In this experiment, three different values of  $N$  are tested, which are  $N = \{7, 8, 10\}$ , and then  $TLCIkTx$  is programmed as 4.14ns, 3.55ns and 2.76ns, respectively.

And according to Equation (2), parameter  $N$  also determines the request period  $TReqTx$  in the transmitting module, where the output port should be enabled every  $N$  cycles of  $LCIkTx$  to assert a request. In the receiving module, the input port can process a request in each cycle of  $LCIkRx$ .

The structure in Figure 40(b) is employed to implement the MUTEX element in standard cell, and the length of delay line  $D1$  is optimized to be 6 cascaded delay slices, which is about 1.554ns in



propagation delay. Therefore the RGW in Figure 32 is  $T_{LCIkRx}/2 = 8.285ns$ , while it is  $T_{LCIkRx} - TD1 = 15.016ns$  in Figure 37. It is obvious that the RGW of pausable clock generator is widened.

To see the influence of RGW in system data throughput, the usually adopted pausable clock generator in Figure 32 and the modified scheme in Figure 37 are both adopted in the receiving module for a comparison. At each value of  $N$ , the scheme in Figure 32 is firstly used to transfer 32 data, and then it is replaced by the modified scheme to run simulation within the same duration.

Table 4 presents the amount of data transfer accomplished in the use of modified scheme and the percentage of improvement in data throughput comparing to the scheme in Figure 32. The maximal improvement in theory is also given for reference. It can be seen that the modified scheme leads to higher throughput, and the throughput difference between two schemes is becoming remarkable with the increase of  $N$ .

	$N=7$ $T_{ClkRx}=4.14ns$	$N=8$ $T_{ClkRx}=3.55ns$	$N=10$ $T_{ClkRx}=2.67ns$
<i>Throughput of the classical solution</i>	32	32	32
<i>Throughput of our improved solution</i>	36	37	38
<b>Difference</b>	<b>12.5%</b>	<b>15.63%</b>	<b>18.75%</b>
<i>Diff. in theory</i>	$1/7$	$1/6$	$1/5$

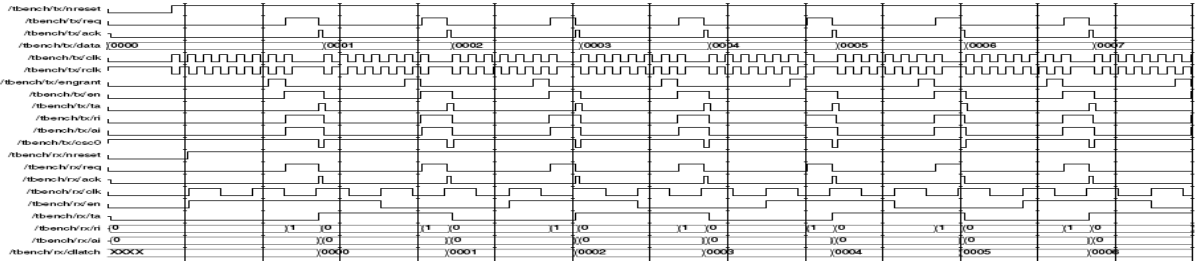
**Table 4: Comparison in throughput of two schemes**

As an example, Figure 44 shows waveform fragments from 0ns to 280ns simulating in gate level with  $N = 7$ . In Figure 44(a) each  $Req+$  occurs simultaneously with  $RClk+$  in the receiving module, and then all  $Req$  are extended for about half period of  $LCIkRx$  to be granted. As a result,  $LCIkTx$  is also paused and stretched periodically. Since  $T_{LCIkTx} = T_{LCIkRx}/4$  when  $N = 7$  in Equation (1), such a half- $LCIkRx$ -cycle pause results in an additional cycle of  $LCIkTx$  consumed in the transmitting module. All these additional cycles accumulate and eventually lead to only 6 data transferred in Figure 43(a). However, in Figure 44(b), due to the widened RGW, few extensions in  $Req$  as well as pause in  $LCIkTx$  happen, and as a result there are 7 data transferred in the same period.

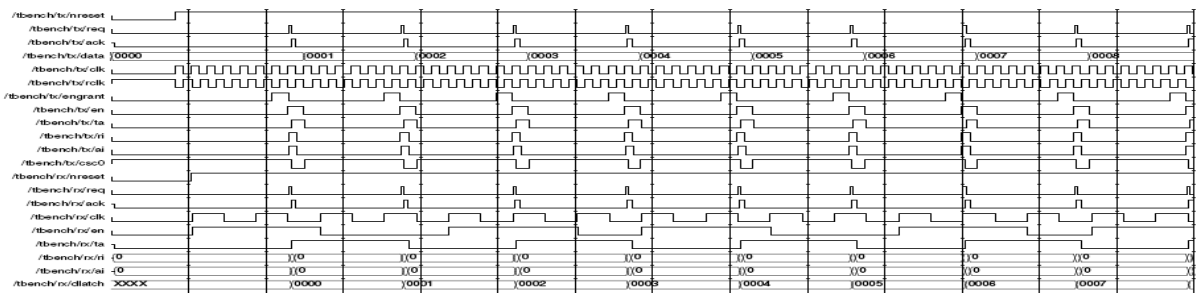
The multi-port pausable clock generators are also synthesized in IHP 0.13- $\mu m$  CMOS library with Synopsys Design Compiler. Table 5 presents a comparison in power consumption and area among different schemes with 8 ports. It can be seen that the scheme in Figure 35 calls for much larger consumption than the others, because there are more amount of delay lines employed to minimize the probability of metastability in  $Ack$  signals. The proposed scheme consumes little additional overheads comparing to the usually adopted multi-port clock generator shown in Figure 34.

	Length of Delay-line (Num. of delay slice)		Area( $\mu m^2$ )	Power( $\mu w$ )
	D0	D1		
<i>Standard scheme</i>	32		1669	370
<i>Scheme from [3]</i>	32	16	2870	760
<i>Our novel scheme</i>	24	8	1790	387

**Table 5: Performance comparison among multi-port schemes**



(a)



(b)

**Figure 44: Simulation waveform (0-280ns) (a) with the standard scheme and (b) with our improved scheme**

## 5.4 CONCLUSIONS

In the previous sections we have tackled some possible improvement of the GALS interfaces. GALS interfaces based on synchronizers and FIFOs have a long history and we see not much scope for further improvements. On the other hand we have shown that some improvements of the pausable clocking scheme are still possible. The main improvements are related to ability of such interfaces to handle bursty data transfer. In addition to that, we propose the improvement of the clock generator for pausable clocking. This gives a hope that such GALS interfaces can be used for much wider spec of applications. As a result of this we can expect that this approach can be successfully utilized in particular for systems targeting low-power and low-EMI features.



---

## APPENDIX

---

### A VHDL HEADER FOR GALS OCP ADAPTER

The header for VHDL files describing OCP master adapter is as following (this includes limited set of supported OCP operations as described in the report):

```
entity OCP_wrapper_master is
  generic (
    addr_width : integer;
    data_width : integer;
    atomiclength_width : integer;
    blockheight_width : integer;
    blockstride_width : integer;
    burstlength_width : integer;
    connid_width: integer;
    threads : integer );

  port(
    -- nrst
    nrst      : in  std_logic; -- reset
    -- basic signals
    Clk       : in  std_logic; -- clock input
    EnableClk : in  std_logic; -- enable OCP clock
    MAddr     : in  std_logic_vector(addr_width-1 downto 0); -- transfer address
    MCmd      : in  std_logic_vector(2 downto 0); -- transfer command
    MData     : in  std_logic_vector(data_width-1 downto 0); -- write data
    MDataValid : in  std_logic; -- write data valid
    MRespAccept: in  std_logic; -- master accepts response
    SCmdAccept : out std_logic; -- slave accepts transfer
    SData      : out std_logic_vector(data_width-1 downto 0); -- read data
    SDataAccept: out std_logic; -- slave accepts write data
    SResp      : out std_logic_vector(1 downto 0); -- transfer response
    -- burst signals
    MBurstLength: in  std_logic_vector(burstlength_width-1 downto 0); -- burst length
    MBurstPrecise: in  std_logic; -- given burst length in precise
    MBurstSeq   : in  std_logic_vector(2 downto 0); -- address sequence of burst
    MBurstSingleReq: in  std_logic; -- burst uses single request/multiple data protocol
    MReqLast    : in  std_logic; -- last request in burst
    SRespLast   : out std_logic; -- last response in burst
    -- thread extensions
    MThreadBusy: in  std_logic_vector(threads-1 downto 0); -- master thread busy
    MThreadID  : in  std_logic_vector(threads-1 downto 0); -- request thread identifier
    SThreadBusy: out std_logic_vector(threads-1 downto 0); -- Slave request thread busy
    SThreadID  : out std_logic_vector(threads-1 downto 0); -- Response thread identifier
    -- GALS interface data signal
    MDataASYNC : out std_logic_vector(data_width-1 downto 0); -- write data asynchronous side
    SDataASYNC : in  std_logic_vector(data_width-1 downto 0); -- read data asynchronous side
    MControlASYNC: out std_logic_vector (threads+threads+addr_width+burstlength_width+10 downto 0);
    SControlASYNC: in  std_logic_vector (threads + threads + 4 downto 0);
    -- GALS interface control signal
    Pen        : out std_logic;
    Ta         : in  std_logic
  );
end OCP_wrapper_master;
```



# GALAXY

GALS InterfACE for CompleX Digital  
SYstem Integration

Confid. Level: Public  
Date : 24/12/2008  
Issue: 2

Similarly the header for the OCP slave adapter compatible with pausable clock interfaces is given in the following text:

```
entity OCP_wrapper_slave is
  generic (
    addr_width : integer;
    data_width : integer;
    atomiclength_width : integer;
    blockheight_width : integer;
    blockstride_width : integer;
    burstlength_width : integer;
    connid_width: integer;
    threads : integer );

  port(
    -- nrst
    nrst      : in  std_logic; -- reset
    -- basic signals
    Clk       : in  std_logic; -- clock input
    EnableClk : in  std_logic; -- enable OCP clock
    MAddr     : out std_logic_vector(addr_width-1 downto 0); -- transfer address
    MCmd      : out std_logic_vector(2 downto 0); -- transfer command
    MData     : out std_logic_vector(data_width-1 downto 0); -- write data
    MDataValid : out std_logic; -- write data valid
    MRespAccept : out std_logic; -- master accepts response
    SCmdAccept : in  std_logic; -- slave accepts transfer
    SData     : in  std_logic_vector(data_width-1 downto 0); -- read data
    SDataAccept : in  std_logic; -- slave accepts write data
    SResp     : in  std_logic_vector(1 downto 0); -- transfer response
    -- burst signals
    MBurstLength: out std_logic_vector(burstlength_width-1 downto 0); -- burst length
    MBurstPrecise : out std_logic; -- given burst length in precise
    MBurstSeq   : out std_logic_vector(2 downto 0); -- address sequence of burst
    MBurstSingleReq : out std_logic; -- burst uses single request/multiple data protocol
    MReqLast   : out std_logic; -- last request in burst
    SRespLast  : in  std_logic; -- last response in burst
    -- thread extensions
    MThreadBusy : out std_logic_vector(threads-1 downto 0); -- master thread busy
    MThreadID   : out std_logic_vector(threads-1 downto 0); -- request thread identifier
    SThreadBusy : in  std_logic_vector(threads-1 downto 0); -- Slave request thread busy
    SThreadID   : in  std_logic_vector(threads-1 downto 0); -- Response thread identifier
    -- GALS interface data signal
    MDataASYNC : in  std_logic_vector(data_width-1 downto 0); -- write data asynchronous side
    SDataASYNC : out std_logic_vector(data_width-1 downto 0); -- read data asynchronous side
    MControlASYNC : in  std_logic_vector (2*threads + addr_width+burstlength_width+10 downto 0);
    SControlASYNC : out std_logic_vector (2*threads + 4 downto 0);
    -- GALS interface control signal
    Pen       : out std_logic;
    Ta       : in  std_logic
  );
end OCP_wrapper_slave;
```