Request-driven GALS Technique for Datapath Architectures

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

(Dr.-Ing.)

genehmigte Dissertation

vorgelegt von

Dipl.-Ing., M.Sc.El.Eng.

Miloš Krstić

geboren am 18. Oktober 1973 in Niš (Serbien und Montenegro)

Gutachter: Prof. Dr.-Ing. Rolf Kraemer

Gutachter: Prof. Dr.-Ing. Heinrich-Theodor Vierhaus

Gutachter: Prof. Christian Piguet

Tag der mündlichen Prüfung: 07.02.2006

Acknowledgements

While working in the IHP's System Design department I had support from many colleagues. At this place I want to thank those who helped me with the thesis:

First of all, I want to thank to my supervisor Prof. Rolf Kraemer who supported my work on this topic over the past years.

I especially want to thank Dr Eckhard Grass who guided and supervised my activities in the area of asynchronous circuit design. In the mutual discussions we had, we defined many solutions described in the following text. Without his support this thesis would not be possible.

Dr Alfonso Troya, Dr Koushik Maharatna, and Ulrich Jagdhold were part of our great team that has developed a synchronous WLAN baseband processor. Many thanks for their support and friendship.

I also want to thank Maxim Piz for his support in performing MATLAB simulations.

Special thanks to my proofreaders: Dr Eckhard Grass, Dr Michael Methfessel, Christoph Wolf and Daniel Dietterle.

I am very grateful to Alexandra Julius, Christian Stahl, Prof. Wolfgang Reisig and Dr Frank Winkler from Humboldt University in Berlin. I really enjoyed working together with them on GALS topics, especially in the area of formal analysis and during the development of the externally driven GALS wrapper. Furthermore, I want to acknowledge Kim Fahrion for developing the 3DConverter tool and Prof. Mark Greenstreet, University of British Columbia - Vancouver, who helped me with many valuable comments.

I want to thank my parents and sister for their support during all these years.

Finally, gratitude and love go to my wife Sanja and son Boris, for their love and support.

Contents

Abstract . . . . **ix**

Zusammenfassung . . . . **xi**

1. Introduction **1**

1.1 Design Challenges for Wireless Communication Systems 1

1.2 GALS as a Solution for the System Integration Problem 3

1.3 Structure of the Thesis 4

2. Related Work – from the Two-Flop Synchroniser to GALS **7**

2.1 Introduction 7

2.2 System Integration Strategies 8

2.2.1 Standard Synchronisers 9

2.2.2 Adaptive Synchronisation 10

2.2.3 FIFO Synchronisation 11

2.2.4 GALS Systems 12

2.3 Power Saving with GALS 18

2.4 Open Questions and Directions for Further Research 19

3. Proposed Novel GALS Architecture **21**

3.1 Introduction 21

3.2 Motivation and General Principles 21

3.3 System Structure 22
3.4 Request Driven Technique with External Clocking 25
3.5 Potential Gain of the Novel GALS Architecture 26
4. Hardware Architecture of the GALS Wrapper 31
4.1 Introduction 31
4.2 Overall Structure of the Asynchronous Wrapper 31
4.2.1 Pausable Clock 34
4.2.2 Clock Control Unit. 36
4.2.3 Time-out Generation 37
4.2.4 Input Port 38
4.2.5 Output Port 44
4.2.6 Mutual Exclusion Element 47
4.3 Formal Analysis of the Asynchronous Wrapper 48
4.4 Externally Clocked Asynchronous Wrapper 48
5 GALS Application in Wireless Communication Systems. 53
5.1 Introduction 53
5.2 Baseband Processor Compliant to IEEE 802.11a Standard 54
5.3 GALS Partitioning 57
5.3.1 Transmitter Dataflow Organisation 60
5.3.2 Receiver Dataflow Organisation 60
5.4 Power Saving Mechanisms. 62
5.5 Important Details – GALS Extensions 64
5.5.1 Activation Interface for Blocks Rx_1 and Rx_2 64
5.5.2 Specific Asynchronous Fork 65
5.5.3 Token Alignment 66
5.5.4 Rate Adaptation 68
5.5.5 Token Synchronisation in the Transmitter 68
5.6 Synchronous-Asynchronous Interfaces 69

5.6.1 Synchronous to Asynchronous Communication 69

5.6.2 Synchronous to Asynchronous Communication with a Continuous Synchronous Data Stream 69

5.6.3 Asynchronous to Synchronous Communication 70

6 Design for Testability in GALS Systems 73

6.1 Introduction 73

6.2 Test Techniques for GALS Systems 74

6.3 Proposed BIST Architecture 75

6.4 Implementation of the BIST in the Baseband Processor 79

7 Implementation and Evaluation of GALS Systems 85

7.1 Introduction 85

7.2 Design Flow 85

7.3 System Integration with GALS 89

7.4 Conceptual GALS Design Framework 90

7.5 Asynchronous Wrapper Implementation 92

7.6 Experimental GALS Chip 94

7.7 GALS Baseband Processor Implementation 96

7.7.1. Evaluation of Synchronous and GALS Baseband Processor 99

8. Experimental Results 103

8.1 Introduction 103

8.2 Functional Verification of the GALS Baseband Processor 103

8.3 Power Measurement 105

8.4 Supply Noise Measurement 106

9. Conclusions 109

9.1 Achieved Results 109

9.2 Request-Driven GALS as a Solution – Pros and Cons 110

9.3 Future Work 110

10. References 113

Acronyms and Symbols 121

Curriculum Vitae **125**

Abstract

In this thesis a novel Globally Asynchronous Locally Synchronous (GALS) technique applicable to wireless communication systems and generally to datapath architectures is presented. The proposed concept is intended for point-to-point communication with very intensive but bursty data transfer between the system blocks.

The GALS technique introduced here is based on a request-driven operation of locally synchronous modules. The key idea behind this request-driven approach is that a module can use the input request signal as its clock while receiving a burst of data. Inactivity of the request line is detected with a special time-out circuitry. When time-out occurs, clocking of the locally synchronous module is handed over to a local ring oscillator or an external clock source. This allows emptying of internal pipeline stages of a locally synchronous module after a burst of data was received.

Based on this concept, a practical hardware implementation of an asynchronous wrapper is proposed. The asynchronous wrapper consists of several components, with different complexity and structure. The internal clocking circuitry is based on a tunable ring oscillator that actually consumes most of the area of the asynchronous wrapper. The main wrapper components are input and output ports, which are developed as an Asynchronous Finite State Machine (AFSM). Those ports perform hazard-free handshake operations between different GALS blocks and control safe transfer of the data. The complete asynchronous wrapper uses a few thousand gates, which is acceptable in comparison with an average synchronous block size of a few hundred thousand gates. Furthermore, we discuss an alternative asynchronous wrapper architecture based on external clocking.

The developed wrapper is applied to the design of an IEEE 802.11a compliant baseband processor with the aim to alleviate the problems of system integration, timing closure, clock skew, power consumption and electro-magnetic interference (EMI). The baseband processor is

partitioned into a set of different GALS blocks. The criteria for GALS partitioning were power saving and natural architectural boundaries between the different blocks. Locally synchronous modules were extended with adequate asynchronous wrappers. In order to control the complex dataflow between the blocks, some additional asynchronous blocks for providing join, fork, and data-rate adaptation functions between the GALS blocks were proposed. Additionally, it was needed to guarantee certain performance levels for communication with the synchronous environment. For that reason, synchronous-to-asynchronous and asynchronous-to-synchronous interface blocks were proposed. For testing purposes, our GALS baseband processor is fitted with Design for Testability (DFT) logic based on Built-in Self-Test (BIST).

The complete baseband processor including GALS wrappers was integrated, synthesized, layouted, and finally fabricated. Implementation details are described in order to evaluate the advantages of the proposed concept. Furthermore, a design-flow for GALS systems is proposed. Finally, results of the measurements are presented and discussed. The GALS design was compared with a synchronous version of the baseband processor with implemented clock-gating as power saving technique. In our experimental setup we have measured a 1% reduction in dynamic power consumption, 30% reduction in instantaneous supply voltage variations, and 5 dB reduction in spectral noise.

Keywords: *GALS, System Integration, Asynchronous design, EMI, BIST*

Zusammenfassung

In dieser Doktorarbeit wird eine global asynchrone, lokal synchrone (GALS) Technik, die auf drahtlose Kommunikationssysteme und im Allgemeinen auf Datenpfad-Architekturen anwendbar ist, dargestellt. Das vorgeschlagene Konzept ist für Punkt-zu-Punkt organisierte Strukturen mit sehr intensiver, aber Block organisierter Datenübertragung zwischen den System-Blöcken vorgesehen.

Die GALS-Technik, die in dieser Arbeit eingeführt wird, basiert auf einem Request-driven Betrieb der lokalen synchronen Blöcke. Die Schlüsselidee hinter diesem Request-driven Konzept ist, dass ein Modul das Request-Signal als Taktsignal beim Empfangen eines Blocks von Daten benutzen kann. Untätigkeit des Request-Signals wird mit einer speziellen Time-out Schaltung ermittelt. Wenn über einen bestimmten Zeitraum kein Request-Signal auftritt, wird die Erzeugung des Taktsignales für den lokalen synchronen Block von einem lokalen Ringoszillator oder einem externen Takt übernommen. Dies dient dem Leeren interner Pipeline-stufen des synchronen Blocks.

Basierend auf diesem Konzept, wird eine praktische Hardware-Realisierung eines asynchronen Wrappers vorgeschlagen. Der asynchrone Wrapper besteht aus einigen Bestandteilen mit unterschiedlicher Komplexität und Struktur. Ein stimmbarer Ringoszillator taktet den lokalen synchronen Block. Dieser Ringoszillator verbraucht den größten Teil der Fläche des asynchronen Wrappers. Die Haupt-bestandteile sind Eingangsschaltung und Ausgangsschaltung, die als asynchrone endliche Zustands-automaten (AFSM) entwickelt wurden. Diese Schaltungen sollen die Hazard-freie Kommunikation zwischen unterschiedlichen GALS-Blöcken durchführen und die sichere Übertragung der Daten steuern. Im Allgemeinen benötigt der vollständige asynchrone Wrapper wenige tausend Gatter, die im Vergleich mit einer durchschnittlichen synchronen Blockgröße von einigen hunderttausend Gattern annehmbar ist. In der Arbeit besprechen wir auch eine alternative asynchrone Wrapper-Architektur, die auf externer Taktung basiert.

Die entwickelte Wrapper-Struktur wird auf das Design eines IEEE 802.11a kompatiblen Basisbandprozessors angewendet, um die Probleme System integration, Timing-closure, Clock-

skew, Leistungsaufnahme und elektromagnetische Störung (EMS) zu vermindern. Der Basisbandprozessor wird auf eine Anzahl unterschiedlicher GALS-Blöcke verteilt. Die Kriterien für die Aufteilung waren Energieeinsparung und natürliche Grenzen zwischen den unterschiedlichen Blöcken. Dann wurden lokal synchrone Blöcke mit passenden asynchronen Wrappern erweitert. Um den komplizierten Datenfluss zwischen den Blöcken zu steuern, werden einige zusätzliche Blöcke vorgeschlagen, die die Datenraten-Anpassung, das Aufteilen eines Datenstomes in mehrere Datenströme und die Kombination mehrerer Datenströme in einen einzigen realisieren. Zusätzlich war es erforderlich, bestimmte Kommunikationmuster mit der synchronen Umgebung zu garantieren. Aus diesem Grund wurden synchron-zu-asynchrone und asynchron-zu-synchrone Schnittstellen-Blöcke eingeführt. Für den Test wurde der GALS-Basisbandprozessor um Design-für-Testability (DFT) Logik erweitert, die auf einem eingebauten Selbsttest (BIST) basiert.

Der komplette Basisbandprozessor einschließlich GALS Wrapper wurde integriert, synthetisiert, und schließlich gefertigt. Implementierungsdetails werden hier beschrieben, um die Vorteile des vorgeschlagenen Konzeptes darzustellen. Zusätzlich wird eine Entwurfsmethodik für GALS-Systeme vorgeschlagen. Schließlich werden die Resultate der Messung dargestellt und diskutiert. Das GALS-Design wird mit einer synchronen Version des Basisbandprozessors verglichen, die Clock-Gating zur Energieeinsparung verwendet. Unsere Messungen ergaben eine Verringerung des dynamischen Energieverbrauchs um 1%, eine Verringerung von Versorgungsspannungs-schwankungen um 30% und eine Reduktion des spektralen Rauschens um 5 dB.

Schlagwörter: *GALS, System Integration, asynchroner Entwurf, EMI, BIST*

Chapter 1

Introduction

1.1 Design Challenges for Wireless Communication Systems

Designing modern wireless communication systems is a very challenging task. The complexity of digital systems grows enormously, as can be noticed from the technology roadmap [ITRS03, MED02] in Table 1.1. We can conclude from the table that this trend will be continued in the following years. The increasing demands of wireless applications create several problems for system design and integration. The following issues will have the main importance in future: integration of complex systems, timing closure including clock generation and control, system noise characteristic, and power consumption for mobile applications.

Table 1.1 Technology roadmap from 1999 to 2011

Property \ Year	1999	2001	2005	2011
CMOS process [μm]	0.18	0.15	0.1	0.05
Transistors on chip [Mtrans/cm^2]	7	14	41	247
On-chip clock [GHz]	1.25	1.77	3.5	10
Off-chip clock [GHz]	0.48	0.722	1.035	1.54
Power dissipation (handheld systems) [W]	1.4	1.7	2.4	2.2
Vdd [V]	1.5	1.2	0.9	0.5

When a complex digital system is designed, system integration and timing closure are very important tasks. A communication system usually contains blocks operating at different frequencies and even different supply voltages. Integration of such blocks requires advanced design techniques. An additional problem is the integration of prelayouted hardware IP-cores from different vendors for specific process technologies. Those blocks are individually tested, but embedding them into the system structure is often not trivial.

Most digital systems designed today operate synchronously. Consequently, one of the crucial problems is the construction of the clock tree. The clock tree in complex digital systems is not just a set of buffers, but it includes clock-gating supporting circuitry, clock dividers for different clock domains, PLLs and complex clock-phasing blocks. Difficulties in clock-tree generation may lead to a substantial slow-down of the design process. Even worse, in some cases it is not possible to design a functional global clock tree at all. Additionally, a designer may have big problems with other timing closure issues like: the appropriate setup and hold time requirements, a reset–tree generation, and control of the boundary timing between different clock domains.

Today's industrial trend is focussed on cost reduction by System-on-Chip (SoC) implementation with integration of digital and analog processing parts on a single chip. There are already products on the market that can be denoted as SoC systems. However, the noise level in the RF part of the circuit can be significantly increased due to interference by strong spectral components of the synchronous clock frequency and their harmonics. The synchronous global clock generates increased electromagnetic interference (EMI), which can lead to severe distortions and crosstalk in the analog domain.

One of the most important properties of a communication system is mobility. However, mobile communication systems have one very critical constraint – power consumption. The limited capacity of the batteries creates firm limits for the system power consumption. Furthermore, the power demands of complex systems are usually high and hence, power consumption must be controlled and minimised. Partly, this can be achieved by using known methods for minimisation and localisation of switching power like clock gating, asynchronous design or voltage scaling. However, clock gating makes the design of the clock tree even harder. Furthermore, mobile systems should support an "idle" mode of operation that deactivates most of the system functionalities and even completely switches off hardware blocks from power supply.

An additional issue is "time to market". The intensive growth on the communication systems market has been followed with rapid technological developments and increased competition between companies. Consequently, design cycle time has been reduced to only few months. For example, the expected design cycle in 1997 was 18-12 months and in the year 2002 this was expected to be reduced to 8-6 months [CHAN99]. Accordingly, all already described design problems have to be resolved rapidly and more or less automatically.

Most known integration concepts are oriented towards general system structures and are not optimized for particular applications. The optimization towards a specific application can lead to increased performance, as well as power and noise reduction. This fact justifies our research effort in this area.

For example, digital signal processing algorithms in communication systems can be implemented with general DSP processors or dedicated datapath oriented hardware. If the system is implemented as a dedicated datapath architecture, it usually contains complex circuit blocks that perform sophisticated arithmetic or trigonometric operations. For example, a wireless LAN modem complying with standard IEEE802.11a [IEEE99] requires an FFT/IFFT processor, Viterbi decoder, CORDIC processor as well as cross- and auto-correlators. Often those blocks have point-to-point communication using localised or distributed control. Typically, the communication between those blocks requires high datarates. In many cases, periods of high data throughput are followed by periods of long inactivity, thus causing bursty activity patterns. An optimal integration technique for this particular system can give much better results than the application of some general integration technique.

The described issues have to be investigated when a complex communication system is being implemented. There are several methods and tools for managing each challenge separately. However, there are almost no techniques which conceptually address most of these issues at the same time. From my point of view, GALS techniques have the potential to solve some of the most challenging design issues of SoC integration of communication systems in the future.

1.2 GALS as a Solution for the System Integration Problem

The idea that system blocks can internally operate synchronously and communicate asynchronously is not novel. Already twenty years ago the first GALS proposal was formulated in [CHAP84]. However, this topic has been reconsidered many years later and it is currently in the focus of research. There are several GALS architectures that are proposed as an elegant solution for the problems described previously [YUN96, BOR97, MUT01, MOO00].

Usually, a GALS system consists of a number of locally synchronous modules each surrounded with an asynchronous wrapper. Communication between synchronous blocks is performed indirectly via asynchronous wrappers. The current level of GALS development offers a reliable framework for the implementation of complex digital systems. However, known GALS techniques may introduce drawbacks in performance, additional constraints in the system, and hardware overhead. On the other hand, all proposed GALS techniques are oriented toward some general architecture with sporadic and not too intensive data transfer on its interfaces. For datapath organised circuits, the known implementations of GALS do not fully utilise the potential of this technique.

There are several goals that we want to achieve with the introduction of GALS for communication systems. Firstly, the proposed GALS concept must guarantee fast and reliable transfer of large data-bursts between locally synchronous modules. The data-transfer must be possible at every clock cycle of the locally synchronous clock. The design-flow must be user-friendly and easily adoptable by a designer who is not familiar with asynchronous design. The construction of complex designs should be simpler than with the standard synchronous approach, and problems of timing closure and clock tree generation should be relaxed.

One of our main motivators is noise reduction inside the digital system. The global clock is the most important source of EMI. It is conceivable to conclude that if we decouple local blocks from the central clock source and connect them to different local clock generators, the spectral noise can be considerably reduced. Furthermore, we are aiming to reduce instantaneous supply current peaks. In order to deal with mixed-signal applications, this can be a powerful technique to decrease noise emission of the digital circuits. However, the noise reduction concept must not rely on the results of the GALSification only. Further improvements can be achieved with additional application of clock-jittering and clock-phasing.

Finally, mobile wireless communication systems' power consumption is limited by battery capacity. Therefore, a GALS design methodology should include power saving mechanisms. The goal is to completely integrate power saving mechanisms into the asynchronous wrappers. Consequently, GALSification would automatically introduce a certain power reduction. However, the power saving in GALS is based on the same assumptions as clock-gating in the synchronous design. The main idea is identical – lowering of switching activity by disabling the clock signal. Consequently, the results of power saving in GALS are expected to be close to the results achieved by clock-gating.

Based on the described motivation, in this thesis we will introduce a novel GALS concept and architecture, optimised for ASIC datapath structures, which are widely used in wireless communication systems. The complete spectrum of advantages and disadvantages of our proposed request-driven GALS technique can be seen best in a real complex datapath application. Our activity to work in the GALS area was initiated by the work on developing an IEEE 802.11a compliant modem. Therefore, a baseband processor for that standard appears to be the perfect candidate for introducing GALS. The baseband processor is a complex design (around 700k gates) with an internal datapath structure which includes sub-blocks such as Viterbi decoder, FFT/IFFT processor, and different CORDIC processors. Introducing the GALS technique in this baseband processor was a challenge and our results will be reported here.

1.3 Structure of the Thesis

The thesis is structured in ten different chapters. In Chapter 2, the state-of-the-art in the GALS area is described. Moreover, some alternative approaches for complex digital system integration are

described. Then, a discussion of possible gains of GALS techniques is presented. The conclusion of this chapter contains a critical view on existing approaches and identifies possible drawbacks when applying these.

Chapter 3 gives the motivation for our new GALS technique as well as the main ideas and theoretical concepts of a request driven GALS technique. Accordingly, a global structure of the request-driven GALS system is suggested and the purpose of the main system components is described. Furthermore, the potential gain of the application of the proposed technique is evaluated. Finally, some directions on the possible adaptation of the proposed concept are sketched. A modified version of the wrapper which avoids local ring oscillators and uses an external clock instead is introduced.

Furthermore, in Chapter 4, a possible request driven asynchronous wrapper implementation is proposed. The detailed structure of the asynchronous wrapper is shown. The main components of the wrapper are thoroughly discussed: pausable clock generator, clock control block, time-out generator, input port, output port, and mutual-exclusion element. Details on the formal analysis of the asynchronous wrapper are also included in this chapter. At the end of the chapter, a modified version of the asynchronous wrapper with external clocking is considered.

Chapter 5 is dedicated to the application of the proposed GALS technique in wireless communication system. The baseband processor for standard IEEE 802.11a is chosen as a verification platform for GALS. In this chapter, some details on the structure of the processor and a proposal for GALS partitioning is described. Special attention is paid to the power saving strategy in this system. Due to the complexity of the system, it was necessary to design several additional blocks as well as interfaces between synchronous and asynchronous blocks.

Modern digital systems must support effective ways of testing. Therefore, Chapter 6 is focused on testing and design for testability (DFT) of GALS systems. Different DFT techniques are evaluated. A test technique based on Built-in Self-Test (BIST) is proposed and the test structure is elaborated.

In Chapter 7 the current state of the design-flow, test flow and implementation details of asynchronous wrappers are given. Additionally, a complete GALS design framework is proposed. Our GALS technique is applied to the baseband processor, compliant with standard IEEE 802.11a. In this chapter all relevant details regarding this implementation are presented.

Chapter 8 reports the results of testing and measurement of the produced GALS baseband processor chip. Much attention is paid to answer the question how GALSification affected the power consumption. In addition to that, the power supply variation spectrum is measured and evaluated. All measurements include a comparison with the pure synchronous variant of the baseband processor.

Chapter 9 summarizes the achievements of this thesis and points to possible future challenges and solutions. Finally, Chapter 10 gives the references used in this thesis.

Chapter 2

Related Work – from Two-Flop Synchroniser to GALS Wrappers

2.1 Introduction

System on Chip (SoC) integration imposes a number of technical challenges on designers and tools. The number of difficulties for the SoC implementation additionally grows in the area of wireless communication systems. The methods and tools for reducing power consumption and minimizing crosstalk between analog and digital parts of the system are very limited and often inefficient.

Many of the challenges are in conjunction with the design of the clock network in complex digital systems. Clock skew appears to be a severe bottleneck for high-performance digital circuits. The synchronous transitions of the clock lines are a strong source of noise and electro-magnetic interference (EMI). Additionally, the power spent just for running the clock tree is comparable to the power consumed in the functional blocks of the system. It is conceivable to conclude that splitting a complex digital system into several independent subsystems, will relax problems significantly. Dealing with smaller blocks is much simpler, and power saving techniques could be more successfully applied. Crosstalk and EMI are suppressed due to the uncorrelated operation of the autonomous blocks. However, synchronisation between blocks operating at the different speed could be very complicated.

Several existing approaches address the problem of block partitioning and data synchronisation between independent blocks. Some of them are used to deal with increased power consumption and EMI. Today, these techniques are mainly referred to as Globally Asynchronous Locally Synchronous (GALS) methods. Many of them are effectively not applicable. However, some of the techniques are actually present in the design practice. Choosing between different proposed strategies depends very much on the particular system architecture.

In the following chapter, different synchronisation and integration schemes will be described. Hereupon, advantages and disadvantages of those techniques will be discussed. Additionally, some open points and issues not covered by the proposed techniques will be pointed out. Finally, directions for possible activities in the area of synchronisation and GALS will be suggested.

2.2 System Integration Strategies

There is a long history of approaches that guarantee safe communication between blocks that do not share the same common clock. In general, most of such systems rely on synchronous operation of the local blocks, and asynchronous communication is based on handshakes between them. All these systems could be referred to as Globally Asynchronous Locally Synchronous (GALS) systems. However, in the literature only one subset of the related approaches is actually referred as “GALS”. The “GALS” techniques are usually oriented toward providing a complete infrastructure for data transfer between the complex synchronous blocks and include pausable clocks as local triggers. On the other hand, there are many other approaches that aim to provide prerequisites for safe data transfer. In Figure 2.1, a classification of system integration concepts is given.

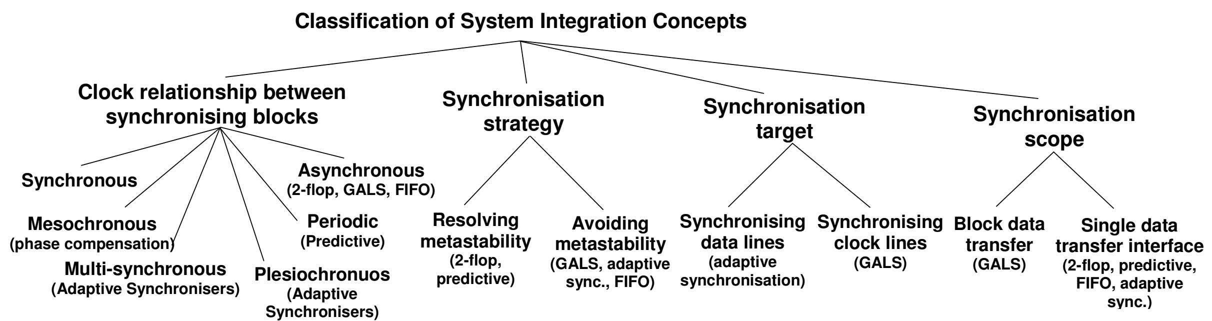

Figure 2.1. Classification of system integration methods

The first classification parameter is the relationship between the clocks of the synchronous modules that we want to integrate [GIN02]. In this context, a system can be synchronous (and no synchronisation is required); mesochronous when there is a fixed phase difference but same frequency of the different system blocks (in this case, a phase compensation can solve the problem); multi-synchronous or plesiochronous when the phase varies or there is a very small difference in frequency between the local blocks (adaptive synchronisation can be used for those cases); periodic (predictive synchronisers are used); or asynchronous (when classical 2-flop synchronisers, FIFO or GALS solutions are applied).

Furthermore, system integration techniques can avoid metastability (like in GALS or FIFO technique) or try to resolve from the metastable state (as in 2-flop or predictive synchronisers). An additional parameter can be the target for synchronisation. On one hand we can synchronise data

lines (as in adaptive synchronisation), and on the other hand, the clock lines can be synchronised (as in GALS).

Finally, the integration approaches can be categorized on the basis of synchronisation scope. Two main groups can be defined: the first that controls all existing data transfers from a single block to all other connecting blocks (the examples are GALS techniques), and the other that focuses only on particular interfaces for data transfer from block A to block B (this approach is used in all other "non-GALS" solutions).

In the following text I will first describe some of the interesting "non-GALS" proposals and then different "GALS" techniques and systems.

2.2.1 Standard Synchronisers

To avoid metastability and perform safe data transfer between asynchronously communicating blocks, it is possible to implement standard synchronisers based on a cascade of registers. These schemes are already known and successfully used for decades. A standard solution is to use two-flop synchronisers (Figure 2.2a) or, as a minimum, one-flop synchronisers (Figure 2.2b) [GIN03]. As can be seen from Figure 2.2, synchronised handshake is being performed to transfer data. In order to avoid metastability problems a number of flip-flops is inserted to synchronise the request (*Req*) and acknowledge (*Ack*) signals.

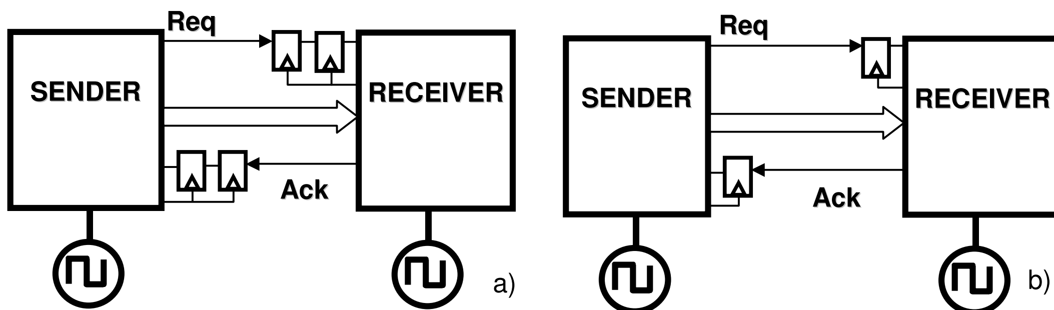

Figure 2.2. Two-flop (a) and one-flop (b) synchroniser

In general, the parameter that defines the synchroniser characteristics is Mean Time Between Failures (MTBF) [GIN03, DIKE99]:

$$MTBF = \frac{e^{T/\tau}}{T_W f_a f_D}$$

where τ is the settling time constant of the flop, T a settling window, T_W a time window of susceptibility, f_a the synchroniser 's clock frequency, and f_D the frequency of data transfer.

The technique that uses a two-flop synchroniser is very safe, and the probability of a failure is negligible. As an example, we calculated MTBF for 0.25 µm CMOS technology, with a clock frequency of 80 MHz and data transfer every 8th cycle. In this case, the MTBF can be estimated as 10^{258} years. Unfortunately, a two-flop synchroniser adds several clock cycles latency, which could be unacceptable for high-speed data communication. A one-flop synchroniser (Figure 2.2b) adds less latency in the communication channel, but decreases MTBF by reduction of the settling window *T*. To conclude, application of standard synchronisers is justified for low-speed data-transfer between independent hardware blocks. For high-speed purposes, some other scheme must be used.

2.2.2 Adaptive Synchronisation

As an alternative, the adaptive synchronisation techniques [KOL98] could be used for mesochronous systems. Mesochronous systems are using clock sources with exactly the same frequency but unknown, constant phase shift between different blocks. The general scheme of an adaptive synchroniser is shown in Figure 2.3. The main idea behind this approach is to delay data lines as much as it is needed in a particular moment, in order to avoid metastability.

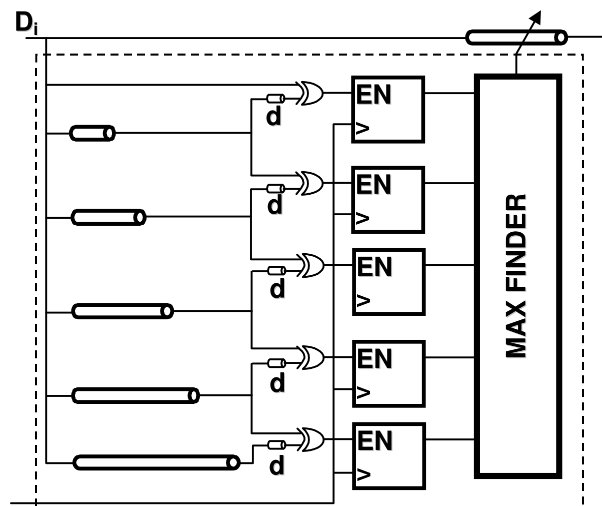

Figure 2.3. Adaptive synchroniser

Additional circuitry is needed to estimate the delay margin needed. For that purpose, a statistical phase detector is used, as shown in Figure 2.3. The role of this circuit is to perform measurements on a statistical basis and to find the delay within one clock period that corresponds to the lowest risk of metastability. The measurement is performed such that several counters count the number of ones at the outputs of the corresponding XOR gates. Every XOR gate should calculate the difference between the instantaneous data input and the same data input delayed by some fraction of the clock period. A configuration as shown in Figure 2.3 uses delay increments of T/5. After a long calibration time, some of the counters should show significantly higher values than the others. This will indicate that for the corresponding values of data delay, the risk of metastability will be very high. On the other hand, if we

choose a data delay that corresponds to the counter with the lowest stored value, this risk will be low. The statistical measurement is usually performed when the system is not in operation. These calibration periods are referred as *training sessions*. The duration of the training session is estimated to be around 100000 cycles in order to generate a representative statistical model of the data delay. Training sessions must be performed after reset and can be repeated periodically in order to keep track with PVT (Power, Voltage, Temperature) changes.

When the most suitable delay value for a data line is found, a tunable delay circuit connected to the corresponding data line is programmed. This way, the probability of metastability can be reduced to some degree, which is sufficient for most practical applications. However, this approach does not offer any power saving mechanism and introduces a relatively large hardware overhead, because a separate delay line is needed for every single data line. Also, the time overhead needed for statistical analysis of the data could be important. On the other hand, this solution could be accepted in the case of mesochronous communication between blocks. For the data transfer between blocks operating at different clock speed, this approach cannot be applied.

Researchers from Technion in Haifa, Israel have recently formulated an optimized solution [FRA04] for the same problem. This paper proposes the use of a *Two-way Adaptive Predictive Synchroniser* for mesochronous operation. The synchronisation latency is smaller than one clock cycle. However, even with providing a possibility for high-speed data-transfer between clock domains, this technique is restricted to mesochronous systems and adds a significant amount of hardware overhead.

2.2.3 FIFO Synchronisation

Another possible approach is interfacing blocks with specially designed asynchronous FIFO buffers. Consequently, hardware redundancy of the FIFO hides the problem of the synchronisation in the system. Such a system can tolerate very large interconnect delays and is also quite robust with regard to metastabillity. It can be used for interconnection of asynchronous and synchronous systems, but also for synchronous-synchronous and asynchronous-asynchronous connections. Acceptable data throughput via such an interface can be achieved [CHE01, CHE00a, CHE00b]. Additionally to the data cells, the FIFO structure includes the full and empty detector as well as a special deadlock detector. The advantage of FIFO synchronisers is that the operation of the locally synchronous module is not affected by synchronisation. 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 [CHAK99]. 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.

2.2.4 GALS Systems

Although all previous systems in general could be referred as GALS systems, usually this term is only used for systems that offer a complete design framework. GALS systems have a unique structure that is similar for all the different proposals. The principle architecture of GALS is given in Figure 2.4. Locally synchronous modules are usually surrounded by asynchronous wrappers. Local clocks drive those synchronous circuit blocks. Stoppable ring oscillators are frequently used to generate the local clocks. Data transfer between different blocks requires stopping of the local clocks during data-transfer in order to avoid metastability problems. The asynchronous wrappers should perform all necessary activities for safe data transfer between the blocks. Locally synchronous modules do not play any role in providing the prerequisites for block-to-block data transfer.

GALS as a technique was for the first time investigated in [CHAP84]. In this thesis, the fundamental basis of globally asynchronous locally synchronous systems was given. Although the circuitry described there cannot be successfully applied to modern high-speed digital systems, the ideas given there are even today very interesting. Use of stretched clocks, for example, is the basic idea behind this work, and is used in most modern GALS proposals.

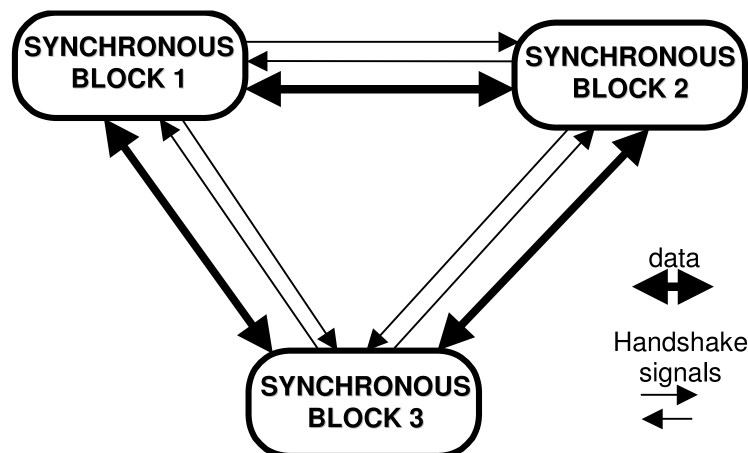

Figure 2.4. GALS architecture

Many years after the first proposal, the GALS idea has been reactivated and a working architecture is described in [YUN96, YUN99b]. This solution is based on pausable clocks in order to prevent metastability. To increase the throughput, an asynchronous FIFO is added to this circuitry as depicted in Figure 2.5.

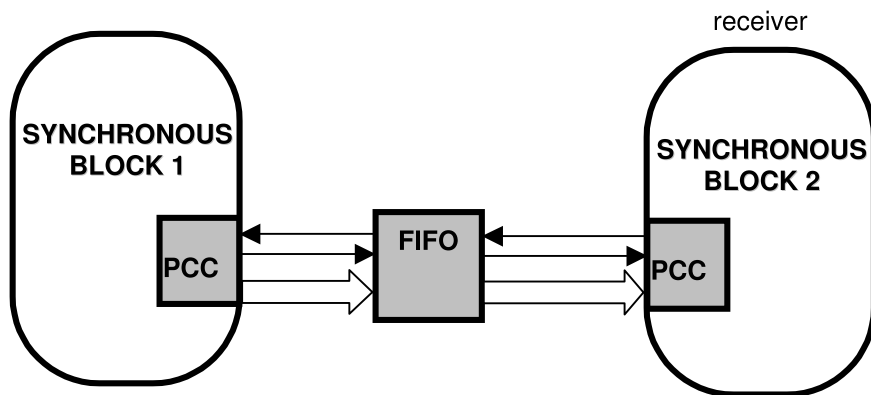

Figure 2.5. Two synchronous modules communicating via an asynchronous channel

The clock signal is controlled by a pausable clocking control (PCC) circuit given in Figure 2.6. A fundamental problem of the GALS technique is a prevention of the simultaneous appearance of the incoming request and the local clock. Consequently, for the purposes of arbitration a mutual exclusion element (ME) is used (Figure 2.6). Therefore, when the clock signal is high, an incoming request will not be processed until the clock is released. Also, when the request is asserted, the rising clock for the locally synchronous module will be delayed until the request is released. If both events appear at the same moment, the mutual exclusion circuit will "toss the coin" and grant just one of its outputs. Consequently, either the next clock cycle will be granted or a data transfer handshake. In this way, asynchronous data transfer between two blocks can be performed safely. The PCC controller is equipped with an AFSM for supporting the asynchronous handshake protocol, and additionally, with a synchronous FSM that should synchronise the acknowledge signal *Ap* to the local system clock *sysclk*. A major drawback is the limitation to transfer only one data item every other clock cycle. Additionally, multi-port applications will lower the maximum throughput. Therefore, this solution results in poor performance for data-transfer intensive multi-channel systems.

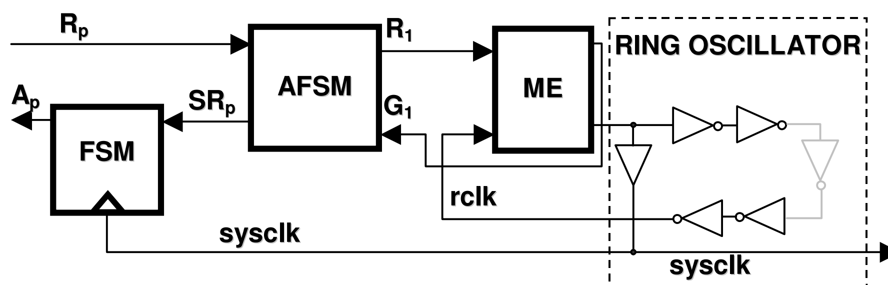

Figure 2.6. Pausable clock control (PCC) circuitry

Further improvements were made in [BOR97]. For the first time the term 'asynchronous wrapper' is used for an asynchronous interface surrounding locally synchronous modules as shown in Figure 2.7. 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. However, during data transfer, the receiver block clock will be stretched in order to avoid metastability. An asynchronous wrapper consists of input and output ports and local clock generation circuitry. Input and output ports can be implemented either as demand or poll ports. Both types stretch the clock during data transmission, but the demand port additionally stops the clock when data transfer is expected, in order to avoid unnecessary clocking. This concept has similar implications like clock gating in the standard synchronous design-flow. The locally synchronous part can be designed in standard fashion. This concept is further elaborated in [MUT01, MUT99], and better arbitration of concurrent requests is achieved. The work given in [MUT01] represents a comprehensive study of GALS systems, including one of the first practical GALS experiments - the implementation of a GALSified cryptosystem chip. Four port types are described there: input and output, as well as poll and demand type. 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 [VIL02, VIL03]. 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.

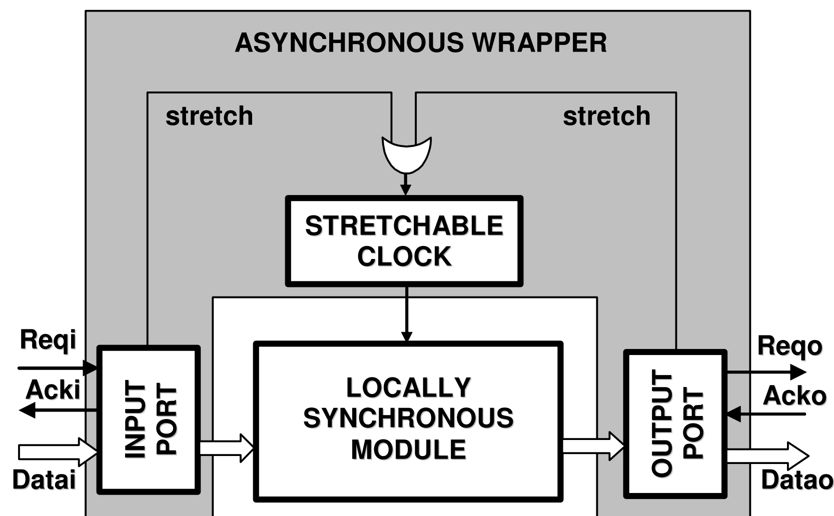

Figure 2.7. Asynchronous wrapper

The structure of the proposed system and specifications of the controllers are shown in Figure 2.8. The system configuration in that figure consists of one output D-type (demand type) port and one P-type (poll type) input port. In the datapath, one latch stage must be inserted to assure safe data transfer without risk of metastability. In the LS part, operation of the asynchronous ports must be supported with the insertion of flip-flops. They shall grant data transfer in every clock cycle. In order to allow high-speed data transfer, the toggle flip-flops (T-FF) are used. Therefore, a single control signal (local clock) can drive data enable flip-flop. In Figure 2.8b and 2.8c, the port specifications of the

Demand-output and Poll-input port are given, respectively. The port structure consists of two identical patterns in order to react on both edges of the data enable signal *Den*. 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. Therefore such controllers offer high throughput.

Recently, the solution from [MUT01] was extended with a test methodology that includes circuitry able to perform a functional test of the asynchronous wrapper [GUR02]. Additionally, an improved adjustable clock generator is presented with the possibility to tune the frequency over a broad range. However, frequency resolution decreases for the higher part of the spectrum.

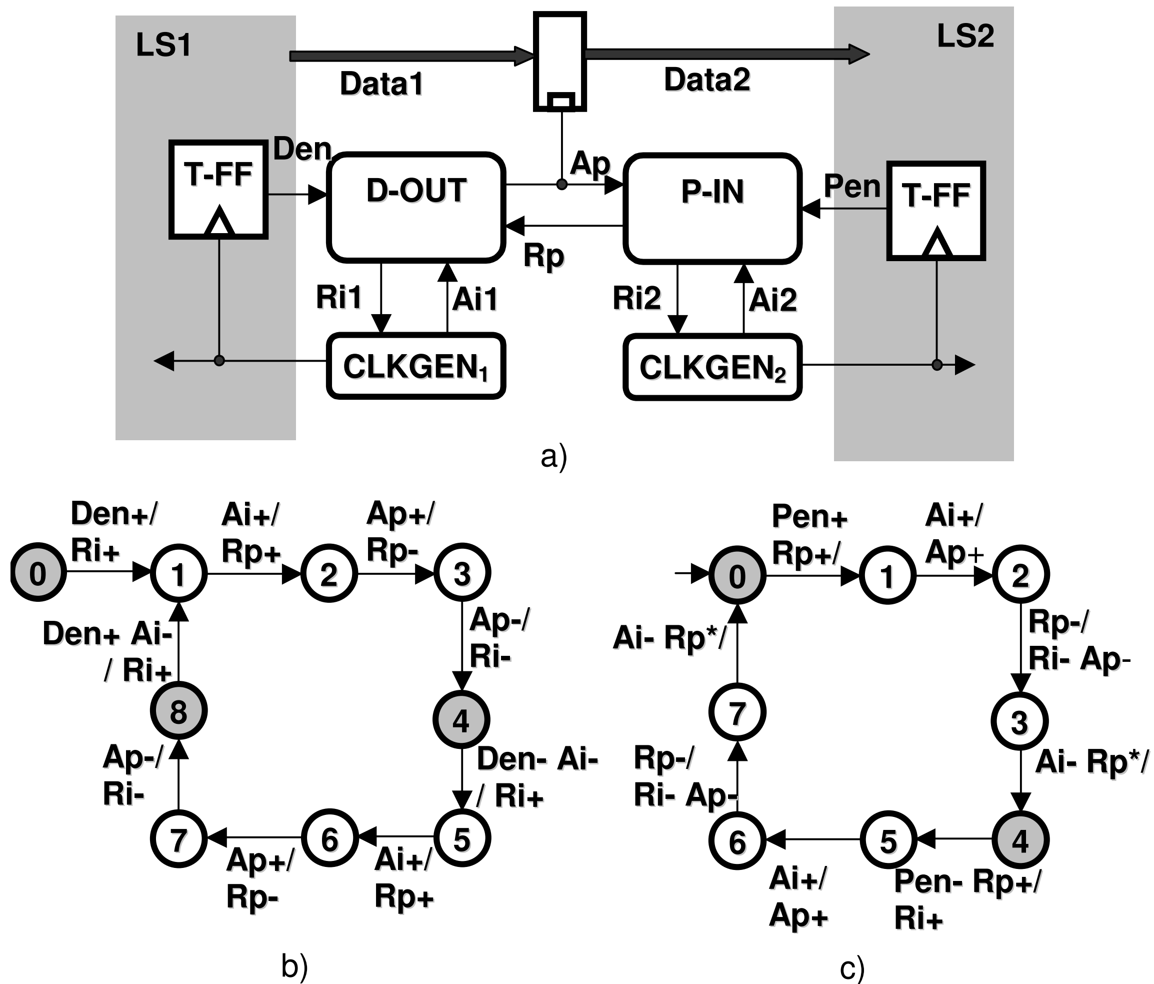

Figure 2.8. GALS architecture for high-speed data communications (a), Demand-output (b) and Poll-input (c) port specifications

Some proposals are made for the calibration of the local clock generator. However, the offered solutions are not automated for industrial application. In [MOO00] 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.

In many papers the proposed solution in [MUT01] is considered as a reference solution for GALS systems. However, some drawbacks are noticeable even with this solution. In [DOB04] it is shown that direct application of that solution may lead to a metastability problem, because the wrapper does not take into account the effect of the clock-tree insertion in the LS module. In this paper some solutions for fixing this problem are presented. However, even this "upgraded" solution suffers from different timing limitations. For example, in the worst case, data needs to propagate within only half a clock cycle, from one LS module to the first register of the subsequent LS module. This implies additional registering of data at the boundary of locally asynchronous blocks. Additionally, the formal analysis in [KON01] showed that several hazards, race conditions, and other faults are present in the proposed wrappers from [MUT01].

A solution for point-to-point interconnection that uses very small and simple asynchronous wrappers is suggested in [MOO02]. The principle used in this work is very similar to the one used in the other GALS approaches, but the used circuitry is further optimized and minimized. In this proposal, as shown in Figure 2.9, in order to achieve maximum throughput, an asynchronous FIFO between producer and consumer is added. In general, a synchronisation circuit contains just a couple of flip-flops for synchronisation, an arbiter, and a stretchable clock generator.

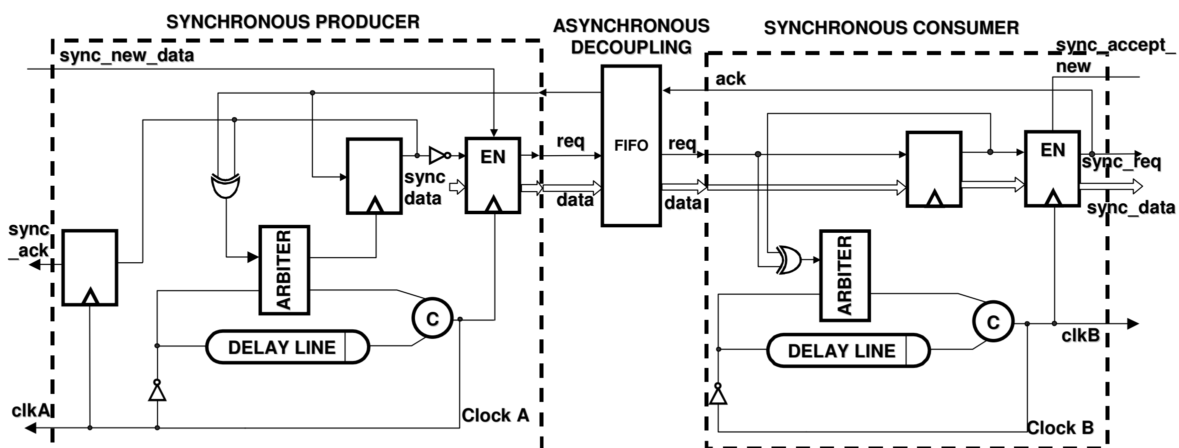

Figure 2.9. Point-to-point GALS interconnect

Unfortunately, the proposed architecture does not offer high-speed data communication between clock domains. For mesochronous systems, the data could be transferred at most every second clock cycle even with FIFO buffering. This limitation restricts the use of the proposed circuitry for any data-transfer intensive application.

Another type of a point-to-point communication GALS architecture is proposed in [ZHU02a, ZHU02b, ZHU02c, CAR03]. The structure of this system is shown in Figure 2.10a. Signal transitions of the write and read port are given in Figure 2.10b. This is probably the simplest GALS controller presented in the literature. It contains just a stretchable clock generator and W-port (Write) and R-port

(Read) asynchronous controllers. The signal transition graph of the ports, as it is shown in Figure 2.10.b, is quite simple. Therefore, the hardware complexity of the controllers is very low.

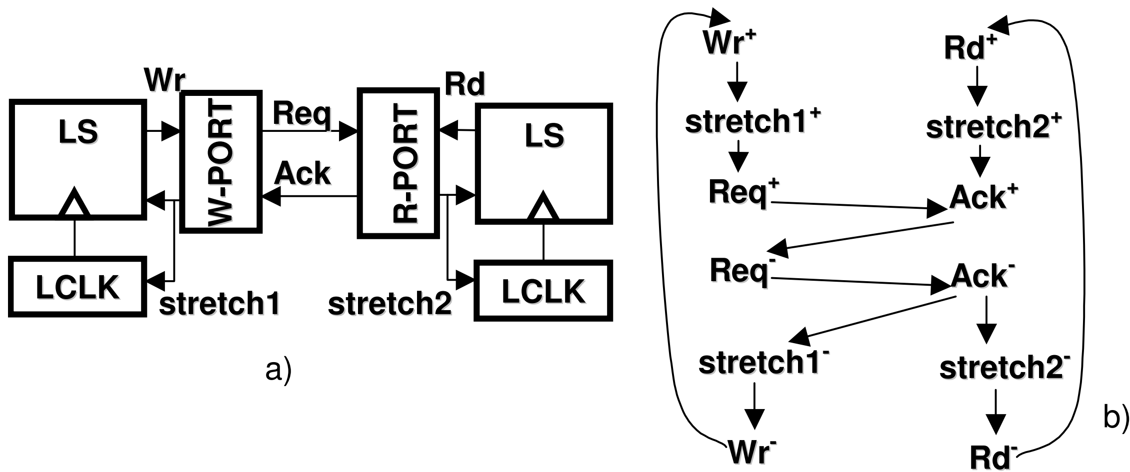

Figure 2.10. Structure of point-to-point system (a) and signal transitions of the ports (b)

However, the applicability of this architecture is limited to specific cases, because there is a strong coupling between the operation of producer and consumer. For example, the producer clock is stretched until the consumer decides that data transfer should be performed. Therefore, this solution could be used only if the data transfer is needed every cycle, and producer and consumer are operating at the same speed. In other cases, the performance of the system will be dramatically decreased. Alternatively, an additional buffer between the ports may increase the performance. On the other hand this solution increases hardware complexity of the GALS interface.

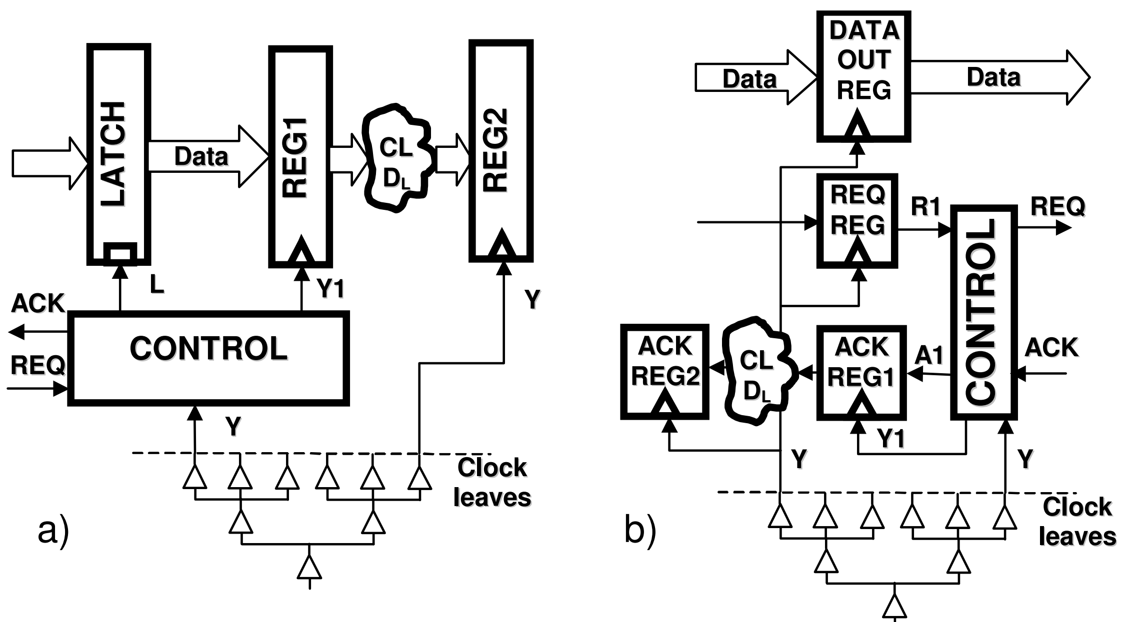

Figure 2.11. Structure of locally delayed latching input (a) and output (b) port

In the scientific community, there is also an opinion that the concept of pausable clocking cannot promise the best results for high-speed applications as stated in [DOB04]. In this paper, several ways of synchronising the data between independent clock domains on the basis of GALS are proposed by the scientists from Technion, Israel. Additionally, in contrary to the other main proposals, the architectures presented in this paper take into account the presence of a clock tree. One interesting solution is shown in Figure 2.11. In Figure 2.11.a, the concept of the locally delayed latching input port is presented and in Figure 2.11.b the structure of the equivalent output port is given. This proposal introduces an interesting and novel concept of locally delayed latching. It allows application of the standard clock generators (and avoids use of ring oscillators) and performs data synchronisation only at the LS module boundaries. Thus, as seen in Figure 2.11.a, in the input port, synchronisation is limited to the stages *Latch* and *Reg1*. Block *Control* performs the asynchronous handshake communication with the blocks and the generation of control signals for the register stages. Similarly, in the output port, a couple of registers are used for the synchronisation with the local clock, and a *Control* unit performs the asynchronous communication.

However, as can be seen from the construction of the output port, the data cannot be transferred every clock cycle due to the synchronous handshake that is performed at the boundaries of the synchronous part of the output port. Actually, one data transfer usually needs several clock cycles. This property of the proposed architecture restricts the application of the proposed circuitry to low speed data-transfer applications.

2.3 Power Saving with GALS

The evaluation of the power saving benefits of GALS systems was performed by several authors. Most of these investigations are based on the application of GALS to high-speed processor implementations. However, these results show some general trend. In Figure 2.12 the power distribution in a high-performance CPU is given [MEI99, HEM99].

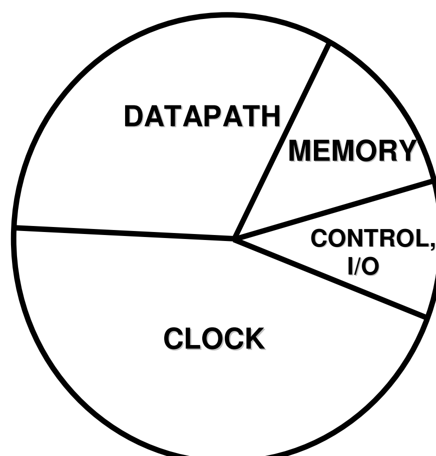

Figure 2.12. Power distribution in high-performance CPU

It can be seen from the figure that the clock signal is the dominant source of power consumption in such an environment. It is reasonable to assume that within GALS, the clock network is split into several smaller sub-networks with lower power consumption. First estimations, according to [MEI99, HEM99], showed that about 30% of power savings could be expected in the clock net due to the application of GALS techniques. Recently, some more pessimistic power estimation figures were presented in [IYE02]. After modelling the GALS superscalar processor, they demonstrated that GALSification of the system actually leads to a performance drop. The achieved power reduction is not very impressive either. The expected drop in performance was estimated around 10% and power saving was about 10%. Finally, an increase of around 1% in energy consumption was observed.

However, GALS techniques offer independent setting of frequency and voltage levels for each locally synchronous module. When using dynamic voltage scaling (DVS), an average energy reduction of up to 30% can be reached, yet associated with a performance drop of 10%, as reported in [TAL05].

In general, we can conclude that the power saving techniques that are immanent with the GALSification of a system have similar limits as clock gating. Further energy reduction is possible only with the application of DVS in conjunction with GALS.

2.4 Open Questions and Directions for Further Research

In the previous subsections, several integration techniques were presented. However, there is still scope for further research. Some of the design challenges are not fully addressed in known GALS systems. Additionally, the proposed GALS techniques are not optimal for many specific hardware architectures.

In many of the proposed solutions, the problems of data transfer and throughput is critical. Most of them can perform data transfer every second clock cycle of the local clock. Even this result can be reached only in an ideal case. In reality, this relation can be even worse. In some proposals, the described circuits can theoretically transfer data every clock cycle, as in [MUT01]. However, the intensive stretching of the pausable clock generator will significantly diminish the practical performance. The performance degradation reported in [MUT01] was around 23%. Stretching the clock every cycle would lead to a situation where the effective clock frequency is determined not by the clock generator but by the rate of communication with other synchronous modules as stated in [IYE02]. Consequently, a novel proposal, optimized for data transfer intensive application is urgently needed.

The clock generation issue is also critical. Usually, standard ring oscillators constructed from standard cells are deployed. They cannot generate a stable clock signal at a defined frequency. That creates difficulties for many GALS systems. The frequency fluctuations complicate data transfer with synchronous data producers or consumers such as analog-to-digital (ADC) and digital-to-analog converters (DAC) operating at a predefined and stabilised clock. In communication systems, it is

nearly always necessary to communicate with such blocks. A way to resolve this problem is to over-constrain locally synchronous modules to work at a higher local frequency. Another solution could be a mechanism to calibrate the local clock. However, this would further complicate the design and cause additional cost. Additionally, in many solutions, the effect of clock tree insertion is not taken into account. Hence there is no real guaranty that metastability is avoided.

In more or less all GALS solutions, the operation of neighbouring asynchronous wrappers is strongly decoupled. The latency of the transferred data is not known in advance and may vary from one data transfer to the other one. Such property inherently leads to increased non-determinism in GALS systems. To conclude, a more stable GALS framework is needed, which is able to guarantee fixed and minimal latency in the data transfer and, consequently, reduces non-determinism in GALS circuits.

All solutions described so far are oriented towards a very general application. Consequently, they are not optimised for specific systems and environmental demands. Although these techniques are, to a certain extent, intended for use by synchronous designers without special knowledge in the area of asynchronous circuit design, some mechanisms cannot be implemented easily. In general, a standard synchronous design strategy must be adapted to support the design of asynchronous wrappers. Additional functions have to be incorporated to facilitate interaction with the asynchronous wrapper. In some cases it is necessary to insert complex interface blocks (for example an additional FIFO) between asynchronous blocks to prevent performance deterioration due to local clock stretching [MOO02].

In many GALS proposals the problem of power saving is not addressed. Additionally, if some power saving mechanism is provided, a deeper practical investigation is needed to estimate the effect of GALS introduction with respect to power consumption.

I believe that, for datapath architectures, the technique presented in this thesis results in better performance than known implementations.

Chapter 3

Proposed Novel GALS Architecture

3.1 Introduction

In the previous chapter a number of different system integration techniques were presented. Most of them are defined just at the conceptual level, but there are some that have been implemented in hardware. Our novel GALS architecture was developed for datapath architectures. It simplifies system integration and solves some of the design problems.

In this chapter, the basic concept of our novel request-driven GALS approach will be presented. First, I will discuss the motivation to work in this area and the general concept of the new technique. Then, some global structure of a possible implementation will be presented. Additionally, the potential gain of this approach will be estimated. Finally, a variant of the introduced GALS concept is proposed that avoids local clock generation.

3.2 Motivation and General Principles

Different GALS systems were reported in the literature for several years. As the result of intensive research, many GALS proposals were defined. On the basis of existing techniques we decided to define a more optimal solution for datapath architectures. Three important aspects motivated the work presented here. Firstly, it was our goal to establish a general and user-friendly design framework for reliable integration of large digital systems with one or more clock domains. Secondly, much of our effort is dedicated to EMI and crosstalk reduction in order to ease the integration of mixed-signal designs. Thirdly, it is our aim to avoid unnecessary transitions and the attendant waste of energy during data transfer between GALS blocks. In this context, it is necessary to achieve high data throughput with low latency.

It appears conceivable that for a continuous input data stream the GALS blocks may operate in quasi-synchronous mode. In this way, no redundant transitions are generated. Therefore, in the mode of operation in which data is continuously received at the input port, the request-driven clock (Figure 3.1) is directly derived from the request signal at the input port. However, for bursty signals, when there is no input activity, the data stored inside the locally synchronous pipeline has to be flushed out. This can be achieved by switching to a mode of operation in which the local clock generator drives the GALS block autonomously. To control the transition from request driven operation to locally driven mode, a time-out function is proposed. The time-out function is triggered when no request has appeared at the input of a GALS block, but data is still stored in internal pipeline stages. It is also conceivable to switch from a request-driven (push) operation to an acknowledge-driven (pull) operation in this situation. However, this idea was rejected since it would be difficult to deal with isolated tokens inside the synchronous pipeline. In particular, if the pull operation does not result in a token being propagated to the primary output, an internal oscillator would still be required. A much more sophisticated control would be required to facilitate this approach, which is in principle feasible.

Our proposed concept, which we have described in [KRS03a, KRS03c], is intended for complex systems mainly consisting of point-to-point connections between circuit blocks. We assume, that the communication between blocks is very intensive (i.e. every clock cycle of the local clock) but bursty, with longer periods of inactivity. For example, in the case of a baseband processor for WLAN [IEEE99], all OFDM symbols are composed of a 0.8 μs guard interval requiring very little computation and the actual 3.2 μs data field, which is processed, using sophisticated algorithms.

It is assumed that a distributed control mechanism according to the ‘token flow’ approach is deployed for synchronous blocks [BUCK93]. Therefore, a separate controller drives each of the blocks. Such controller also sends a signal (token) to the next controller indicating its status. Depending on this token signal, the next controller drives the block it is responsible for. In addition to this, the modules belonging to the same block generate another (token) signal for the subsequent module on completion of operation. Depending on this signal, the module accepts the data from the previous one and processes it. This allows controlling of the complete system in an elegant and effective way. This technique is applied to the datapath system, which consists of autonomous blocks. Usually one block processes input data-bursts coming from the previous block and, as a result, creates an output data burst for the following block. Accordingly, when the input token indicates the validity of data, this should be a signal for the receiving block to start operation in request-driven mode.

3.3 System Structure

The request-driven GALS technique is based on the standard asynchronous wrapper (AW), as other main GALS techniques. Conceptually, the locally synchronous module is driven both by the incoming request as well as the local clock signal. The driver of the request input signal is the output of

the asynchronous wrapper of the predecessor block. It is aligned with the transferred data, and can be considered as a token carrier.

When there is no incoming request signal for a certain period of time (defined as $T_{time\text{-}out}$), the circuit enters a new state where it can internally generate clock cycles using a local ring oscillator. The number of internally generated clock cycles is set to match the depth of the locally synchronous pipeline. When there is no valid token in the synchronous block, the local clock will stall and the circuit remains inactive until the next request transition, indicating that a fresh data token has appeared at the input. In this way, we avoid possible energy wastage.

This proposed solution allows a completely autonomous operation of the asynchronous wrapper from the locally synchronous module with respect to clock control. However, this is just one possible way of controlling the local clock generator. If this control mechanism is not applicable for a particular system, it is possible to directly control the stopping of the clock via some synchronously generated signal that indicates an empty local pipeline.

The scenario after time-out and start of the local-clock generation is more complex and demanding. When a new request appears before the synchronous pipeline is emptied, it is necessary to complete the present local clock cycle to prevent metastability at data inputs. Moreover, one important issue is to assure that the clock period is within specified limits. However, it is possible to safely hand over clock generation from the local ring oscillator to the input request line. To deal with this situation it is necessary to implement some additional circuitry to prevent metastability and hazards in the system.

The principle architecture of our modified asynchronous wrapper around a locally synchronous module is shown in Figure 3.1. In order to better understand the operation of the asynchronous wrapper, its structure is partitioned into several sub-blocks with different functionality. The input and output ports are controlling data transfer to/from the GALS block, a time-out generator observes inactivity of the input request line, and the local clock generator triggers the locally synchronous module when this is needed.

With the proposed solution we can guarantee the frequency and the shape of the clock pulse distributed to the locally synchronous module. On the one hand, the clock may be derived from the local clock generator, which is tunable. Consequently, we can control the rate and width of the clock pulses. On the other hand, the source of the clock could be the request signal that is generated from the previous asynchronous wrapper in the dataflow chain. This request signal is eventually controlled by the corresponding clock signal of the data source. In this way, the specified time limits for the period of the request signal are guaranteed. In addition to that, the control of the clock signal transition time is provided by the clock-tree insertion in locally synchronous modules. The activity of the request-driven clock and the locally generated clock is mutually exclusive and hence hazards are avoided.

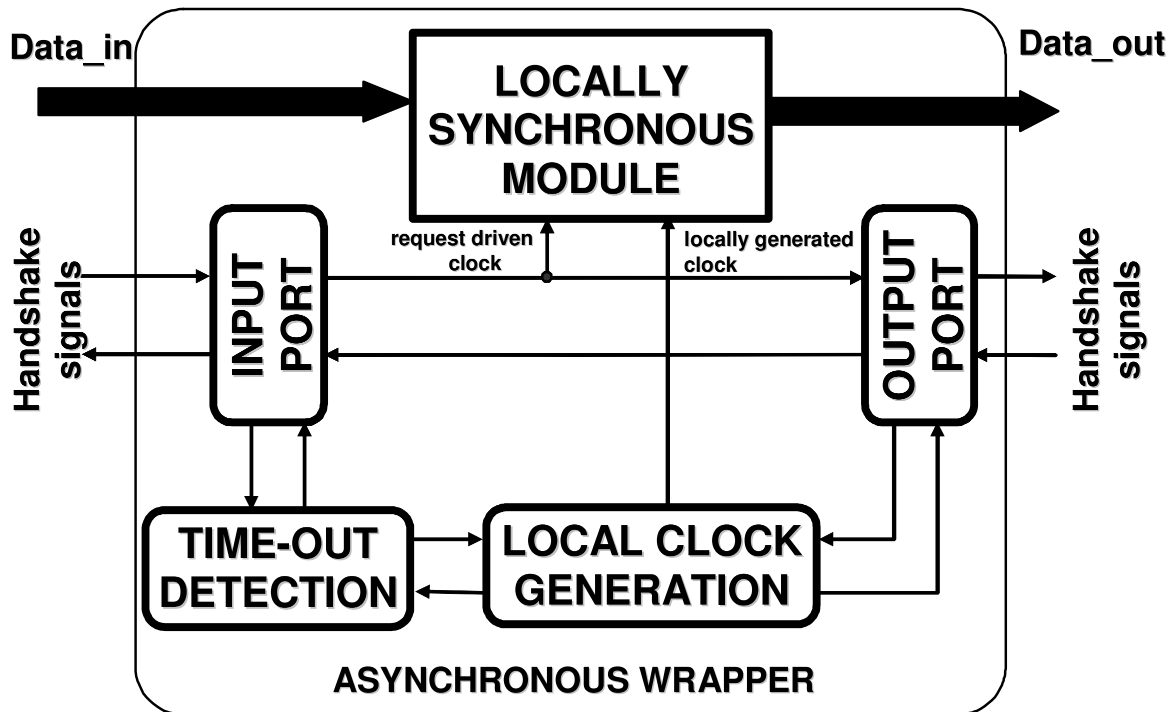

Figure 3.1. New asynchronous wrapper around locally synchronous module

Communication between different AWs is based on a standard 4-phase handshake protocol. Therefore, although the concept is invented for point-to-point interfaces between blocks, a more complex dataflow system can be constructed. Standard asynchronous elements such as 'joins' and 'forks' can be employed, as shown in Figure 3.2. For example, a join of two token streams generated from two AWs can drive the receiving AW. However, only a single data stream can drive a GALS block due to the request-driven property of the wrapper. Consequently, the different data streams approaching from different data sources must be joined before they enter a particular GALS block. In the current implementation, the output port generates single data stream. Therefore, if the output data stream has to be distributed to several sinks, a fork construct must be implemented. On the other hand, it is possible to adapt the proposed wrapper architecture in order to support multi-output-port structures.

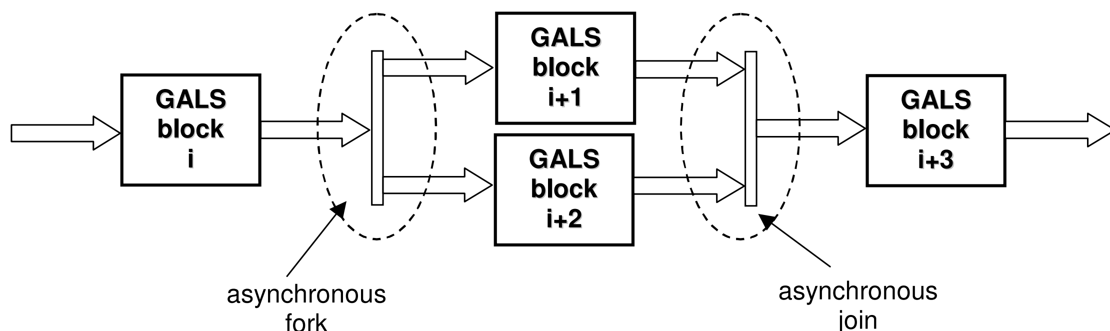

Figure 3.2. Implementation of some more complex dataflow scenarios

3.4 Request Driven Technique with External Clocking

Alternatively, the same concept could be implemented without the application of pausable clocks. Pausable clocks are usually implemented as ring oscillators. When ring oscillators are used, additional power is spent for switching of the numerous invertors inside the oscillator. The amount of this power could be considerable. Additionally, in order to apply ring oscillators in practice, a careful calibration of the oscillator frequency for each produced chip is needed. This fact may be a disadvantage for wide industrial application of our request-driven GALS technique.

However, an adaptation of the proposed GALS concept is possible in order to accommodate the application of standard external clocks, as we proposed in [GRA05]. In general, a GALS wrapper should continue to operate in request driven mode in the same way as described in the previous subsections. For the time-out measurement, instead of the local clock, an external clock source can be used. When the time-out is reached, the locally synchronous module should start token generation until the internal pipeline is emptied. For this triggering we may use an external clock source instead of the locally generated clock. When the pipeline is empty, the external clock must be disabled by clock-gating. In this case, the block denoted as Local clock generation in Figure 3.1 should be substituted with the Clock Management Unit (CMU). The modified block diagram is shown in Figure 3.3. The purpose of this block is the safe gating of the external clock source when it is not needed.

The advantage with such an implementation of the request-driven GALS technique is that usage of the ring-oscillators is avoided. Power consumption is reduced due to abolishing the oscillator. The designer is able to work with the standard external clock source. The determinism in the system is increased and testing of the designs is easier.

However, there are some drawbacks as well. The performance of the system is in general decreased. This is due to the fact that we cannot anymore pause the clock for a moment as with local clock generation. A particular external clock cycle can be either accepted, or completely discarded. Consequently, the latency in the system is increased. Furthermore, the CMU block must be very carefully designed in order to avoid problems of unsafe clock-gating, such as clock clipping and possible glitches. Finally, using an external clock source decreases the autonomy of the local block and also limits the possibilities for optimal clock profiling for the particular block. With the tunable local generators it is very easy to set the clock frequency to the minimal value needed to perform the desired function. When the external clock is used, it is very expensive to use a separate external clock source for every local block. In that case it is better if several blocks share the same clock source.

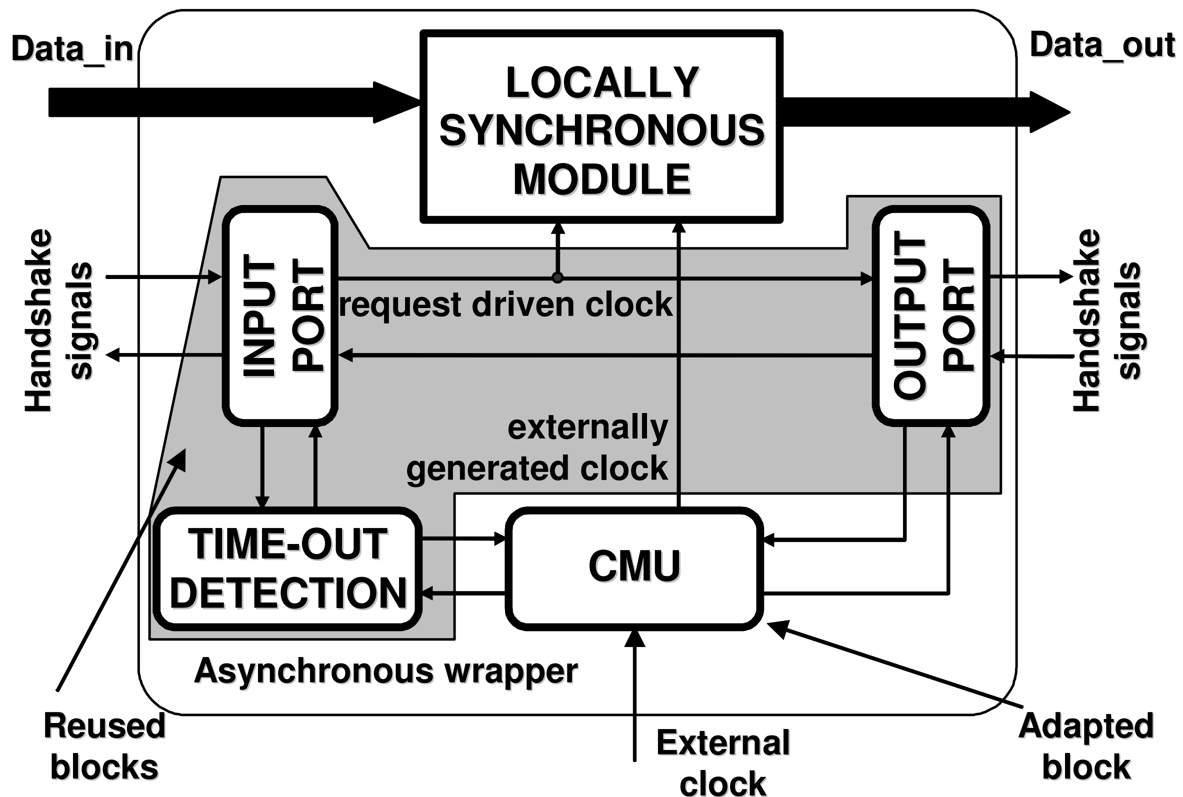

Figure 3.3. Externally driven asynchronous wrapper

In general, it is up to the designer to decide which version of the request-driven GALS technique should be used. There are pros and cons for both techniques, and depending on the particular application, the best one can be selected. In the following text, more attention will be paid to the technique that involves local clock generation, and the practical application is based on this technique. However, basic implementational ideas will be presented also for the externally clocked GALS wrapper.

3.5 Potential Gain of the Novel GALS Architecture

The proposed request-driven architecture has several potential advantages. It can be an excellent basis of the design framework for integration of complex digital datapath systems. For example, reliable and fast transfer of large bursts of data is achieved. Data transfer is possible at every clock cycle of a locally synchronous module. This feature, in general, is missing in most of the other published proposals. The clock speed is determined by the clock speed of the communication master and not by the slower participant in the data transfer. Additionally, in request driven mode, when data receiver and sender operate quasi-synchronously, it is possible to transfer the data without any performance loss, because no additional synchronisation of the locally synchronous systems is needed. This is a big differentiator compared to all other GALS techniques where every cycle of the data transfer is connected with another series of synchronisation and arbitration operations. Moreover,

the data transfer latency is decreased, when the circuit is operating in request-driven mode. In this mode, arbitration is not needed and incoming requests are automatically considered as a clock signal. Therefore, our implementation does not suffer the performance drop incurred by other GALS solutions. The proposed GALS approach follows the usual design style of synchronous blocks and does not require significant changes of locally synchronous components.

Hence, we can expect some system improvements as a consequence of GALS introduction. GALSification of a complex system results in independent clocks driving locally synchronous modules. These clocks will have arbitrary phases due to varying delays in different asynchronous wrappers. Furthermore, the clock generators of the local blocks will run at different frequencies. In this way the generation of substrate noise, supply noise, and crosstalk can be reduced.

In order to estimate the effect of GALSification on EMI characteristic, we have created a MATLAB model of the supply current variation of an externally and an internally driven GALS system and compared them with an equivalent synchronous system. The model for the current shape is based on the same assumptions as in [BAD04, BLU04]. First, we have modelled the current shape for a synchronous system as a triangular waveform with a 5 ns rise time and 10 ns fall time [GRA05]. The peak supply current was assumed to be 1 A and the clock period was assumed to be 20 ns (50 MHz). Additionally, we have assumed that after GALSification 10 blocks are formed. GALSification leads to a different timing profile of the current consumption. As a consequence, we assumed that as a result the following shapes are created: 2 blocks of 200 mA each, 4 blocks of 100 mA each and 4 blocks of 50 mA each. Rise time and fall time were not changed. We performed separate simulations for the externally and for the internally driven GALS system. In order to model the externally driven GALS system, each of the 10 current waveforms was given an equally distributed random phase in the range of ±10 ns. For the locally driven GALS system we additionally defined that those 10 blocks have a difference in frequency of ±2 % due to the tuning resolution of the ring oscillators. The ten waveforms, each representing the supply current of a GALS block were combined to estimate the total supply current. Subsequently, the power spectrum of the total supply current was calculated using a Fourier transform. The results of this Fourier transform given in Figure 3.4 show, that the maximum spectral peak with GALS is reduced by around 20 dB. For a wide range of frequencies the spectral components in the GALS system are reduced by at least 10 dB when compared to the synchronous circuit. Furthermore, in the time domain the supply current peaks are reduced to about 40% of the synchronous system. Both, for the frequency domain and time domain, externally and internally driven GALS show similar results.

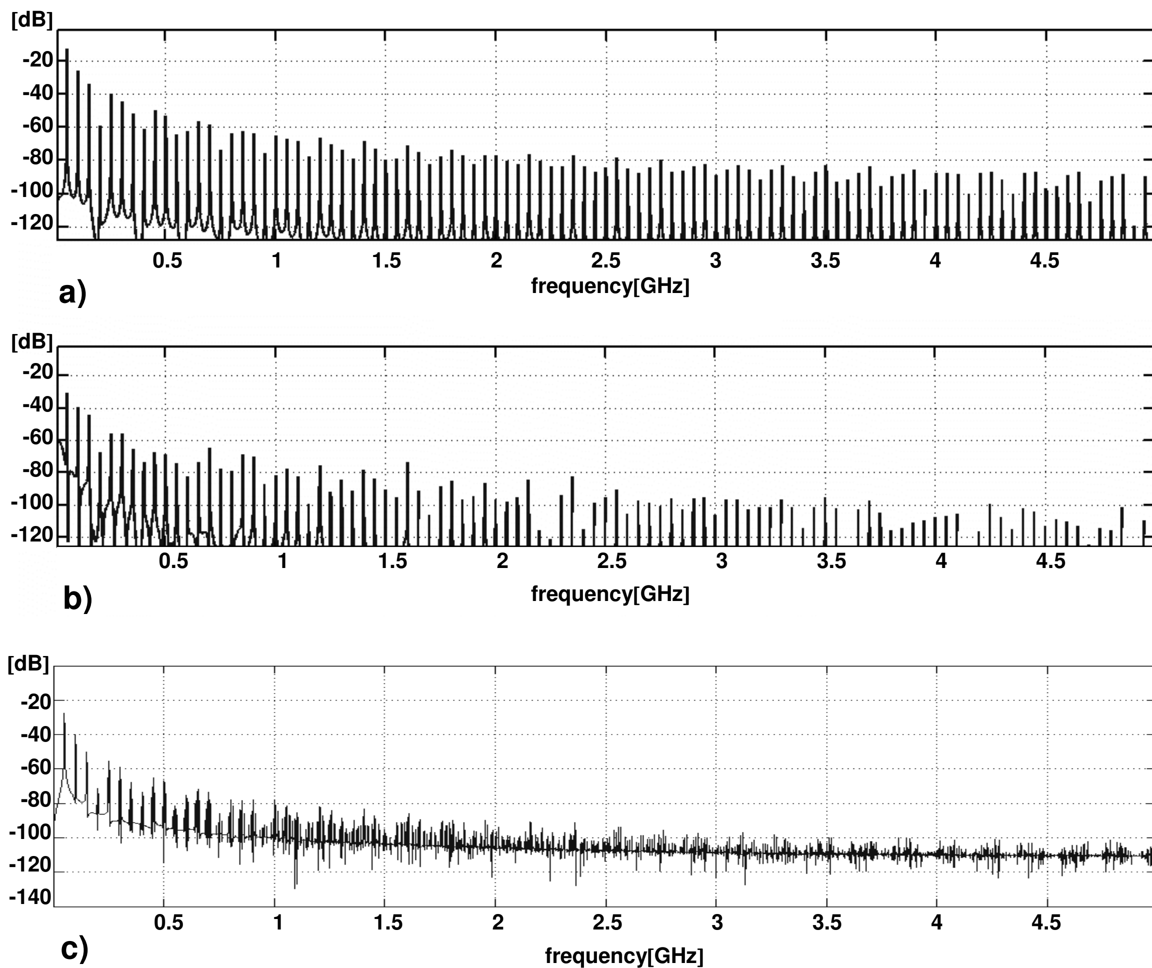

Figure 3.4. Current spectrum of synchronous design (a), externally driven GALS (b), and internally driven GALS (c)

Our proposed architecture offers an efficient power-saving mechanism, similar to clock gating. A particular synchronous block is clocked only when there is new data at the input, or there is a need to flush data from internal pipeline stages. In all other cases there is no activity within this block. Moreover, the depth of the clock tree is reduced, due to the decoupling of the global clock tree into a set of independent local clock trees. In this way, the power consumption can be further reduced. GALSification also more efficient facilitates application of other low-power techniques. For example, in pure synchronous systems, blocks are often clocked at higher frequencies than actually needed. This is done to avoid problems of synchronisation with neighbouring blocks and to limit the number of clock domains. Using our GALS technique, we can optimally tune the clock frequency of any particular block and the asynchronous wrapper will take care of the communication. Additionally, for each GALS block, the optimal supply voltage could be applied, which would enable further reductions in power dissipation. Furthermore, fine-grained clock gating can be applied within locally synchronous modules. However, the general power saving limit of any GALS technique is very much comparable to clock

gating and it is not realistic to expect significant savings in power consumption with GALS application in comparison with a synchronous system with clock-gating.

The target applications of the proposed GALS technique are datapath architectures, often deployed for baseband processing in communication systems. From a top-level view, our architecture is equivalent to an asynchronous design with coarse stages. In contrast to 'normal' asynchronous circuits, a stage is not restricted to a single register with combinational logic, but can be a complex and deeply pipelined synchronous block.

As with all GALS systems, no global clock-tree is required. The clock signal is generated by 'multiplexing' the local clock and the input requests. Due to the request-driven operation, the local clock usually does not need to precisely match the frequency of the global clock or the datarate. Hence, there are little constraints for the design of the ring oscillators. Locally synchronous modules do not have to be 'overconstrained' and can be designed for clock frequency f_c as with standard synchronous designs. For the proposed implementation there is no need for registering input data in locally synchronous modules. This prevents unnecessary latency and hardware in the system.

A token-flow approach, often deployed in a synchronous environment anyway, seems to be a more natural style to design synchronous blocks for GALS application than the design rules proposed for the GALS architecture in [MUT01]. An additional advantage is that our locally synchronous pipelines can be easily substituted with fully asynchronous circuits if needed.

Standard GALS implementations as advocated in [MUT01] do not well support communication with ADC and DAC running at a fixed and stable clock f_c. The local clock signals within those GALS blocks would have to be tuned to a higher frequency than f_c or large interface FIFOs must be used. With bursty data transfer it can be shown that stretching the local clock at every single data transmission will significantly diminish performance. We think that our approach is better suited for the particular systems described above.

However, there are some issues that limit the applicability of the proposed technique. For example, there is a performance loss due to the time-out measurement. The introduction of 'end-tokens' to trigger emptying of internal pipeline stages, as we proposed in [KRS03a], can improve this situation. In that paper, we suggest introduction of an additional token indicating that the transmitting GALS block has no more tokens to transfer. Consequently, time-out measurement is not needed anymore. However, this solution leads to non-standard changes of the locally synchronous module. The applicability of our wrapper is limited to architectures with data-streams based on token bursts. Consequently, processing of single tokens can diminish performances. Finally, the proposed implementation of the GALS concept is more complex than some other GALS techniques.

Chapter 4

Hardware Architecture of the GALS Wrapper

4.1 Introduction

The request-driven concept as described in the previous chapter can be implemented in many different ways. This chapter derives one possible implementation of an asynchronous wrapper compatible with the proposed concept. First a detailed structure of the wrapper is described. Then the following GALS system blocks are discussed: pausable clock generator, time-out block, input and output ports. In order to verify the correctness of the planned implementation, a formal analysis of the proposed asynchronous wrapper is performed. Finally, a possible modification of the GALS implementation that avoids local clock generation is considered.

4.2 Overall Structure of the Asynchronous Wrapper

Based on the concept presented in the previous chapter, an asynchronous wrapper can be designed. However, it is hard to realize an optimal configuration of this request-driven circuit. Ideally, a single asynchronous controller can provide the correct operation of the wrapper. However, due to the heterogeneous nature of the wrapper and the relatively complex request-driven protocol that has to be followed, it is very hard to define and fulfil all requirements for the safe operation of such a controller. Therefore, we decided to split the wrapper structure into several major sub-blocks and to distribute the wrapper functionality in order to reduce the controller complexity. Consequently, each sub-block has a very limited task and the sub-block structure is relatively simple. A comprehensive block diagram of a single GALS block, including a locally synchronous module and the asynchronous wrapper, is shown in Figure 4.1. Handshakes are performed in the asynchronous wrapper at three positions. There is an input handshake (signals *REQ_A* and *ACK_A*), an output handshake (*REQ_B* and *ACK_B*), and finally

an internal handshake between input and output port (signals *REQ_INT* and *ACK_INT*). A request-driven clock is generated in the input port and in order to achieve the correct clocking of the system an internal handshake is necessary. In this way, the input port has information about the status of the output port which is essential for the correct clock generation.

We defined the interfaces of the GALS block to the outer world as follows. Firstly, *REQ_A* and *ACK_A* are input 4-phase handshake control signals. Data is transferred via the *Data_in* signal. The data bus is assumed to be stable when *REQ_A* is asserted high. The behaviour of the data lines has to adhere to the “broad” handshake protocol [MUT01]. On the output side of the wrapper, the output handshake control lines *REQ_B* and *ACK_B* and the respective *DATA_OUT* bus are placed. The asynchronous wrapper delivers to the locally synchronous module data signal *DATA_L* and a corresponding control signal *DATAV_IN* that defines the validity of the input data in the respective clock cycle. Correspondingly, on the output side, the LS module should generate the signal DATA_OUT and control signal *DATAV_OUT*. Additionally, the signal *DATAV_OUT* must indicate validity of data for the next and not for the current clock cycle.

The asynchronous wrapper consists of the input and the output port(s), a data latch, a pausable clock generator, a time-out generator, and a clock control circuit, as can be seen from Figure 4.1.

The purpose of the input port is to perform the input and internal handshake and to guarantee safe input transfer of the data. Furthermore, the input port resets the time-out and clock control circuitry after every handshake by activation of signal *RST*. The input handshake request signal *REQ_A* is not directly used in the input port, because this signal is arbitrated at the time-out generator. Therefore, a derived signal *REQ_A1* is used for the handshake procedure in the Input port. This signal behaves deterministically and cannot cause some hazard or deadlock of the input port controller.

The output port supports the data transfer to the following GALS block and performs the output handshake. While a particular data transfer is not finished the output port will stretch the clock. Alternatively, for the clock cycles when the output data transfer is not needed the output port just supports internal handshake.

The time-out generation unit is implemented with a small number of hardware components. This unit operates as a counter of the local clock on negative edges. The time-out signal is generated when the counter is triggered for a sufficient number of times without being reset. This reset is activated by the input port whenever the input handshake is performed. Consequently, the time-out signal can be generated only when an input handshake channel has been inactive for a sufficient amount of time. Additionally, the time-out unit performs the necessary arbitration for the safe operation of the input port.

A local clock generator (LCG) generates triggers for the time-out measurement. Moreover, when time-out is reached, it generates clocks for the LS module. The pausable clock generator is

implemented as a tunable ring oscillator. Tunability is a very important property of the proposed LCG in order to calibrate the clock frequency and avoid the effect of process, temperature or voltage changes. The LCG can be stretched both from input and output ports. From the input side the lines $REQI_1$ and $ACKI_1$ and from the output side the signal *Stretch* are used for this purpose. Signal *LCLKM* is the output of the pausable clock generator, gated with signal *ST* as shown in Figure 4.1. *LCLKM* is delivered to the Input port where is used for generation of the data valid signal and in the clock control circuitry (that counts the number of locally generated clock cycles). From Figure 4.1 it can be seen that signal *INT_CLK* is generated by an OR operation of signals *REQ_INT* and *LCLKM*. *REQ_INT* and *LCLKM* are mutually exclusive. Therefore, glitch-free operation of *INT_CLK* is preserved. The signal *INT_CLK* is delayed in comparison to *REQ_INT* and *LCLKM* for the time of the OR operation together with the delay of the clock tree.

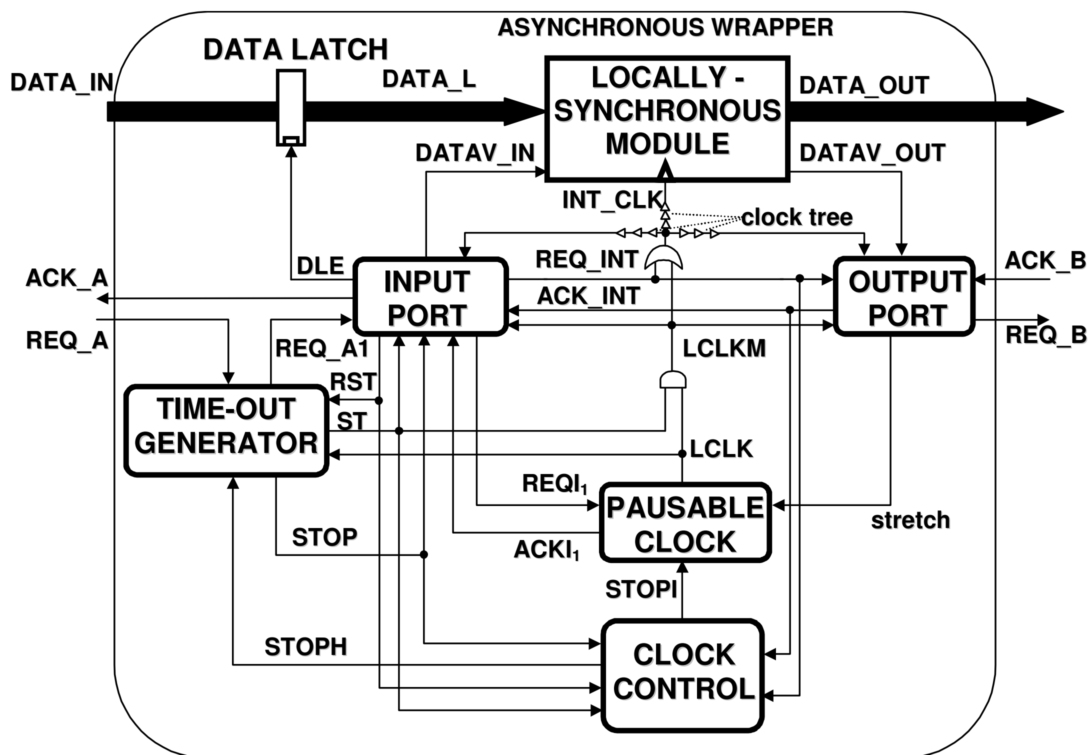

Figure 4.1. Block diagram of the proposed asynchronous wrapper

The clock control unit is designed to increase power-saving capabilities. The role of this block is to count the number of locally generated clock cycles. When the LS pipeline is empty signals *STOP* and *STOPI* will indicate that the local clock generation should be disabled. The internal request signals *REQ_INT* and *RST* are used for resetting the clock control block.

Input data is buffered in a transparent latch. This is needed to prevent metastability at the input of the locally synchronous module. The operation of this data latch is controlled by signal *DLE* with the latch being transparent when *DLE* is asserted. Signal *DLE* is asserted after a transition of the local clock, when previously latched data is already written into the register stage of the locally synchronous

module. The input of the locally synchronous module is assumed to have no additional registers. The data can be directly fed to a combinatorial logic block in front of the first register stage. In the following sections all blocks shown in Figure 4.1 will be described in detail.

4.2.1 Pausable Clock

This local clock generator is implemented as a ring oscillator, and is shown in detail in Figure 4.2. The structure of the generator is described in [TAY00, MUT01]. Generally, a pausable clock generator consists of a delay line, a C-element, an arbitration section and one NOR-gate for enable/disable function.

The delay line is designed in such way that the tunability of the clock generator is achieved. For this property, a slice architecture of the delay line is used. The principle of delay element slicing is shown in Figure 4.3a and the structure of the particular delay element is given in Figure 4.3b. With such architecture, it is possible to choose how many delay elements will be included in the delay chain. The more delay slices are active, the higher the delay of the signal (i.e. lower clock frequency) will be achieved. Additionally, to forward and backward data lines, every delay slice has its own control signal that will either enable further the forward path of the delayed signal or create a shortcut between forward and backward delaypath.

The difference between the proposed implementation in [TAY00] and the one used in our approach is that we use a simplified delay element. With that simplification only a coarse control signal (*CC* in Figure 4.3b) is used for the delay element. For our request-driven GALS technique, in general, it is not needed to have perfect matching of the operating frequency. Therefore, an additional signal for a fine control and respective circuitry for support of this feature, as proposed in the literature, is not needed. The step size of the generated frequencies is estimated to 180 ps, for the IHP 0.25µ CMOS process at 25 °C operating temperature. However, the step size can be easily changed if different standard cells are used for delay slice implementation.

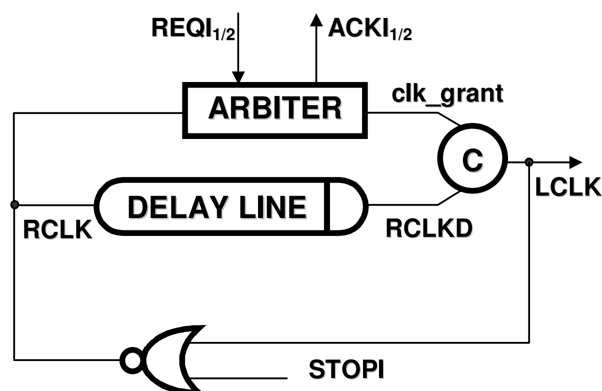

Figure 4.2. Pausable clock generation

The arbitration section consists of two mutual exclusion elements as depicted in Figure 4.4. One is used to control the request from the input port of the asynchronous wrapper and the other for the output port. The purpose of the MUTEX blocks is to arbitrate between incoming request signals and clock generation. In general, if a request arrives when line *RCLK* is high, an acknowledge will not be granted. Alternatively, if request is a high, the clock will be stretched until the request line is released. In the case that both inputs of one of the MUTEX blocks go to high at the same moment, the MUTEX will “toss the coin” and only one of the outputs will be granted.

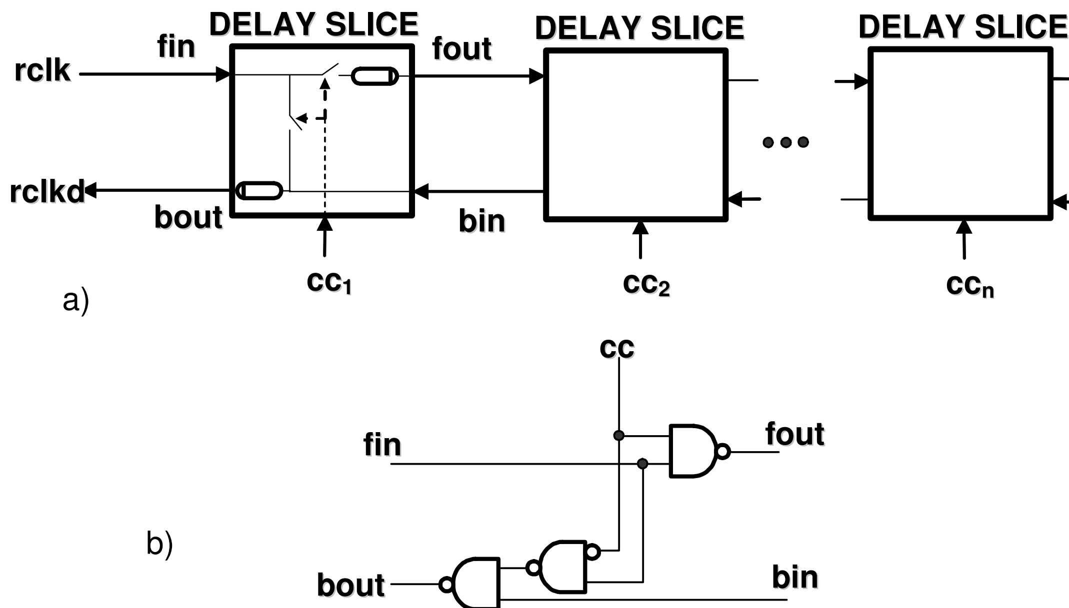

Figure 4.3. Tunable delay concept (a) and delay element structure (b)

In the current implementation with one input and one output port, two MUTEX blocks are used in the arbiter. One is fully used by the input port with input signal *REQI1* and the respective output signal *ACKI2*. For the output port a return information (acknowledge of the request) is not needed, so only one signal *Stretch* is connected. When *Stretch* is high, new clock cycles will not be allowed.

Differing to the implementation in [TAY00, MUT01] it is possible to stop the clock with an additional signal *STOPI* that is activated in two cases: Firstly, immediately after reset, to prevent the activation of the clock oscillator before the first request signal arrives at the local block. Secondly, after time-out, i.e. when the number of local clock cycles is equal to the number of cycles necessary to output all valid data tokens stored inside the pipeline. In this situation, the local clock is stopped to avoid unnecessary waste of energy.

To conclude, the ring oscillator of the clock generation unit can operate in three principle modes: sleep mode, time-out measuring mode, and local clock generation mode. In sleep mode the stop signal *STOPI* disables the clock generator’s operation. In time-out measuring mode the clock generator is used for triggering the time-out block. Finally, the local clock generation mode is activated

after occurrence of the time-out. In this mode, the clock generator drives the locally synchronous module in order to flush all pipelined data.

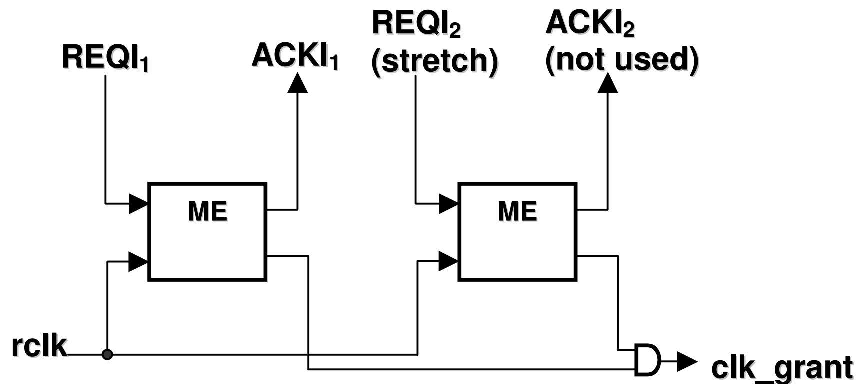

Figure 4.4. Arbiter used for GALS wrapper

4.2.2 Clock Control Unit

Activation of the pausable clock is controlled by a block named *Clock control,* the structure of which is shown in Figure 4.5. This block generates two output signals: *STOPI* and *STOPH*. Signal *STOPH* is implicitly used as a control signal for the asynchronous finite state machine (AFSM) in the input port. Eventually, when *STOPH* is asserted the local clock will be halted. This signal is activated when the counter, clocked with the local clock, reaches the number equal to the depth of the synchronous pipeline. The counting number should be predefined during the design process. After that, clocking of the locally synchronous module is not needed, and clock generation should be deactivated. With this solution a significant amount of energy can be saved. Signal *STOPI* is directly used as a control signal for the ring oscillator. This signal should hold the value of signal *STOPH*. One D-flip-flop will keep this signal in asserted state until a new request arrives, even when signal *RST* resets *STOPH*. A new request (*REQ_INT* high) resets this flip-flop and value of signal *STOPI*. However, it is hazardous to directly drive the *STOPI* flip-flop with the *STOPH* signal. If we do that there is a certain risk of deadlock in the input-port. Therefore, signal *STOPI* is arbitrated with signal *REQ_A,* in the time-out generator and only the signal after arbitration (*STOP*) can be safely used for the *STOPI* generation.

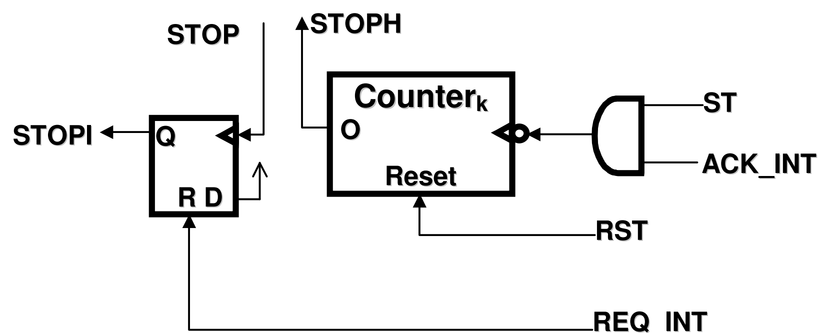

Figure 4.5. Clock control circuit

This is just one possible way of controlling the ring oscillator. This solution relies on the most general construction of the locally synchronous module and allows completely autonomous operation of the asynchronous wrapper. If, in any case, this control is not satisfying, it is always possible to directly control the stopping of the clock via some synchronously generated signal that indicates whether the local pipeline is empty.

4.2.3 Time-out Generation

The time-out generation unit is implemented with a relatively simple hardware as shown in Figure 4.6. Nevertheless, it has to support a complex operating scenario. Generally, it consists of one counter (*Counter*$_n$ in Figure 4.6*)* that counts the number of negative edges of the local clock. This counter is designed as a standard synchronous counter. When the final value is reached it eventually generates a time-out signal *ST*. The counter's reset signal *RST* is activated once during every input port handshake. *RST* and the clock signal (extracted from inverted signal LCLK) are not a priori mutually exclusive. This potentially gives rise to a metastable behaviour of this counter. To avoid metastability, one mutual exclusion element must be inserted which resolves the simultaneous appearance of the rising clock edge and the falling edge of the reset signal. Additionally, one flip-flop is inserted to reshape the signal waveform of *LCLK* into pulses with much shorter duration than *RST*, allowing that the reset cycle actually resets the time-out generator in every handshake.

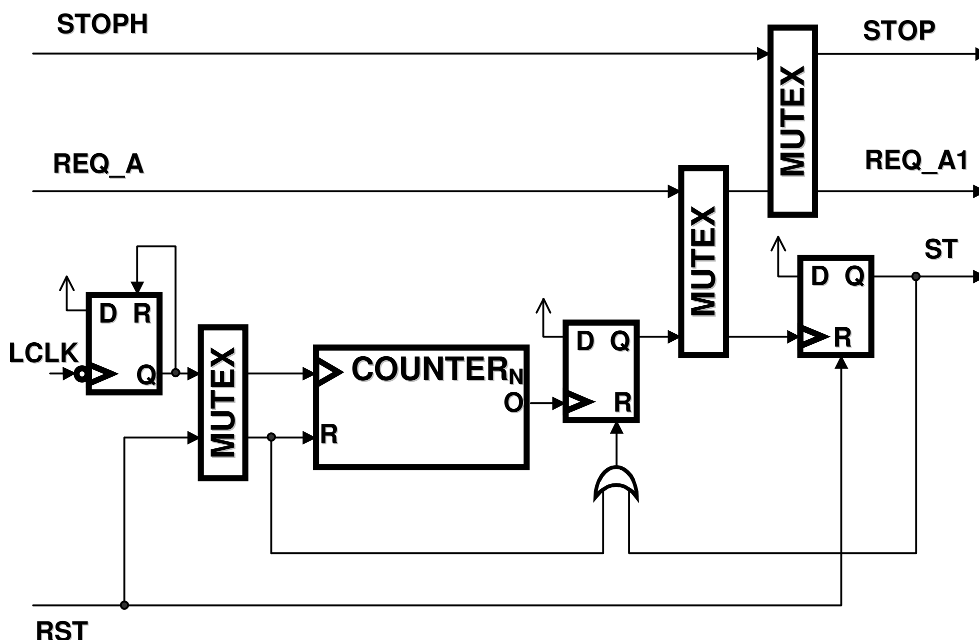

Figure 4.6. Time-out generation circuit

Another situation that must be dealt with is the appearance of an external request signal simultaneously with the time-out signal (*REQ_A* and *ST*, respectively, in Figure 4.6). This condition may violate the assumed burst mode operation and cause erroneous operation of the input port

AFSM. In order to resolve this potential conflict, an additional mutual exclusion element is added. To make the line *REQ_A1* available for most of the time, the time-out signal entering the mutual-exclusion circuit should be active for only a very short period of time. This behaviour is achieved using two flip-flops (FF) as shown in Figure 4.6. The first FF is set to ‘1’ when time-out occurs i.e. the counter output is ‘1’. When after arbitration time-out is granted, the second flip-flop is clocked. Clocking the second flip-flop activates signal *ST.* This in turn resets the first FF, allowing fast propagation of any external request signal *REQ_A* to the AFSM in the input port. Additionally, one mutex element is placed to arbitrate between signals *REQ_A* and *STOP*. Simultaneous appearance of both signals may lead to a hazard in the input port AFSM and must be avoided.

4.2.4 Input Port

The input port mainly consists of an input controller along with some supporting circuitry shown in Figure 4.7. The input controller must guarantee safe data transfer and is implemented as an AFSM working in burst mode. Port specifications of the input controller are given in Figure 4.8. The input port, described here is a pull-type port. A push-type port can be constructed in a similar way with some minor modifications. In the normal mode of operation it just reacts to the input requests and initiates ‘clock cycles’ (signal *REQ_INT*) for every incoming request. Therefore, the start of an input handshake (signals *REQ_A1* and *ACK_A*) will initiate the internal handshake as well (signals *REQ_INT* and *ACK_INT*). The internal request will then be used as a clock source for the locally synchronous module. During this “normal” mode of operation the time-out function will be regularly reset with signal *RST*.

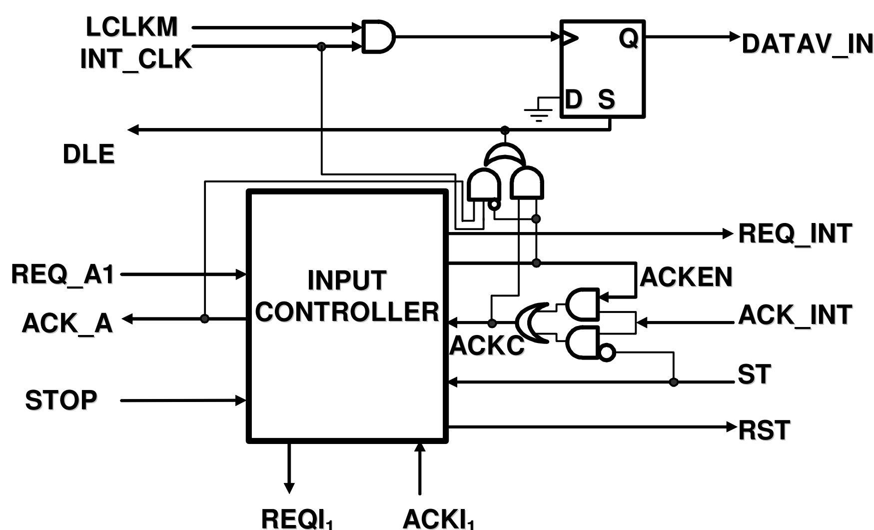

Figure 4.7. Input port block diagram

If there is no activity on the request line for a certain period of the time, signal *ST* is activated (time-out). This will disable further activities on the internal acknowledge line (signal *ACKC*) because signal

ST will disable the rising edge of acknowledge. Now the circuit waits for one of two possible events. The first one is the completion of the expected number of internal clock cycles (indicated by signal *STOP*). This would bring the AFSM back into its initial state. The second possible event is the appearance of an input request in the middle of the pipeline flushing process. This will activate a so-called "transitional mode". The already started local clock cycle must be safely completed and clock generation must be handed over to the input requests.

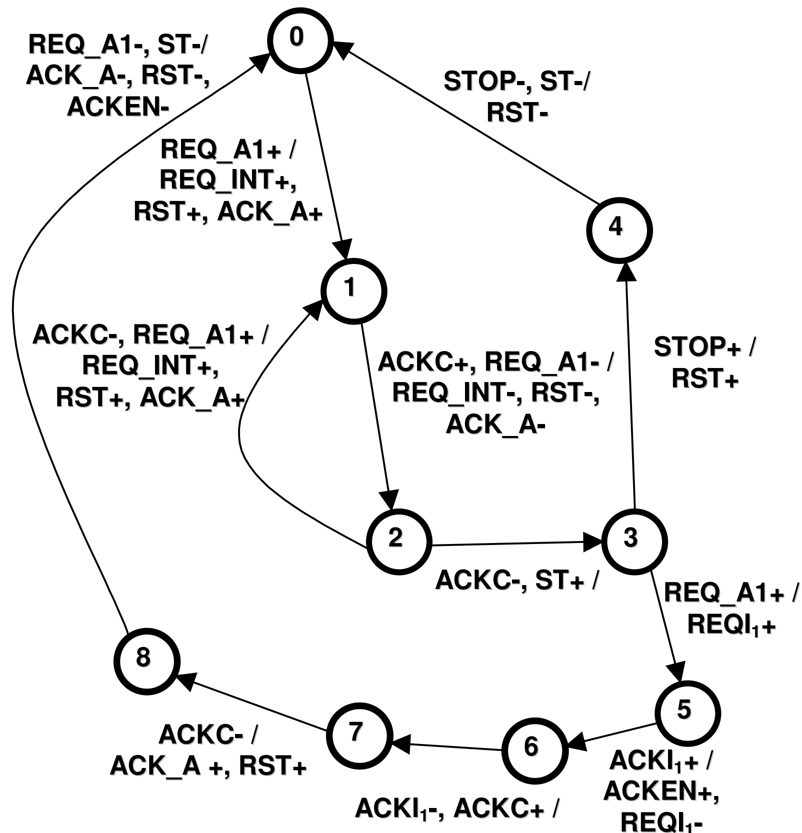

Figure 4.8. Specification of the input controller AFSM

In our implementation, the input controller is connected with a circuit adhering to a 'broad' 4-phase handshake protocol. For designing the AFSM, the 3D tool [YUN99a] was used. The main reason was the possible application of the extended burst mode specification. However, in the end we used only burst mode specifications. The logic equations generated by 3D from the specifications in Figure 4.8. are listed below:

$$REQ_INT = REQ_A1 \cdot REQ_INT + \overline{ACKC} \cdot REQ_INT + REQ_A1 \cdot \overline{ACKC} \cdot \overline{ST} \cdot \overline{ACKEN}$$

$$ACK_A = \overline{ACKC} \cdot REQ_INT + \overline{ACKI_1} \cdot ACKEN \cdot \overline{ACKC} \cdot ST \cdot \overline{Z0} + REQ_A1 \cdot RST$$
$$+ REQ_A1 \cdot \overline{ACKC} \cdot \overline{ST} \cdot \overline{ACKEN}$$

$$ACKEN = ACKI_1 + REQ_A1 \cdot ACKEN + ACKEN \cdot \overline{ST}$$

$$RST = STOP + \overline{ACKC} \cdot REQ_INT + REQ_A1 \cdot RST + ST \cdot RST + \overline{ACKI_1} \cdot ACKEN \cdot \overline{ACKC} \cdot ST \cdot \overline{Z0} + REQ_A1 \cdot \overline{ACKC} \cdot \overline{ST} \cdot \overline{ACKEN}$$

$$REQI_1 = REQ_A1 \cdot ST \cdot \overline{ACKI_1} \cdot \overline{ACKEN}$$

$$Z0 = ACKI_1 \cdot \overline{REQ_A1} \cdot ACKC + \overline{REQ_A1} \cdot \overline{ST} \cdot Z0 + \overline{ACKC} \cdot ACKEN \cdot Z0 + ACKC \cdot \overline{ACKEN} \cdot Z0$$

Z0 is an internal signal, added to provide hazard-free operation of this AFSM. In order to guarantee the correct functioning of the AFSMs, it is needed to adhere to the “burst” mode operation. This mean that inputs arrive in bursts in any order (but glitch-free). Once an “input burst” is complete, the machine generates glitch-free “output burst” and a concurrent state change to a new state.

However, a new input burst must not arrive until the previous input burst has been fully processed and the AFSM has stabilized from a previous input and state change [NOW02]. Having in mind this particular implementation, “burst mode” requirements are fulfilled due to the slow environment of the AFSM. This fact is confirmed by extensive simulations under different timing conditions.

The additional circuitry in Figure 4.6 is to disable, during local clock generation, the acknowledge signal *ACK_INT* generated by the output port. *ACK_INT* is enabled once again after transition from local clock generation to the request-driven operation by activation of signal *ACKEN*. Signal *ACKC*, which is fed to the input port AFSM as shown in Figure 4.7, is also used as a control signal *DLE* for the input data latch. Signal *DLE* is active when a new input transfer is being performed. Also this signal will enable the data latch only when the input data is stable. We have searched for a long time the right combination of the signals that guarantees that all requirements are fulfilled. Finally, for the generation of this signal a combination of four signals as in the following logic equation is used:

$$DLE = INT_CLK \cdot ACK_A \cdot \overline{ACKEN} + ACKC \cdot ACKEN$$

The first term in the OR operation will enable the data latch during a normal input handshake when both *INT_CLK* and *ACK_A* are high. That combination will result in a relatively short enable signal. Another term is generated with signals *ACKC* and *ACK_EN* and it should support the DLE generation during transition mode.

Furthermore, there is an additional flip-flop for generating the data-valid input signal for the synchronous block. The data valid signal is enabled when there is new data at the input port and disabled when the input handshake lines are idle and the GALS block operates in local clock

generation mode. Signal *DLE* activates the data valid signal, and for the deactivation of this signal a combination of the signals for internal clock *INT_CLK* and local clock *LCLKM* is used.

In figures 4.9, 4.10, 4.11, and 4.12, four possible scenarios of input port operations are given. In Figure 4.9 the usual start from the “zero” AFSM position is given. The input port is activated when signal *REQ_A1* arrives. Accordingly, handshakes at the input and internal nodes are performed. The asynchronous wrapper operates in request-driven mode. The AFSM of the input port is either in state “one” or in state “two” in this phase.

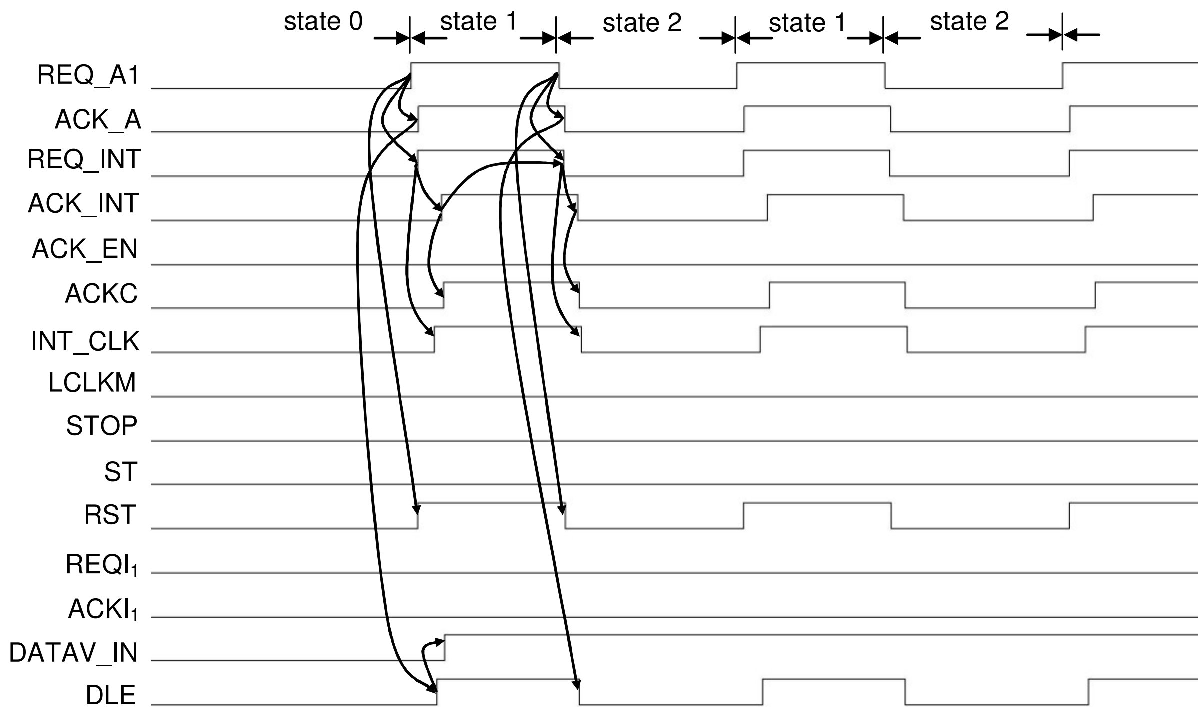

Figure 4.9. Phase 1 – start of the request-driven mode

In the Figure 4.10, a transition from request-driven mode to local-clock generation mode is shown. This transition will occur when, after some period of inactivity on the *REQ_A1* line, signal *ST* becomes active. Accordingly, the source of the LS clock will no longer be signal *REQ_INT* but signal *LCLKM*. Figure 4.11 shows the transition of input port AFSM from state “3” to state “0”. That corresponds to the situation when the internal LS pipeline is empty and clock generation can be stopped. Disabling the local clock will start the activation of signal *ST*.

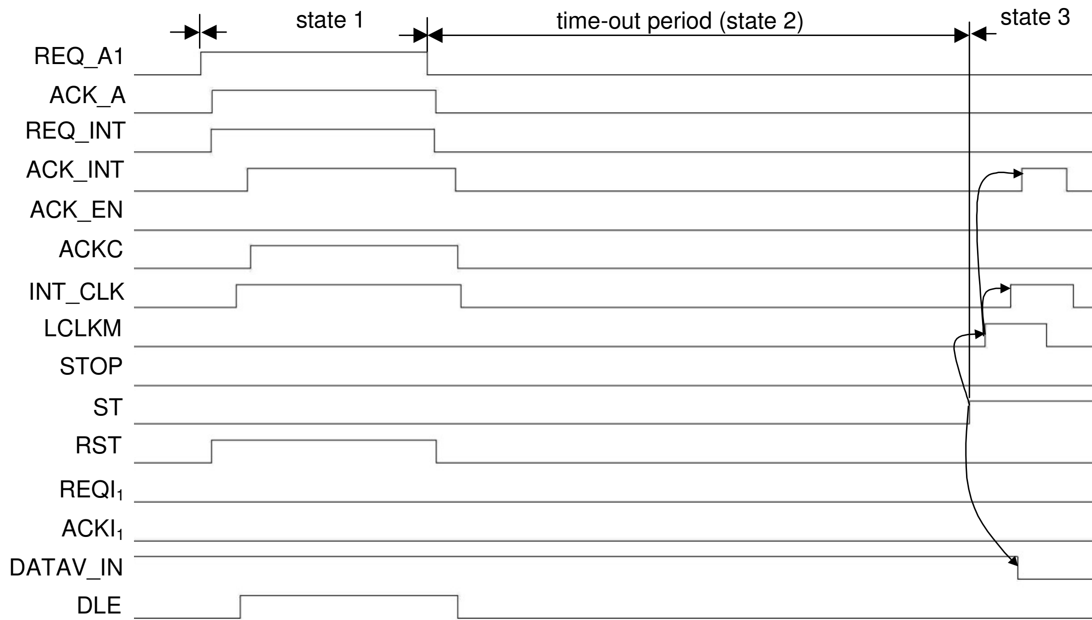

Figure 4.10. Phase 2 – start of the local clock generation mode

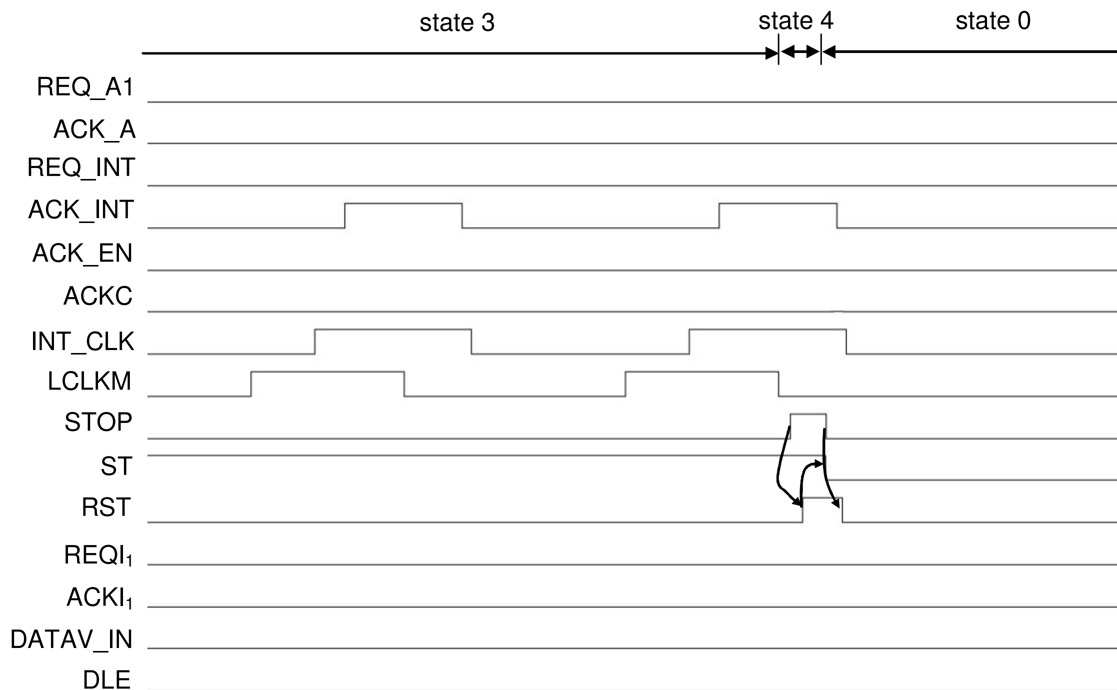

Figure 4.11. Phase 3 – local pipeline is empty

The fourth simulation (Figure 4.12) shows the most complicated scenario. There, the local clock-generation mode has to be disabled when a new request appears at the input. In order to safely perform this operation, a transitional period is provided. In this transitional period, first the current local clock cycle has to be finished and then the control of the clock can be handed over to the request input.

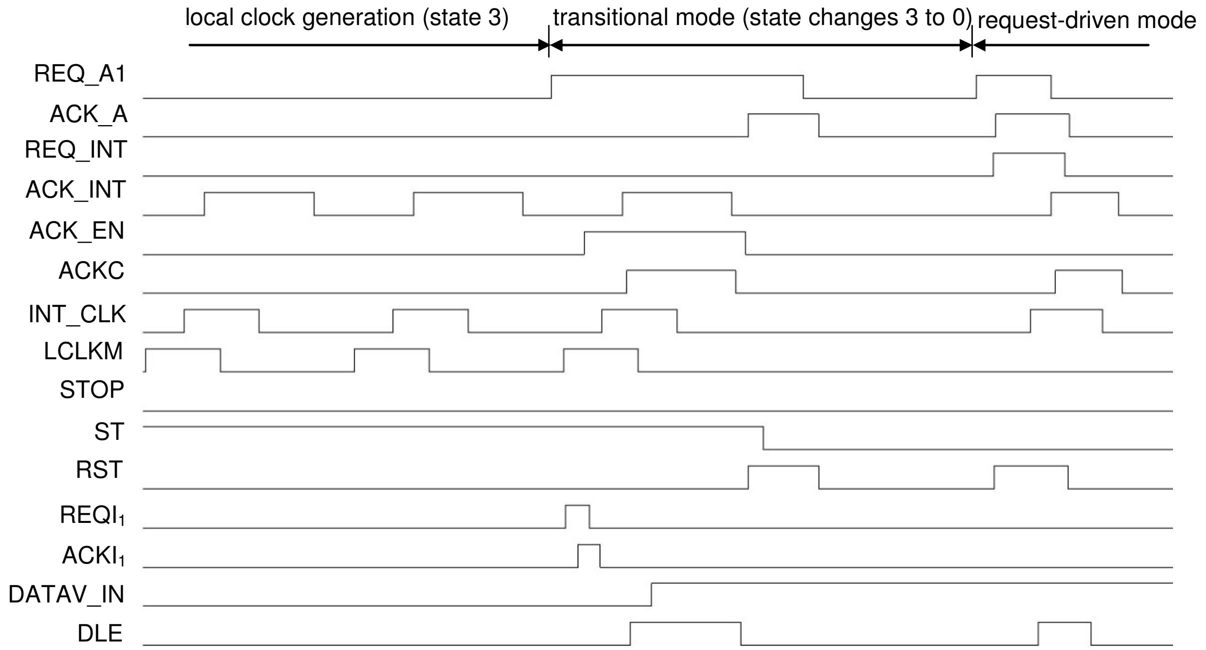

Figure 4.12. Phase 4 – during local clock generation an input request appears

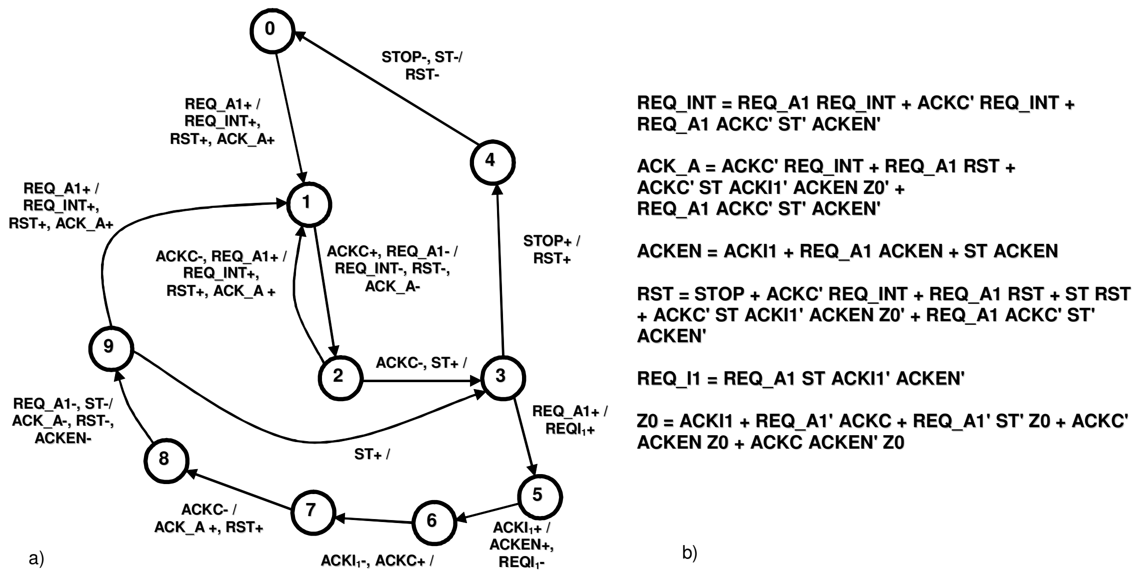

Figure 4.13. a) Specification of the input controller AFSM that absorbs single tokens and b) equations of the corresponding hazard-free logic (symbol ‘ denotes inversion)

The proposed input port AFSM assumes bursty data and does not cover all different scenarios when a single token approaches the asynchronous wrapper. For example, in local clock generator mode, the arrival of a single request will lead to a transition from state 3 to state 0 (state changes 3→5→6→7→8→0). However, the time-out function will not be available in state 0 if we apply the originally proposed AFSM. However, it is possible to change and adapt the input port AFSM in order to

adapt to the different applications and scenarios. The modified asynchronous wrapper is given in Figure 4.13.

On the other hand, the proposed change will make the AFSM more complex and we did not use it in our GALS circuits. Furthermore, the scenarios that are additionally covered with the updated AFSM were not of interest for the intended applications.

4.2.5 Output Port

The purpose of the output port is to safely perform the output handshake of the GALS block. Therefore, a generation of the new clock cycle will be disabled until the handshake is finished. When there is no output data to be transferred, the output port has only the passive role to acknowledge any internal request. A block diagram of the output port is shown in Figure 4.14.

It mainly consists of an AFSM output controller and two additional flip-flops. Those flip-flops are used to condition the signals indicating data output valid (*DOV)* and not valid (*DONV)* for use in the AFSM. Since the AFSM is event- and not level-driven, the level-based signal *DATAV_OUT* is transformed into the two event based signals *DOV* and *DONV*, by strobing them with the local clock signal *INT_CLK.* Signal *DATAV_OUT* is generated in the locally synchronous module and it should indicate whether the next clock cycle generates output data or the output circuitry will stay idle. Consequently, those two flip-flops for generating signals *DOV* and *DONV* are in general part of the synchronous block and their output must be synchronised with the output data. Data valid signals are triggered with the local clock. Therefore, their input should be the validity signal of the output data for the next cycle and not for the current one.

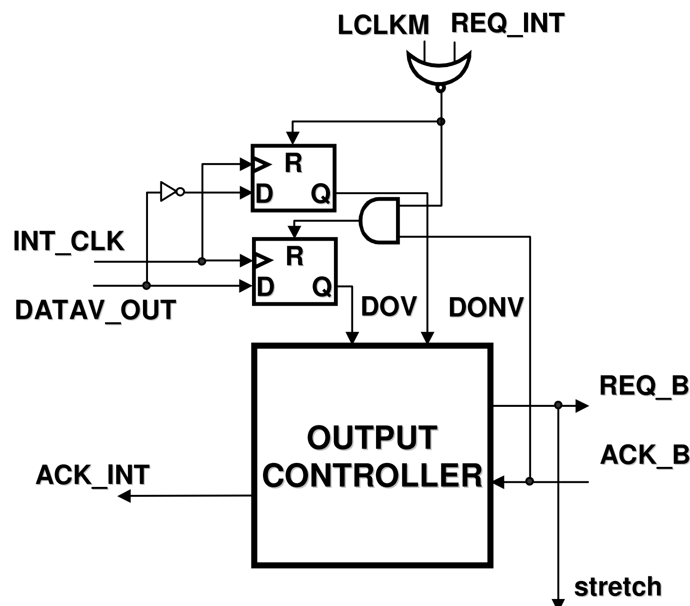

Figure 4.14. Output port

The AFSM specifications of the output port controller are given in Figure 4.15. If there is no valid data at the output (*DONV* is activated), an internal request is immediately acknowledged by activating signal *ACK_INT*. When output data is to be transferred to the next GALS block (*DOV* is activated), an output handshake (signals *REQ_B* and *ACK_B*) must be performed. When *DOV* is activated then the local clock must be stretched using signal *stretch*, until an output handshake is performed. This will prevent a new clock cycle before completion of the output data transfer.

The logic equations for a hazard-free AFSM implementation of the output controller, again generated using the 3D tool [YUN99a], are listed below:

$$REQ_B = \overline{ACK_B} \cdot REQ_B + DOV \cdot \overline{Z0} + \overline{ACK_B} \cdot DOV$$

$$ACK_INT = \overline{ACK_B} \cdot REQ_B + DOV \cdot \overline{Z0} + \overline{ACK_B} \cdot DOV + \overline{ACK_B} \cdot DONV$$

$$Z0 = ACK_B \cdot Z0 + ACK_B \cdot \overline{DOV} + \overline{DOV} \cdot \overline{DONV} \cdot Z0$$

Z0 is an internal signal added to achieve hazard-free behaviour of the AFSM. The burst mode requirements are confirmed by the large set of simulations covering different modes of wrapper operation.

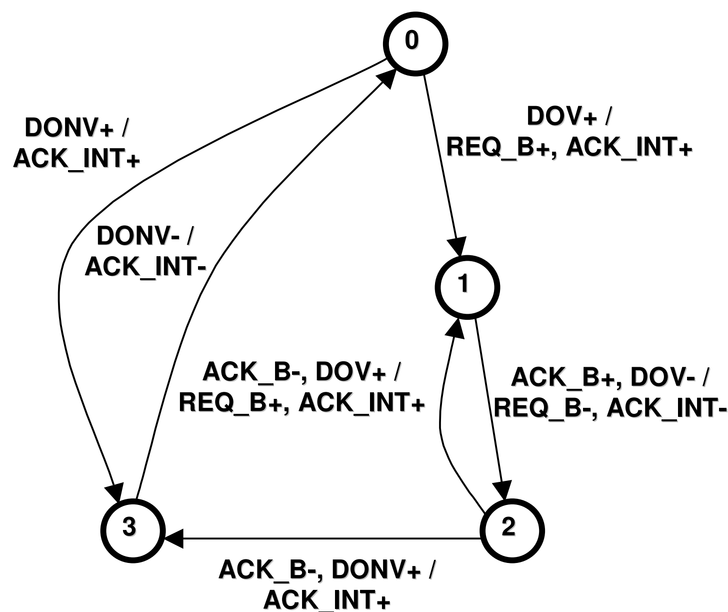

Figure 4.15. Output controller specification

Figures 4.16 and 4.17 show the operation of the output controller. In Figure 4.16, the output port performs the data output. This corresponds to the transition between states “1” and “2” of the output port AFSM. In that case, the internal handshake is coupled with the output handshake. Figure 4.17 gives the function of the output port when there is no data to be output. In that case, every request

coming from an internal handshake channel is confirmed. The output port, described here, is a push-type port. The architecture of a pull-type port would be very similar, with only minor modifications.

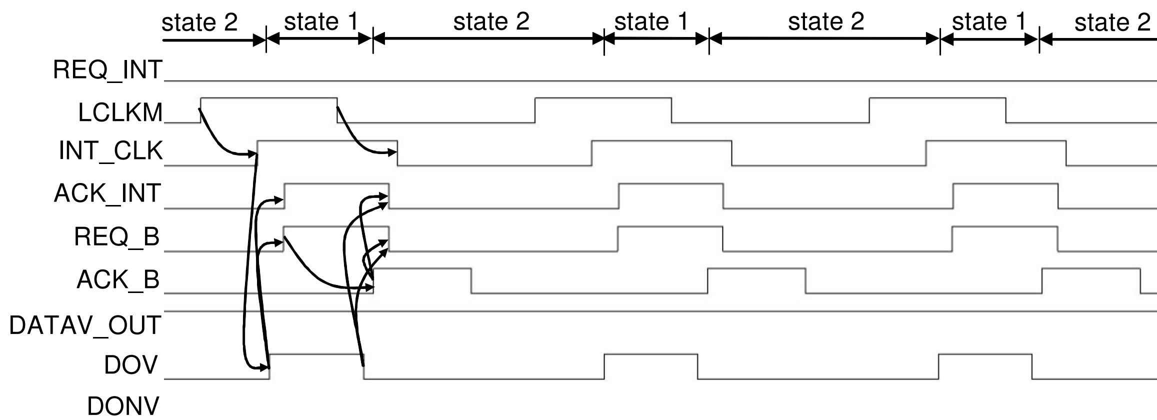

Figure 4.16 Operation of the output port when there is data to be output

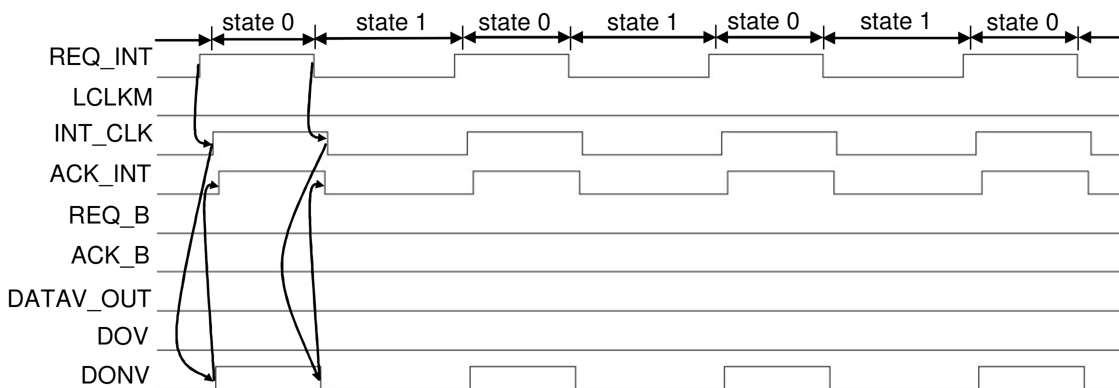

Figure 4.17 Operation of the output port when there is no data to be output

The implemented circuit is based on worst-case assumptions. It allows that in the middle of flushing the pipeline, a new request at the input port may appear. However, there are some cases where the operation of the output wrapper will not always grant safe data transfer. Problems may appear when the clock is generated from the local oscillator and the output port is connected to a very slow token absorber. From the AFSM specification it can be seen that the clock is stretched only in one half of the period (when *REQ_B* is high). After setting *ACK_B* to high (during the handshake operation) the local clock will be released from the stretch control. If, after that, setting *ACK_B* to low takes two much time (longer than one period of the local clock generator) it may happen that we have more than one clock cycle between consecutive handshakes. Under normal conditions that will never happen, but if it is necessary to prevent such behaviour it is possible to add simple extensions to the proposed architecture. For example, the output port could be structured as in Figure 4.18. The proposed circuitry will ensure perfectly safe data transfer between two GALS blocks. For our purposes, we have used simpler circuitry to increase the throughput.

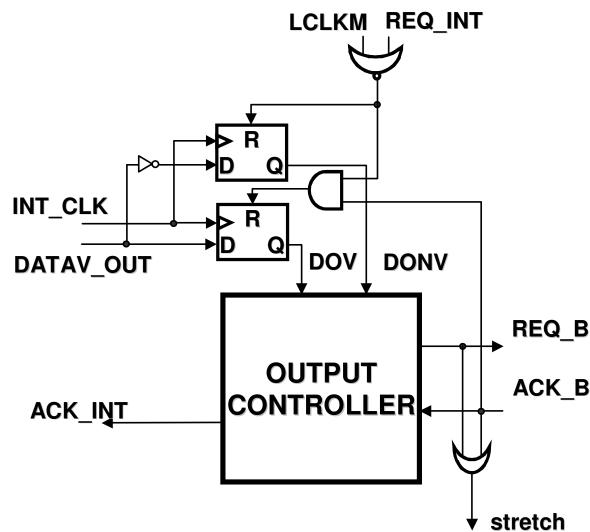

Figure 4.18. Safe output controller architecture

4.2.6 Mutual Exclusion Element

For the purpose of arbitration in asynchronous circuits, it is necessary to use Mutual Exclusion elements (MUTEX). The block diagram of the MUTEX is given on Figure 4.19a. When both inputs are low, both outputs are low as well. When one input goes high, the corresponding output also goes to high and the other output cannot be set until the first input is released. If both inputs go to high at the same moment, the circuit tosses a coin to choose one of the outputs to go to high and the other should remain low. In order to perform this function, Seitz proposed in [SEITZ80] an efficient circuit structure given in Figure 4.19b. Unfortunately, a standard cell based on this schematic is very seldom in libraries. So, in [M0098, KES97] a simple solution that uses only standard gates (Figure 4.18c) is proposed. In this structure, 4-input NOR gates have two main functions. One is to filter the glitches on the NAND outputs with considerable capacitance on the NOR gate, and the other one is to shift the threshold voltage. We have used this solution in our implementations.

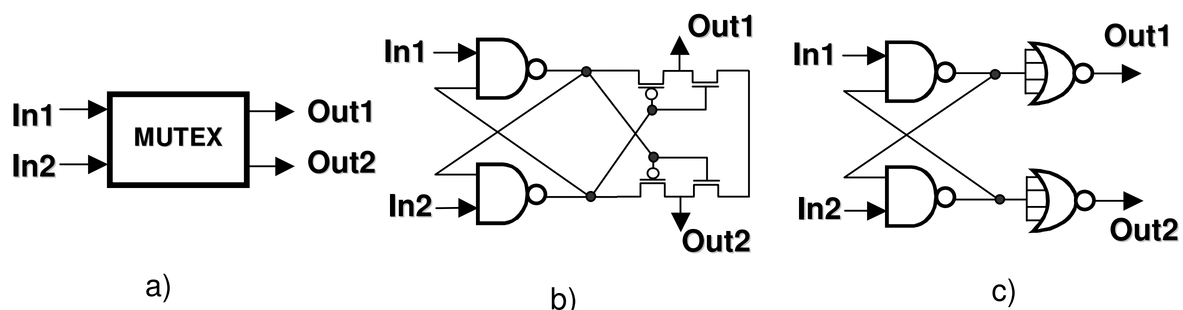

Figure 4.19. Mutual-Exclusion element, from block diagram to implementation

4.3 Formal Analysis of the Asynchronous Wrapper

A formal analysis of the proposed asynchronous wrapper was performed [STA05], in order to validate the correctness of the proposed concept. Additionally, we wanted to investigate possible hazards, deadlocks and races in the system. A Petri-net model was built for every gate. It expresses possible output signal changes and the next state of the gate as a consequence of the input signal changes and the current state of the gate. The asynchronous wrapper contains only a small number of different library elements (basic logic gates, flip-flops, counters, mutex, C-elements). For modelling the gates a methodology similar to the one proposed in [GEN92] is used. All functional properties of the elements are described via the Petri-net model; thus the complete wrapper can be structured by plugging the respective models together.

For wrapper verification the Petri-net based model checker LoLA [SCHM00] was used that utilizes powerful state space reduction techniques. In a first step, the net size was reduced. The model completely ignores any timing relationships caused by switching and propagation delays. Therefore, through introduction of causalities into the model, we can simplify the complete structure. Secondly, sequenced gates are merged, e.g. two AND gates whose output signals are the input signals of an OR gate. Unfortunately, the full state space did not fit into the memory of a computer. However, the LoLA tool has features that allow checking for hazards even in this case.

In order to find potential hazards all possible hazardous paths have to be calculated and their reachability has to be checked. In the case of a potential hazard, LoLA writes out a path from the initial state to this marking. Each of the critical paths was analyzed. Afterwards, these scenarios were checked by simulation, in order to confirm whether a risky scenario might occur in reality. During the analysis, some possible hazards and glitches were found. After this analysis, the asynchronous wrapper was modified in order to avoid hazardous behaviour. To summarize, the formal analysis was very useful to verify the proper operation of the request-driven GALS method and to clearly define the timing constraints for the wrapper implementation.

4.4 Externally Clocked Asynchronous Wrapper

In the previous chapter we have already mentioned that it is possible to avoid using the local clock generators (i.e. ring oscillators) in conjunction with the request-driven GALS solution. Furthermore, the basic concept of the necessary wrapper adaptation was introduced.

The implementation of the externally clocked asynchronous wrapper is based on the previously described solution for the wrapper with ring oscillator. The block diagram is given in Figure 4.20. From this figure, it is noticeable that the basic structure is very simple. Most of the blocks are the same, including input and output port, time-out generator, data-latch and clock control circuitry. The only

difference is the introduction of one new block - Clock Management Unit (CMU), as we described in [GRA05]. Consequently, the Local Clock Generation block is not used any more.

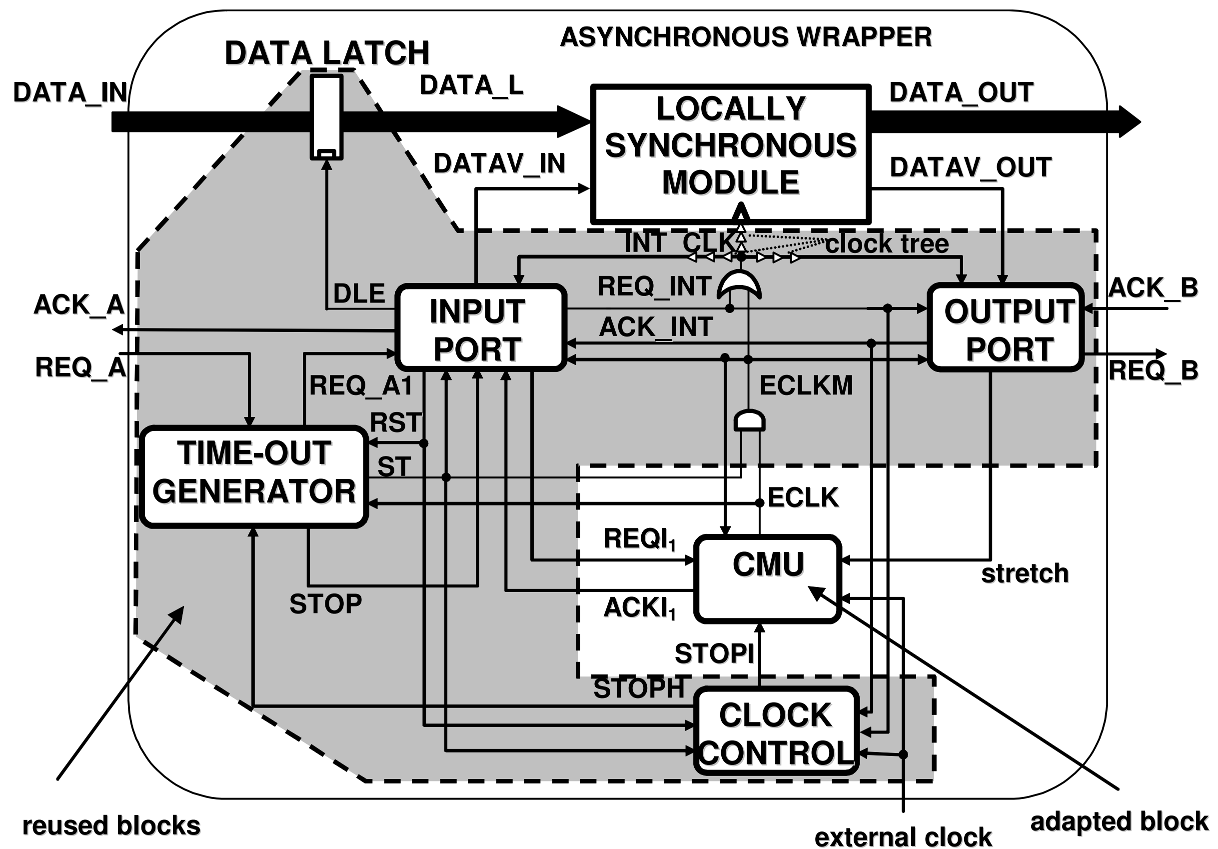

Figure 4.20. Block diagram of the externally clocked asynchronous wrapper

In Figure 4.21, the hazard-free implementation of the CMU unit is given. The proposed CMU structure is very similar to the arbitration unit of the pausable clock generator of the internally driven GALS wrapper. The external clock signal should only be stopped during its logic low phase. In order to prevent metastability, the CMU is responsible for stretching the clock signal while there is handshake activity at the interface of the wrapper. This means that the clock signal is kept low when data is received at the input of the wrapper, during the time-out phase, and when data is transferred at the output of the wrapper. For clock arbitration and halting, three mutex circuits are used.

Mutex block M1 is used to give the time reference for the $REQI_1$ and $ACKI_1$ signals. Activation of $REQI_1$ will stop the propagation of the external clock. Additionally, mutex block M2 is present to disable the external clock when signals *STOPI* and *stretch* are active.

The role of mutex block M3 and the supporting asymmetric C-elements is to ensure that the clock can be re-enabled again, only when the external clock is on low level. If this is not the case, the arbitration circuitry will delay state change until the external clock changed its value to low. The C-elements C1, C2 and C3 perform the glitch-free generation of the clock enable signal. For example,

when *stretch* goes high, signal *ste* will go low. The precondition for this state change is that *external_clock* is in the low phase. After that, the following scenario happens: C2- → *sti-* → →*cg-* and clock is disabled. When *stretch* goes low then *ste+* → C1+ → C2+ (when *external_clock* is low) → *sti+* → *cg+* and the clock is re-enabled again. In this scheme, a possible race can occur between signals *cg* and *clk_grant* during arbitration. To avoid this hazard, it is needed that the path *external_clock* → M2 → *clk_grant* is faster than path *stretch* →M2 → *ste* → *sti* → *cg*. However, this precondition is easily achievable.

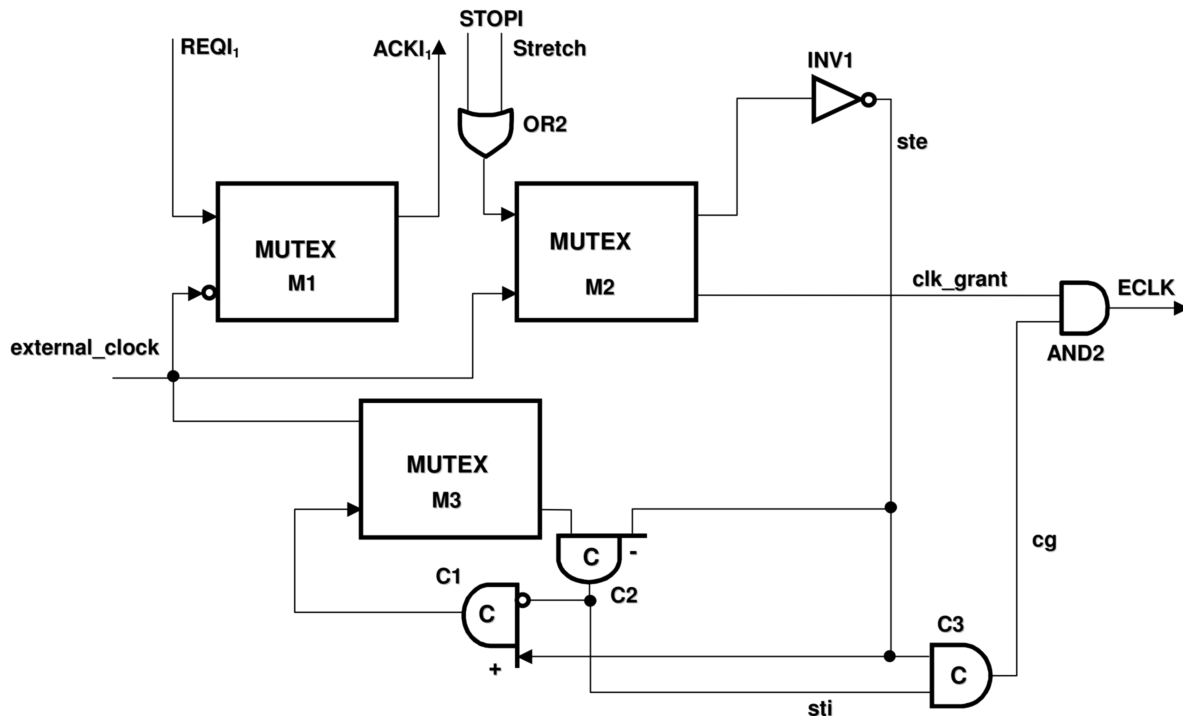

Figure 4.21. Clock Management Unit (CMU)

In general, it is possible to combine both internally and externally clocked solutions in one wrapper. In this case, the wrapper should be structured as it is shown in Figure 4.22. For this solution we have reused the part of the arbitration unit for both internal and external clocking modes. The advantage of this solution is flexibility and fault tolerance. In some cases it maybe better to use embedded ring oscillators (if we want to improve throughput and lower EMI). In some other application, if the power is an important issue, the wrapper can be set to externally driven mode. Additionally, if for some reason the internal solution is not functional, we can always apply the external clock. In this way, fault tolerance is supported.

In Figure 4.23 a sample simulation run of the proposed externally driven GALS system is given. During this run, the system changes into different operational modes (request-driven, externally–driven, transitional and idle mode). It is noticeable that due to stretching many externally distributed clock cycles are discarded in the externally driven mode. This happens, when the external clock

frequency is beyond the maximal throughput of the wrapper. In other cases, with lower external frequency every external clock cycle would be accepted and the operation of the wrapper will be very similar to the operation of the internally driven wrapper.

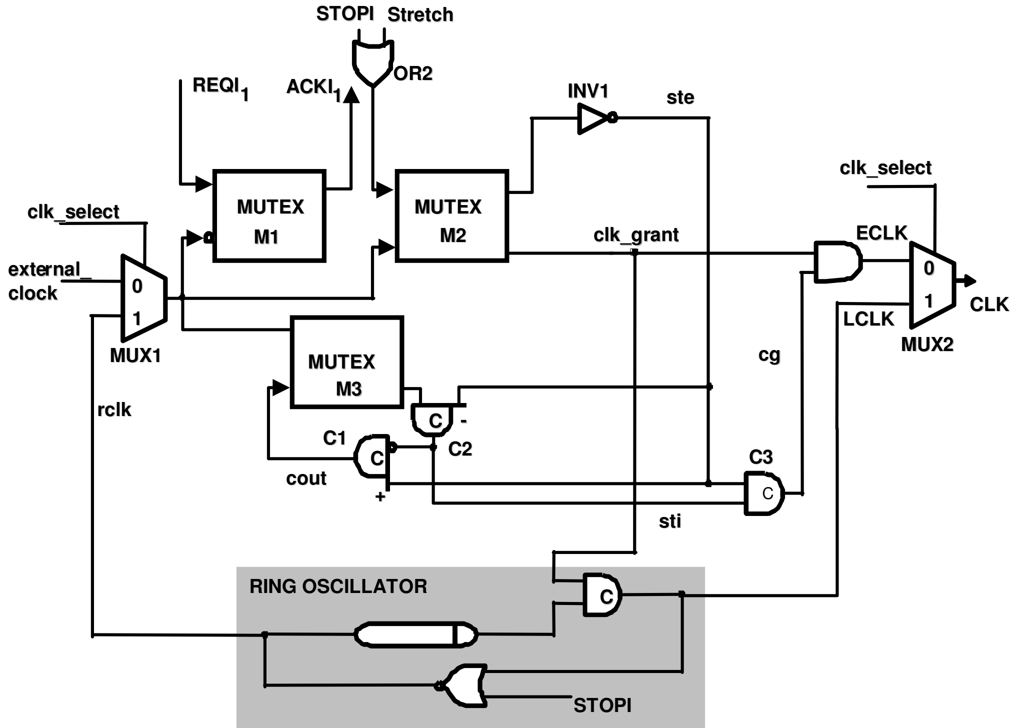

Figure 4.22. Mixed-mode arbiter

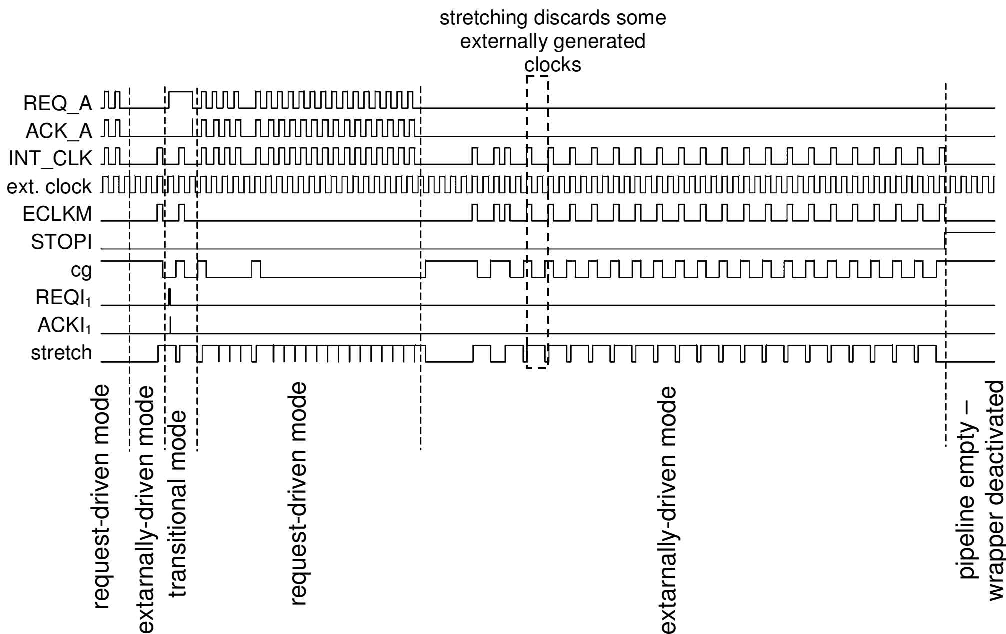

Figure 4.23. A simulation run of the externally clocked GALS system

However, with this solution we can only accept or reject an external clock cycle. It is not possible to delay or pause clock cycle. Accordingly, the performance of the system will be decreased to a certain degree compared with the previously described request-driven GALS technique, based on pausable clocks. On the other hand, with the externally clocked solution, it is possible to avoid problems of local clock calibration and to save power spent in the ring oscillators.

In the following text, we will evaluate in a practical application the local clock generation GALS technique based on ring oscillators. However, in Chapter 7, the externally driven and mixed mode wrappers will be compared to the internally driven wrapper in respect to throughput, area, power consumption and latency.

Chapter 5

GALS Application in Wireless Communication Systems

5.1 Introduction

The research and development of fourth generation (4G) wireless and mobile communication systems is in industrial focus today. Such systems will offer revolutionary new types of services to consumers. For example, broadband wireless networks will provide high-speed data transfer, suitable for video transmission and mobile Internet applications.

The work on GALS implementations presented here is the outcome of a project that aims to develop a single-chip wireless broadband communication system in the 5 GHz band, compliant with the IEEE802.11a standard [IEEE99, GRA01, TRO02, TRO03, KRS03b, KRS03d]. This standard specifies the application of Orthogonal Frequency Division Multiplexing (OFDM) with datarates ranging from 6 - 54 Mbit/s. The datarate depends on the applied modulation techniques that can vary from BPSK over QPSK and 16-QAM to 64-QAM. The physical layer of the WLAN modem includes the analog front-end (AFE) and a digital baseband processor. The design of the required baseband processor for this standard involves a number of design challenges. In particular global clock tree generation, power consumption, testability, EMI, and crosstalk are very demanding issues for the designer. Our proposed GALS architecture is evaluated as possible solution for those problems as described in [KRS05a, KRS04a, KRS04b].

In the following chapter some details about the structure of the baseband processor compliant with IEEE 802.11a are given. After that, the approach for a GALS partitioning and general problems of GALSification of the baseband processor are discussed. Finally, the structure of interface blocks necessary for GALSification is described.

5.2 Baseband Processor Compliant to IEEE 802.11a Standard

Practical implementations of the IEEE 802.11a compliant baseband processor can be achieved in many ways. Mainly, there are two possible implementation styles: a software based DSP solution or alternatively application of dedicated ASIC (Application Specific Integrated Circuit).

In general, software based baseband processing can be done using either a multiprocessor system or a single DSP processor with a number of hardware accelerators. The standard defines very intensive computational activities, and possible software solutions lead to increased power dissipation.

The wireless LAN standard is mainly intended for handheld and mobile applications. Therefore, power consumption is a critical issue. Consequently, we have decided to use a dedicated ASIC for baseband processing. Initially, the baseband processor was implemented as an ASIC working synchronously. This design requires about 700k gates, and clock gating was used as a low-power technique. The architecture is divided in two principle blocks: Transmitter and Receiver block. The baseband processing is separated into two independent dataflow directions: transmit and receive. The transmitter and receiver are implemented as datapath architectures. The block scheme of the designed synchronous baseband processor is given in Figure 5.1. This baseband processor is deeply pipelined and register based without memory structures. To decentralise some timing critical control functions a token-flow approach [BUCK93] for communication between successive blocks was adopted.

The standard [IEEE99] defines the algorithms for receiver and transmitter datapath processing. The initial part of the transmitter stage consists of a small data input buffer, data scrambler, signal field generator, convolutional encoder, interleaver, circuitry for pilot insertion (with pilot scrambler), and a mapper. The standard defines application of several modulation schemes including BPSK, QPSK, 16-QAM and 64-QAM. A 64-point IFFT/FFT (Inverse Fast Fourier Transform / Fast Fourier Transform) processor is used for transfer from time to frequency domain. The IFFT/FFT is a single block used in both, receive and transmit direction in order to minimize the baseband processor silicon area. On the other hand, this solution is more complex for implementation, because of the incomplete decoupling between the transmitter and receiver datapath. Finally, a guard interval insertion block is needed to reduce inter-symbol interference. The final block in the transmitter is the preamble insertion stage. This block generates a preamble which is hard-coded in order to decrease the latency of the system.

The receiver has a more complex structure than the transmitter. The fundamental issues, not specified by the standard, are the mechanisms of synchronisation, channel estimation, and equalization. The solution for this problem is one of the most important outcomes of our project.

The synchroniser has to fulfil the following operations: frame detection, carrier frequency offset estimation, symbol timing estimation, extraction of the reference channel, and data reordering. A block scheme of the synchroniser is given in Figure 5.2. In order to obtain a power efficient design, the

synchroniser structure was split into two mutually exclusive paths: tracking datapath and processing datapath [KRS03d]. The main function of the tracking datapath is to detect an incoming frame by searching for the periodic structure of the preamble symbols, and to estimate the carrier frequency offset. In our design, a wide range of frequency offsets can be estimated (±80ppm) using only two autocorrelators. The output of one of those is also used in the frame detection mechanism. This allows a significant core area reduction in comparison with other proposed solutions, as in [SCHW01]. Here, the range of the estimated frequency offsets is ±40ppm and three autocorrelators are used for the frame detection, but only two of them for the frequency offset estimation.

After autocorrelation, the signal is averaged with FIR filters and the square magnitude is calculated. In our design, frame detection is performed by a plateau detector, which has to detect a specific plateau shape in the incoming preamble symbols. When this plateau is detected, the carrier frequency offset can be estimated with a CORDIC processor operating in arctangent mode [MAH05]. As a result of this estimation, the frequency correction parameter ε is generated.

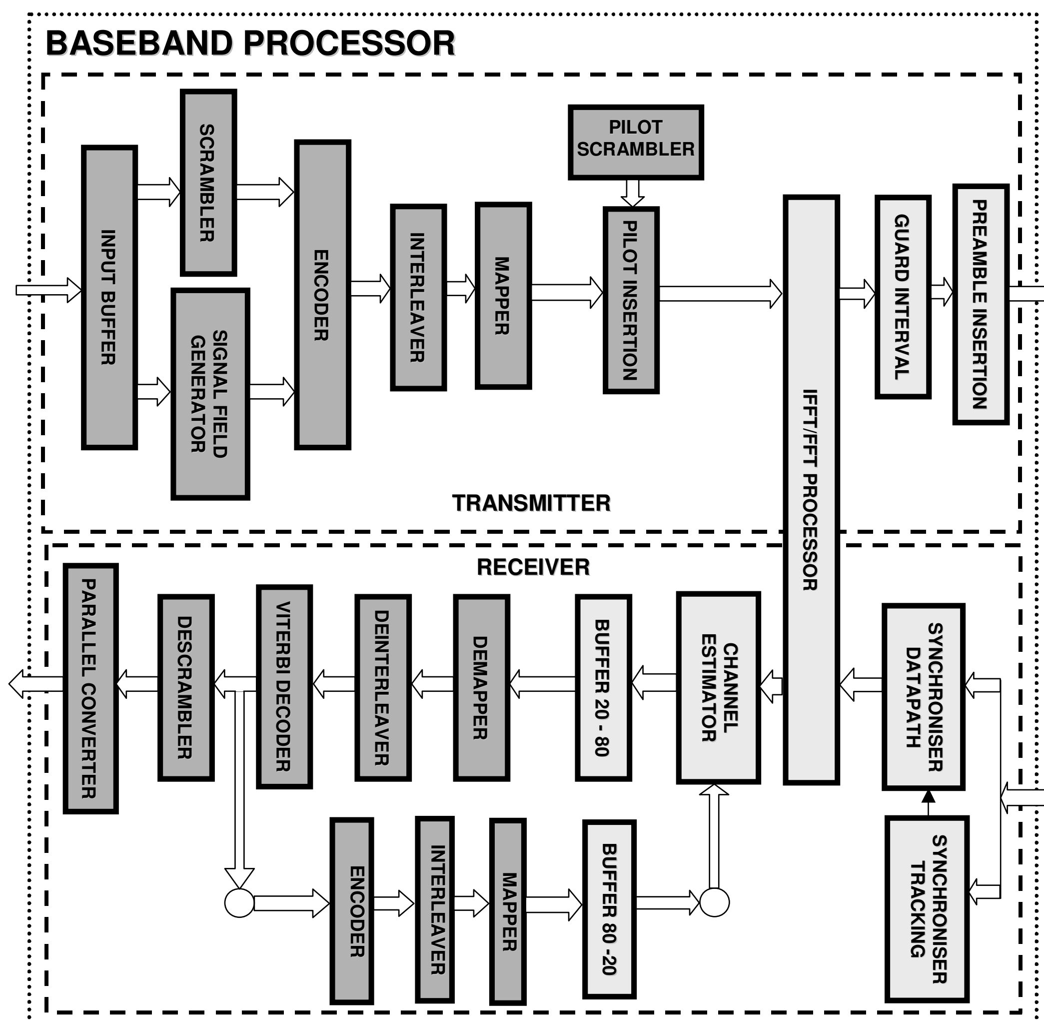

Figure 5.1. Baseband processor block diagram

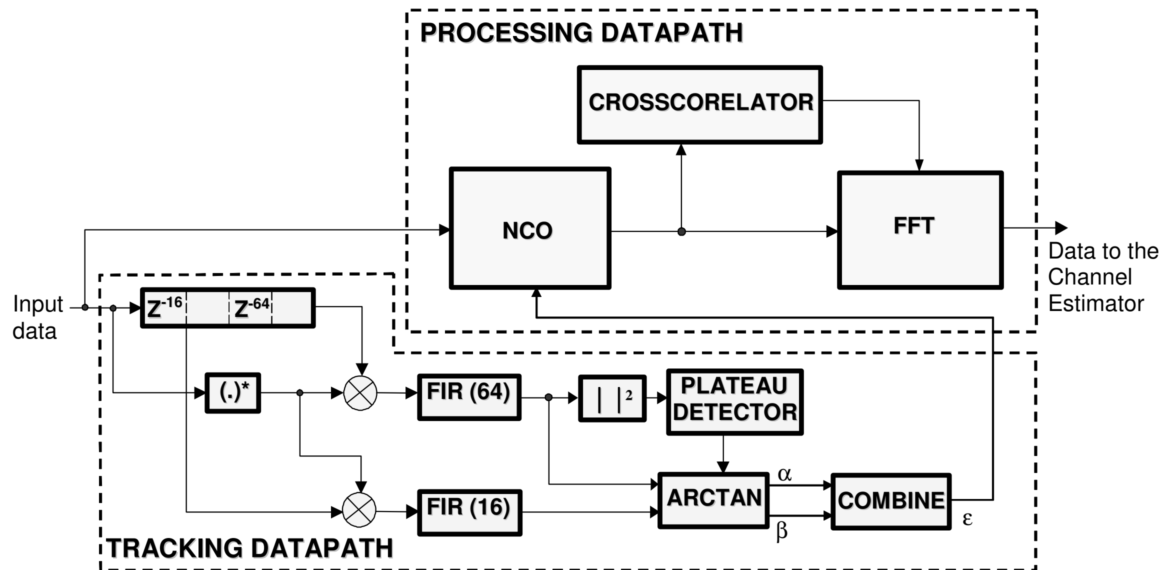

Figure 5.2. Structure of the synchroniser

The activity of the processing datapath starts after the frame is detected and the estimated value for frequency offset is available. This part of the synchroniser performs the carrier frequency error correction, estimates the symbol timing, and obtains the reference channel estimation. It consists of an NCO (Numerically Controlled Oscillator, in this case a CORDIC processor operating in rotational mode), an FFT processor and a simplified crosscorrelator based on XNOR gates [KRS03d].

Channel estimation and equalization is based on a decision-directed method [MIG96] with simplified residual phase estimation and correction. This type of channel estimation is based on a feedback loop. Therefore, our receiver performs additional convolutional encoding, interleaving and mapping (Figure 5.3). The interesting point in this concept is that it uses a division unit to correct the data samples (blocks equalizer and zero forcing in Figure 5.3. The estimator is designed in such a way that the samples of symbol *i* are used to calculate an estimation of the channel, which will be used to correct the symbol *i+D*, where *D* is the delay introduced by the feedback loop. Therefore, a delay buffer is needed to store the delayed symbols.

Other blocks in the receiving path are demapper, deinterleaver, descrambler, Viterbi decoder, and additional buffers. In order to simplify processing of data and reduce power consumption, the complete structure was divided into two clock domains. Computationally complex blocks without high data throughput were designed for 20 MHz and high data throughput demanding circuits were designed for 80 MHz clock frequency. With this clock domain separation, significant power saving was achieved.

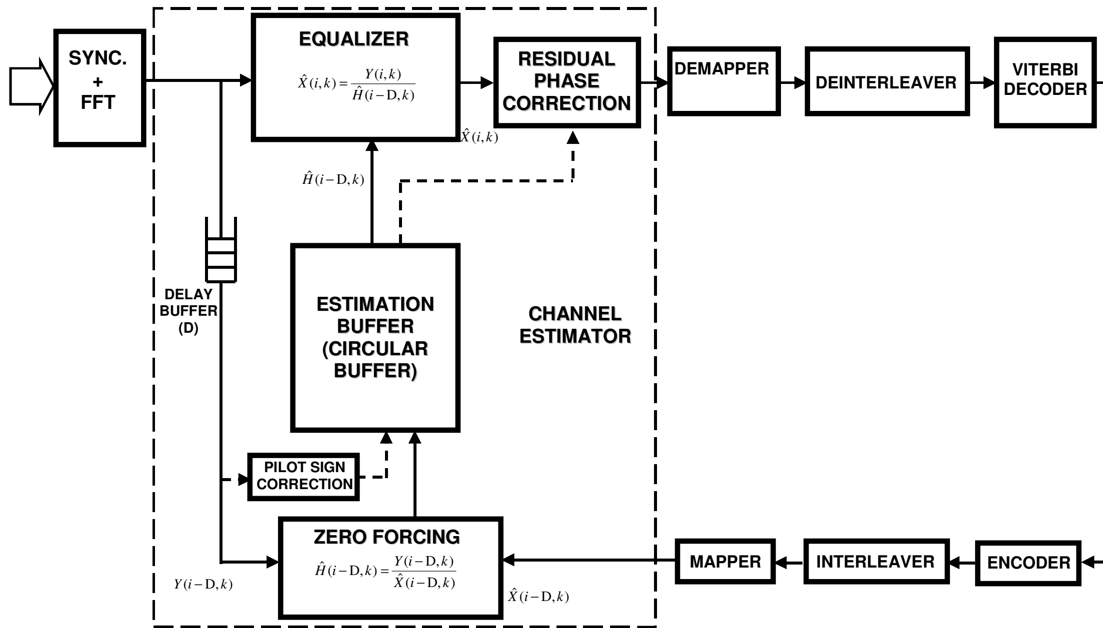

Figure 5.3. Structure of the channel estimator

The design of the synchronous baseband processor involves a number of integration challenges. For example, it was very complicated to generate the global clock tree due to two different clock domains (80 and 20 MHz, and an on-chip clock-divider) and an internally generated 10 MHz clock domain for the Viterbi trace-back logic. On top of this structure, clock-gating was applied as a power saving mechanism. Additionally, the complete design is flip-flop and not memory based, resulting in leaves in the design of approximately 36k flip-flops. That led to an even more complicated generation of the global clock tree. As a consequence, we had to solve enormous clock-skew problems. Therefore, due to CAD tool limitations and design complexity, the complete process was very prolonged and iterative.

As we plan to integrate a complete WLAN modem into a single chip, the integration problems will be even worse. The transformation of the standard synchronous design to a request–driven GALS architecture was pursued as a possible solution for the integration problems. The first task in order to perform GALSification of the baseband processor is block partitioning.

5.3 GALS Partitioning

It is always hard to define the strategy for partitioning a complex digital system into blocks. There are several options how this partitioning could be done, as we described in [KRS04a, KRS04b]. Additionally, the problem of GALS partitioning is elaborated in [MUT01].

It is plausible to take into account the natural division of processing stages of a system. In addition to that, an important issue is the power distribution in the processor. An optimal partitioning should support power saving in a similar way as clock gating.

For example, if we have one simple input block which has to process data very frequently, and the following very complex processing block which has to acquire data from time to time from the latter one, we could achieve large power savings if we partition those two blocks into separate GALS blocks. In this way, a complex block will be triggered only in the periods when there is some data to be processed.

Several additional factors are important for partitioning, such as complexity of the dataflow, clock frequency of locally synchronous modules, clock-tree complexity of the block, number of interconnects between the blocks, and gate count of the blocks. For example, it is always desirable to achieve acceptable trade-off between the gate size of a GALS block and the size of the wrapper. Additionally, a certain performance level must be achieved in order to provide the correct functioning of the baseband chip. Following this, during the process of block partitioning, we should take into account, whether the required performance level can be achieved with this particular block division or not.

The proposed request-driven operation of the GALS blocks induces a problem when we want to connect two blocks operating at different clock speed. Generally, low rate requests can drive high clock speed blocks without additional timing requirements. However, this configuration may lead to an unwanted performance drop. A more severe situation occurs when we connect high-speed request to the block synthesized with relaxed timing constraints. In that case, the timing requirements of the lower frequency may not be fulfilled. For example, in our baseband processor we have blocks that are operating at 80 MHz and at 20 MHz. In order to resolve such problems, we generated a special interconnection block that performs rate adaptation (dataflow transformation from 80 Msps to 20 Msps). For 80 Msps to 20 Msps rate adaptation, we could simply use a standard asynchronous FIFO as an interface. For dataflow transformation from 20 Msps to 80 Msps, it is best to use a separate GALS block as an interface. This block just buffers data in request-driven mode, and when an input burst is received, the interface GALS block will retransfer data with a higher clock rate.

The baseband chip is naturally divided into two different processing blocks – transmitter and receiver. In this implementation, there is no coupling between those two processing stages. Therefore, GALS partitioning is separately performed in the transmitter and in the receiver. This partitioning took into account natural boundaries between synchronous blocks, power saving issues, and need for uniformly distributed area between the GALS blocks. Furthermore, we decided to introduce the limited number of GALS blocks in order to simplify the GALSification and the hardware overhead. The result of our GALS partitioning is shown in Figure 5.4.

The principle structure of the GALS and the synchronous baseband processor is the same. The main change in the GALS implementation is a separation of IFFT and FFT processors in order to simplify the dataflow and consequently the implementation. In contrast, in the synchronous implementation, a single processor executes both operations. However, with reasonable effort it is possible to merge FFT and IFFT processors for the GALS implementation as well.

The baseband processor is a design with complex control structures. There are numerous control signals connecting different blocks. Therefore, we separated all those signals into two basic types: static and dynamic. An inter-domain signal is considered to be static when it doesn't change its value frequently and when this change is performed only when the receiving GALS block is inactive. Consequently, static signals can be transferred between the blocks without synchronisation. On the other hand, all other signals are considered as dynamic and they must be included in the data transfer via asynchronous wrappers.

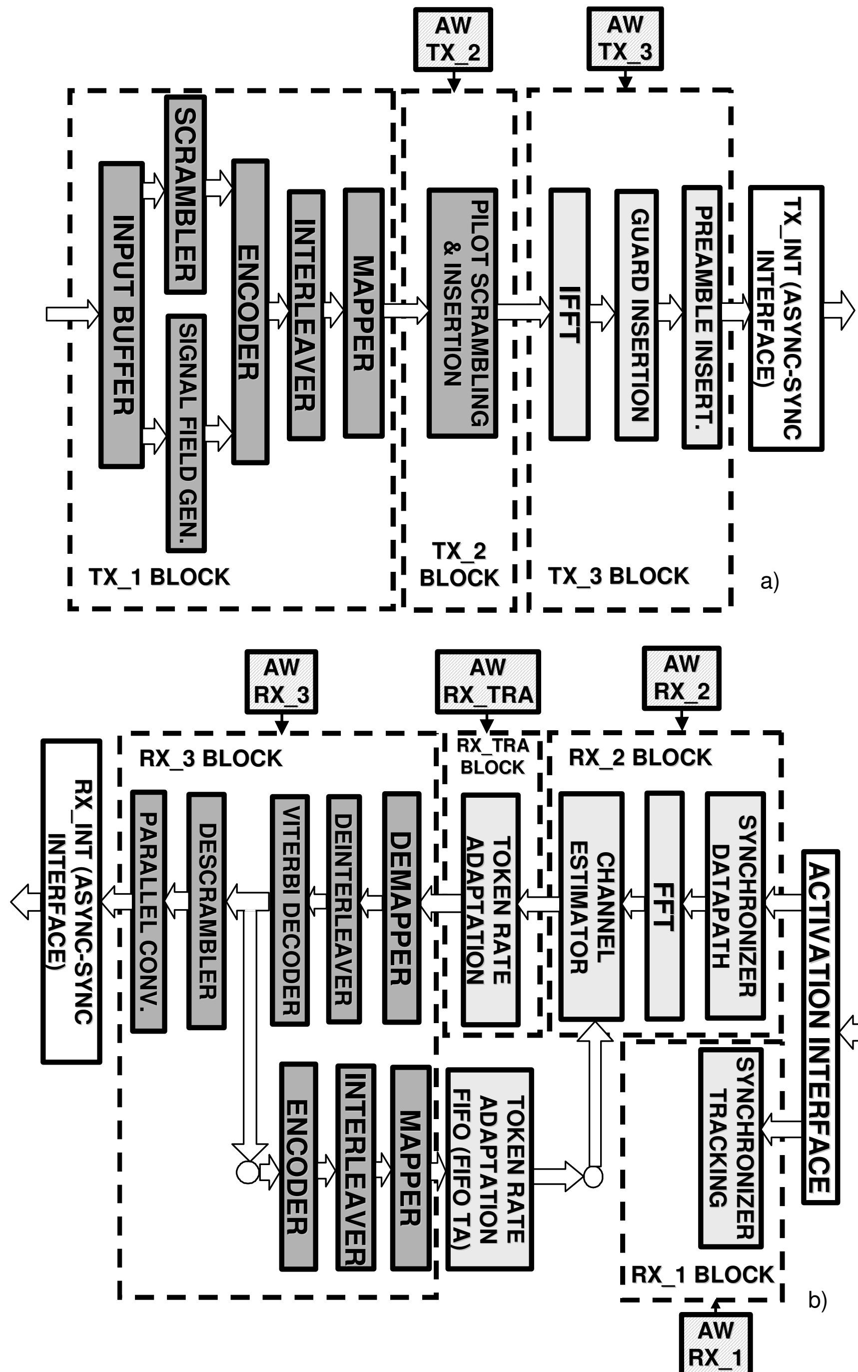

Figure 5.4. GALS partitioning in the transmitter (a) and receiver (b)

5.3.1 Transmitter Dataflow Organisation

Partitioning the transmitter is relatively easy. The transmitter is a straightforward datapath architecture, based on a token-flow design technique. Normally, a block division must take into account the functionality and complexity of the existing blocks. Consequently, the complete baseband transmitter is divided by natural functional boundaries into three modules. Block Tx_1 comprises 80 MHz components such as signal generator, scrambler, encoder, interleaver and mapper. The complexity of the block is about 17k gates (inverter equivalents). Block Tx_1 is not GALSified due to the complex off-chip interface protocol. The intermediate block Tx_2 (20k gates) has to perform pilot insertion and serial to parallel conversion. Additionally, this block performs token rate adaptation because it receives tokens at a frequency of 80 Msps and should generate tokens at 20 Msps for the last block. This is performed in the following way: When block Tx_2 is in request-driven mode, it will collect data at 80 Msps, but it will not generate any token to the neighbouring block Tx_3. When a complete data burst for one symbol is collected (48 data tokens), the local clock generator, set to 20 MHz, will initiate data transfer to block Tx_3. Block Tx_3 (153k gates) contains the very complex IFFT processor as well as guard interval insertion and preamble insertion. There is an additional problem in the communication between blocks Tx_2 and Tx_3. Generally, communication between those two blocks is performed in bursts of 8 data tokens and then block Tx_3 has to process the data for the next 72 cycles without any new incoming data. Therefore, block Tx_3 generates a special signal that grants transfer of a token burst (8 tokens) from the Tx_2 side. After that, this signal is released for 72 local cycles needed for processing.

5.3.2 Receiver Dataflow Organisation

The GALS receiver is significantly more complex. The gate count is more than double that of the transmitter. Moreover, the dataflow is much more complicated. The receiver can be partitioned into blocks operating at 20 Msps and into blocks with 80 Msps datarate. Additionally, the 20 Msps domain can be divided into two sub-blocks in order to limit the switching activity of the receiver. Consequently, there are three major blocks in the receiver. Block Rx_1 is a tracking synchroniser block with a hardware complexity of 80k gates and 20 Msps datarate. Block Rx_2 performs the datapath synchronisation and channel estimation. This block is quite complex (235k gates) and it operates with 20 Msps datarate. Block Rx_3 (168k gates) performs, in terms of timing and power, the most critical operations such as Viterbi decoding and deinterleaving. The block operates with 80 Msps datarate. One of the main challenges in the receiver is how to control the token ring between blocks Rx_2 and Rx_3, and how to achieve token rate adaptation from 20 Msps to 80 Msps (in forward data transfer Rx_2 $\rightarrow$ Rx_3), and from 80 Msps to 20 Msps (in backward data transfer Rx_3 $\rightarrow$ Rx_2). An additional problem is how to synchronise backward data transfer with forward data transfer. For solving those problems there are two possible solutions.

The first solution synchronises backward and forward dataflow. It appears plausible to define the backward dataflow as a slave (dependent) dataflow and the forward as a master (independent) dataflow. In this case, the slave dataflow can be established only in parallel to the master one. Token transformation in the baseband receiver according to this proposal is illustrated in Figure 5.5. In order to make this interface possible, one additional buffer stage, named Dataflow synchronisation, must be added to block Rx_3. This stage has to synchronise the backward data transfer at 80 Msps with the incoming forward data transfer to block Rx_3. Data can be transferred back to block Rx_2, only when the new data is supplied to block Rx_3. With such solution, the slave dataflow will not be an independent constituent of the asynchronous communication between the blocks, and the backward data transfer will be coupled with the forward data transfer. Similarly, block Rx_TRA is added in order to perform token rate adaptation (dataflow transformation from 20 Msps to 80 Msps) for the forward dataflow.

However, we decided to use the more general token ring solution and to retain independent backward dataflow. This can be achieved by adding two types of interface blocks: GALS block Rx_TRA and the asynchronous FIFO_TA, as shown in Figure 5.4 and with more details in Figure 5.6. This way, the backward dataflow is treated as an independent dataflow that creates asynchronous tokens. Rx_TRA will perform datarate transformation from 20 Msps to 80 Msps and FIFO_TA datarate transformation from 80 Msps to 20 Msps.

Module RX_TRA is implemented as a synchronous FIFO. It functions like any other GALS block. In request-driven mode, it receives 48 data samples per OFDM symbol at a datarate of 20 Msps. When the complete burst is received, the local oscillator resends the data to block Rx_3 at a datarate of 80 Msps. An equivalent block with the same structure and the same complexity exists also in the synchronous implementation of the baseband processor. Block FIFO_TA is implemented as a latch-based asynchronous FIFO. This block is supplied from Rx_3 block with 48 tokens per OFDM symbol at 80 Msps datarate. The data into Rx_2 comes from two sources, FIFO_TA and activation interface.

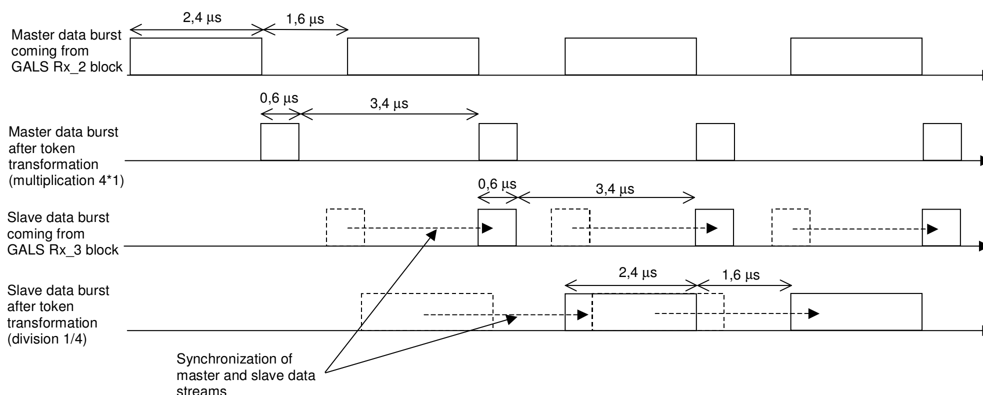

Figure 5.5. Possible solution of dataflow synchronisation

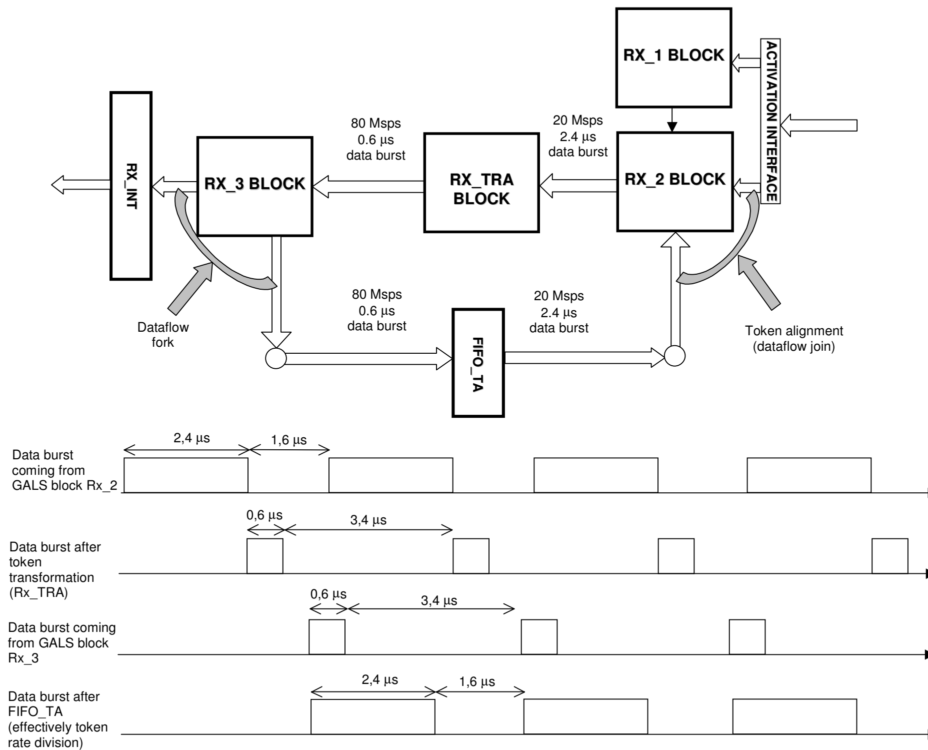

Figure 5.6. General solution for dataflow organisation

The FIFO output datarate is determined by the activation interface datarate. Since the activation interface operates at 20 Msps, the output datarate of the FIFO_TA will be the same. A similar circuit exists in the synchronous implementation of the baseband processor. However, the asynchronous FIFO is less complex than the synchronous counterpart.

A critical aspect in this solution is the control of the asynchronous token ring. Since the latency in this feedback loop is not known, joins and forks created in the system must be designed in such way that the operation of the system is not disrupted. Therefore, we have implemented the special join circuit for the alignment of tokens entering block Rx_2. FIFO_TA block performs the handshake very fast. The rate of the joined dataflow is determined by the activation interface rate (20Msps).

5.4 Power Saving Mechanisms

Using the proposed partitioning inside the Baseband processor it is possible to efficiently implement power saving mechanisms. In the following, we will analyze the activity of the baseband processor for a typical scenario of transmitting one frame, waiting for a response from the other station and receiving the acknowledgement frame.

As shown in Figure 5.7, after reset the baseband processor will produce no switching until the first data from outside arrives. The activity starts when the first data from AFE arrives or a frame transmission is initiated. After that, in the process of receiving, most of the time only block Rx_1 will be

active, trying to find the synchronisation pattern. This block is relatively small and the power consumption level will be acceptable. When synchronisation is reached, block Rx_1 becomes inactive and block Rx_2 will start its activity. This is achieved by switching the token multiplexer that is termed 'activation interface' in Figure 5.4 from 'tracking' to 'datapath' synchroniser. The activation scenario from Figure 5.7 did not cover the activity of the local clock of block Rx_2. In general, the local clocks of the blocks Rx_1 and Rx_2 operate for very few cycles after the activation mechanism disables the dataflow from the respective block. During those few cycles, the synchronous block should be reinitialized and prepared for the next data processing. Block Rx_3 will be activated only after a frame was detected and channel estimation was performed. Then the data is processed by the Viterbi decoder and Rx_3 becomes active. Blocks Rx_TRA and FIFO_TA are active only for a short period of time. During operation of the receiver, the transmitter is inactive.

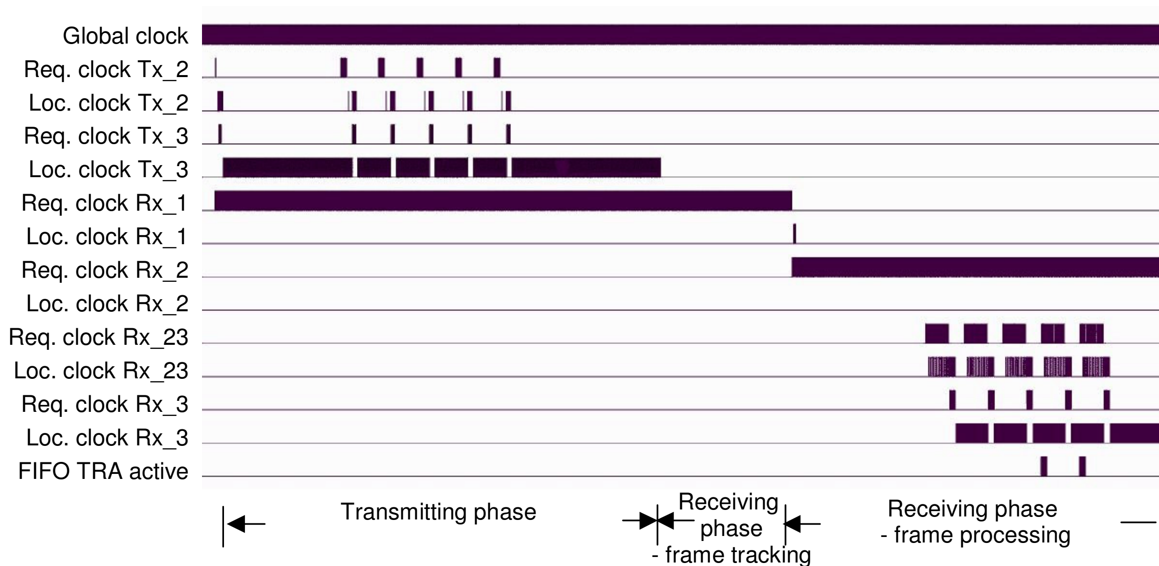

Figure 5.7. Activation scenario in the baseband processor

The amount of power saving achieved by this technique is comparable to clock gating with a similar strategy. An additional possibility to save power with respect to the standard synchronous solution is, that we are dealing with generally simpler local clock trees. For our synchronous version, block Rx_1 is active even when the system is in transmit mode. In order to allow a fair comparison we have retained this property for the GALS implementation.

The GALS introduction did not cause significant changes and challenges for the high-level system integration. The same asynchronous wrapper architecture was used for all blocks. All synchronous blocks remained unchanged. In principle, the interface block Rx_TRA already existed in the synchronous design, whereas other interfaces had to be designed additionally. However, for future GALS designs all blocks can be re-used.

5.5 Important Details – GALS Extensions

In order to control the operation of the GALS baseband processor it was necessary to develop several additional components. In general, to create a functional asynchronous token flow, it is needed to construct asynchronous building blocks such as join, merge, and mux. Moreover, it is needed to build blocks which provide communication of GALS wrappers with standard synchronous blocks. In general, these blocks must be able to absorb all tokens coming from the synchronous circuitry, and, on the other hand, to communicate all the time with asynchronous wrappers adhering to the handshake protocol.

In the following subsections, the structure and operation of those block will be described. Special attention will be paid to low power design of such components and possible optimizations of the suggested concepts.

5.5.1 Activation Interface for Blocks Rx_1 and Rx_2

The receiver activation scheme, introduced in the previous section, requires implementation of the interface between the blocks Rx_1 and Rx_2 with the synchronous decimator block. The purpose of this interface is to activate just one of the receiver blocks depending on the state of the receiver. In the tracking phase, block Rx_1 is activated and block Rx_2 is disabled. In the processing phase, block Rx_2 is active and the token flow toward block Rx_1 is disabled. The interconnect structure is shown in Figure 5.8.

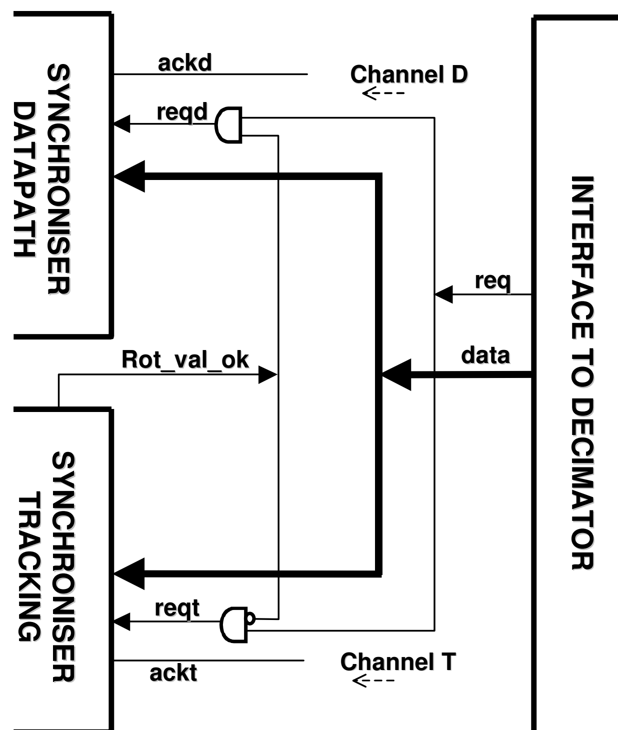

Figure 5.8. Synchroniser/decimator Interconnect

In general, signal *Rot_val_ok* allows a handshake on one of two possible handshake channels *Channel D* and *Channel T*. When *Rot_val_ok* is low, *Channel T* is active and the tracking part of the synchroniser (GALS block Rx_1) receives the data. When coarse synchronisation is reached, *Rot_val_ok* goes high and enables *Channel D* that triggers the datapath part of the Synchroniser (GALS block Rx_2). With this solution only one of two synchronising datapaths are active at one moment. The synchronous interface to the decimator cannot utilize the acknowledge signals of the asynchronous channels. Therefore, the asynchronous wrappers for blocks Rx_1 and Rx_2 must be able to finish a complete handshake within one clock cycle. In the GALS baseband system, this prerequisite is easy to fulfil. Signal *Rot_val_ok* changes its value only when both requests are low. This guarantees that the triggering of the request-driven blocks *Rx_1* and *Rx_2* is glitch free.

5.5.2 Specific Asynchronous Fork

From the baseband processor architecture in Figure 5.4 it is possible to observe the specific dataflow configuration of block Rx_3. This receiver stage is supplying two adjacent blocks with tokens. Therefore, it is necessary to implement a fork circuit for connection of block Rx_3 with FIFO rate adaptation circuitry and synchronous interface. Two channels that are generated (*Channel F* and *S*) with this fork can be activated only when there is an activation token for the particular channel. The FIFO rate adaptation circuitry consumes 48 data tokens per symbol (4 μs). The synchronous interface, on the other hand, receives the irregularly distributed tokens, indicating a valid received output byte. The two control signals, coming from block Rx_3, used to enable *Channel F* and *S* are *Start_m* and *Write_fifo*, respectively. The logical structure of the fork is given in Figure 5.9.

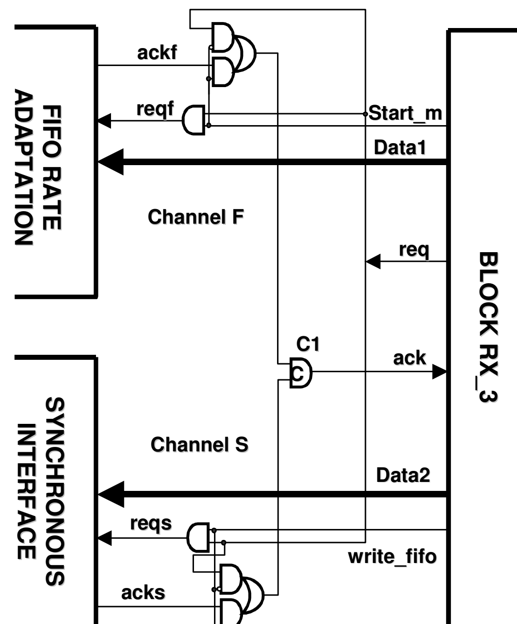

Figure 5.9. Circuit for connecting Rx_3, FIFO and synchronous interface

When only one of the channels is active (indicated by activation of either *Start_m* or *Write_fifo*), the simple handshake should be performed. When both are active (indicated by activation of both *Start_m* and *Write_fifo*), the asynchronous fork operation should be performed. Activation of both signals (*Start_m* and *Write_fifo)* is synchronised due to the fact that they are generated in the LS part of the block Rx_3. The specific fork circuitry contains just AND gates for generation of the respective request signals (*reqf* and *reqs*). They are activated if the request signal (*req*) and the respective activation signal (*Start_m* and *Write_fifo) are* simultaneously present. On the other hand, the acknowledge signal (*ack*) is generated from C-element *C1*, that joins acknowledge tokens from *Channels F* and *S*. Two AND-OR gates drive this C-element. When the channel is active (indicated with *Start_m* and *Write_fifo)* they should just pass the respective acknowledge signal (*ackf* or *acks*) to C-element. If the channel is not active, every request will immediately generate an acknowledge from the respective AND-OR structure. A precondition for glitch-free behaviour of the fork circuit is that signals *Start_m* and *write_fifo* must change their value before or at the request rising edge. However, this is easy to be fulfilled.

5.5.3 Token Alignment

In order to create the specific join between the synchronous interface and the FIFO token rate adaptation circuitry, a token alignment circuit was developed. From the synchronous circuit, tokens are coming every clock cycle (20 Msps), but from the FIFO, tokens are arriving in bursts and with a different rate (48 tokens per 80 tokens from the synchronous part). The joined tokens trigger block Rx_2. In order to not decrease the performance of the system, it is necessary to implement the appropriate join circuitry, as proposed in Figure 5.10.

When tokens are expected from both channels, the handshake circuitry will perform a join of requests *REQ* and *REQL*. When a token is expected only from the synchronous interface, the simple one channel handshake (*REQ-ACK*) will be performed. The circuitry consist of an asynchronous controller (given on Figure 5.11) which, depending on control signal *EN*, should join two incoming requests or just perform a simple handshake.

Signal *EN* is generated from additional circuitry that incorporates one counter that should signal when tokens on both channels are expected and when not. The counter counts until 48 and it will be triggered with every incoming token from FIFO. Before this counter reaches 48, the full asynchronous join is performed. After that, until a new *REQL* comes, only a simple handshake will be performed. After that, the handshake circuitry will start again with the join operation. Implementation of the ACONT controller is quite simple and it is defined by the following logic equations generated from 3D tool [YUN99a]:

$$REQA = REQ^* \cdot REQL^* + REQ^* \cdot REQA + REQL^* \cdot REQA + REQ^* \cdot \overline{PASS}$$

$$\mathrm{PASS} = \overline{\mathrm{EN}}$$

Additional circuitry (Mutex element, RS-flip-flops, AND gates) shown in Figure 5.10 is required in order to guarantee burst-mode behaviour of the AFSM controller. Mainly, those blocks should prevent the appearance of signal REQL in the middle of a handshake on the REQ-ACK channel.

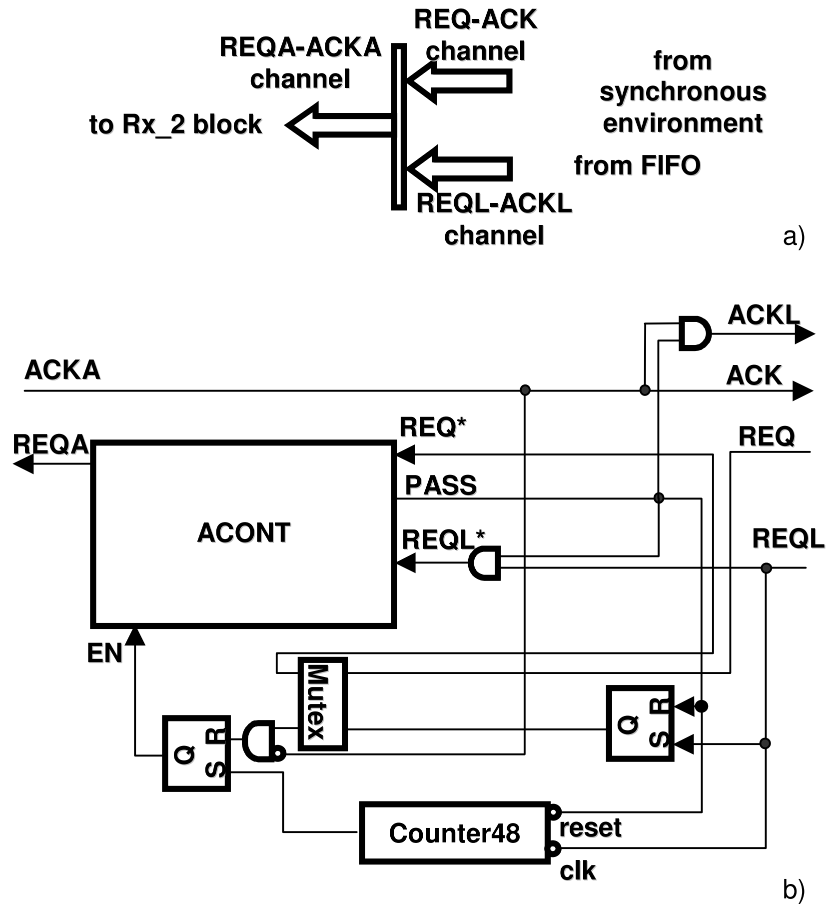

Figure 5.10. Join channel block diagram (a) and schematic (b)

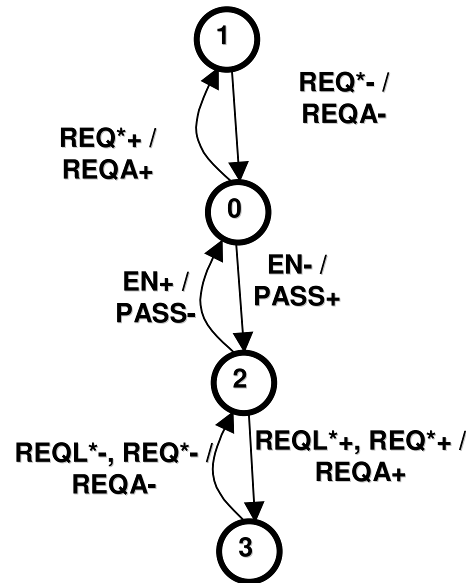

Figure 5.11. AFSM for ACONT controller

5.5.4 Rate Adaptation

At two interfaces in the receiver it is necessary to connect GALS blocks operating at different clock speed. In both cases it is a connection between block Rx_2 (20 Msps block) and block Rx_3 (80 Msps block). In one case, tokens are generated from Rx_2, and, in the other case, from Rx_3. Because our GALS blocks are request-driven, it is necessary for the correct operation to have tokens coming at the expected clock speed. In order to solve this issue, we have developed two decoupling circuits RX_TRA and FIFO_TA as explained in section 5.3.1.

5.5.5 Token Synchronisation in the Transmitter

The token-flow of the baseband transmitter has one additional requirement, which cannot be covered with the functional behaviour of the already proposed simple asynchronous wrappers. In the transmitter, the last block in the dataflow - Tx_3, starts to send the preamble sequence immediately after the transmit command. In the meantime, the other transmitter blocks process the signal and data field information. After transmission of the preamble block, Tx_3 starts to send the signal field information. However, data transfer between Tx_2 and Tx_3 must be synchronised in order to acquire data in block Tx_3 in a particular clock cycle. For that reason, the data transfer in block Tx_2 is prepared well before Tx_3 need the data, but it is not enabled. If the data is ready but the transfer is not granted, further clocking of block Tx_2 is disabled with pausing of the local oscillator. When the data is really needed, the clock oscillator of block Tx_2 will be released and the data transfer is performed. In hardware, this is performed in a very simple manner. One AND gate disables incoming requests between blocks Tx_2 and Tx_3. This gate will be released only when tokens from block Tx_2 are expected.

5.6 Synchronous-Asynchronous Interfaces

The baseband processor is intended for use in a standard synchronous environment. In transmit mode, it has to generate data tokens at every clock cycle (20 Msps) to the DAC (Digital to Analog Converters). In receive mode, it must be able to absorb data tokens from the ADC (Analog to Digital Converter) at every clock cycle (20 Msps). These specific requirements are very hard for a non-deterministic GALS system. In order to perform those tasks, specific interface circuits are proposed. The proposed blocks are of two types: the first for supporting the communication between a synchronous producer and an asynchronous consumer and the second for communication between an asynchronous producer and a synchronous consumer.

5.6.1 Synchronous to Asynchronous Communication

To connect a synchronous token producer with a request-driven GALS block, a relatively simple pipeline circuit can be used (Figure 5.12). Pipeline stages are used to store tokens coming from the producer, if the GALS block cannot immediately consume them. A tunable delay element is inserted on the acknowledge line to the GALS block in order to preserve the minimum cycle time for this GALS block (for request driven operation of GALS). However, this solution may cause significant delays in the GALS block. On the other hand, if the delay is not long enough it is possible to trigger the GALS block with a higher frequency than its locally synchronous pipeline can operate at.

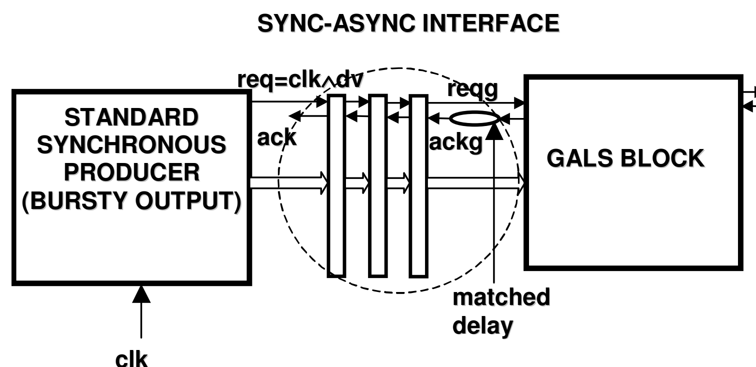

Figure 5.12. Synchronous to asynchronous connection

5.6.2 Synchronous to Asynchronous Communication with a Continuous Synchronous Data Stream

When data from the synchronous producer is continuously generated, we propose that the first GALS block in the datapath is driven directly from the synchronous block (as depicted in Figure 5.13). In order to avoid the performance drop in the first GALS block an interface described previously is attached. The task of the interface is to decouple GALS block 1 and GALS block 2, and to collect some possibly delayed tokens.

However, even this solution may suffer from similar problems as the one from the subsection 5.6.1. Therefore, in the GALS baseband processor we connected the request line of the first GALS block in the dataflow directly to the synchronous clock. In this case, the acknowledge line is not used. Consequently, we must rely on the fact that a token consumer is able to absorb every token for the time of one synchronous clock cycle. For our GALS processor it was easy to guarantee such behaviour. This condition is fulfilled with the insertion of decoupling circuitry between blocks RX_TRA or Tx_2 block. The interface GALS block, when operating in request-driven mode, is completely decoupled from the subsequent GALS blocks. Therefore, it can easily respond to the incoming request without additional delay due to the handshake with the following GALS block.

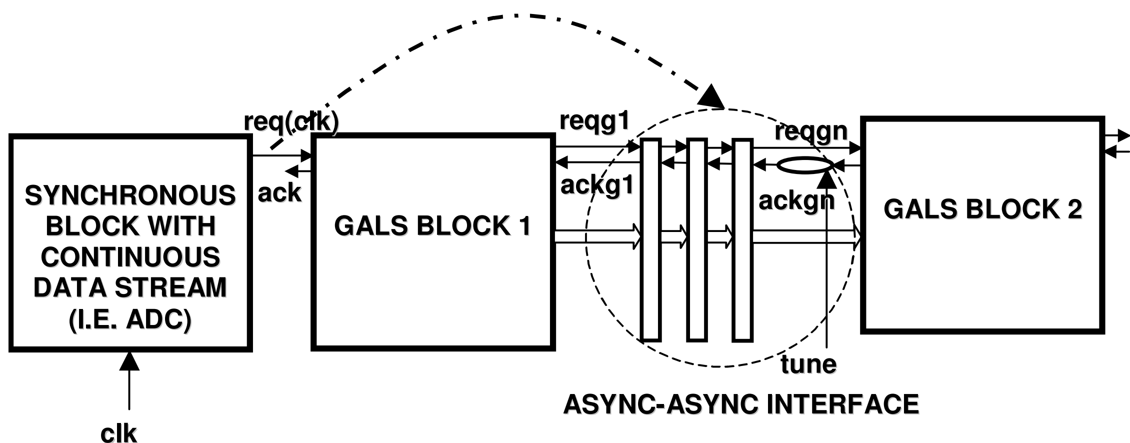

Figure 5.13. Connecting synchronous and asynchronous blocks with intensive data transfer activity

5.6.3 Asynchronous to Synchronous Communication

The interface between an asynchronous token producer and a synchronous consumer is a simpler problem than the previous one. The reason is the request-driven nature of the GALS block, which does not severely affect this type of communication. To connect an asynchronous producer with a synchronous consumer we used well known pipeline synchronisation [SEIZ94] (Figure 5.14). This solution is used for the connection of GALS baseband blocks with the DAC. It is also applied to the receiver output stage when connecting block Rx_3 to the synchronous interface. Several MUTEX elements are used to synchronise the incoming request with the synchronous clock. With this solution, synchronisation problems are hidden in the hardware of FIFO stages. This way, we can safely connect an asynchronous producer with a synchronous consumer. Addition of more pipeline stages leads to an increase of the system robustness. However, an increased number of stages also leads to a significant increase in power consumption and area.

Using pipeline synchronisation, the dataflow between an asynchronous producer and a synchronous consumer is reliable and fast. However, in order to guarantee robust operation of this

interface, the structure of the implemented pipeline synchroniser is very complex and power hungry. Therefore, one of our future tasks is simplification of the applied concept.

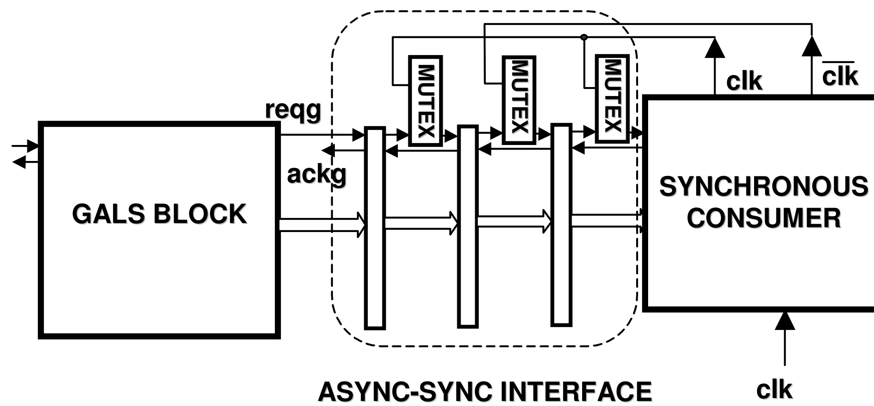

Figure 5.14. Asynchronous to synchronous interface

Chapter 6

Design for Testability in GALS Systems

6.1 Introduction

Design for testability has always been a bottleneck for asynchronous systems and one of the most important reasons why asynchronous techniques have not gained adequate industrial popularity yet. In recent times, Globally Asynchronous Locally Synchronous (GALS) techniques are proposed as an effective way of complex digital system integration. GALS wrappers have to facilitate the integration of large synchronous blocks into complex digital systems. In order to achieve wide commercial use of the GALS technique, it is needed to provide an adequate way of testing it.

Our aim is to test GALS chips with standard hardware testers. However, usually such equipment is oriented towards pure synchronous systems. On the other hand, most GALS systems are externally synchronous, but they are not timing deterministic. Therefore, a pure functional test of an asynchronous digital system with standard synchronous test equipment is not easy. A simple and efficient alternative is needed.

The choice of a suitable test technique is a complex problem. Generally, the design for testability (DFT) of synchronous digital systems is based on the scan chain approach. There are similar techniques also in the asynchronous side [BER02]. On the other hand, there are claims that a functional test is adequately efficient for asynchronous circuits. Recently, a GALS test technique based on a functional test is proposed in [GUR02]. Built-In Self-Test (BIST) could be a promising approach for GALS systems and it is already successfully used in the asynchronous world [ALV98]. In several papers BIST is reported as a possible technique for testing GALS systems [HEA03, DAM02, BLA02]. However, in none of them particular test strategies and configurations are proposed.

In the following sections, different techniques for testing GALS systems will be evaluated. This was reported in [KRS05b]. A test structure based on the BIST architecture will be described. Based on the proposed strategy and circuits, BIST will be applied for the testing of our GALS baseband processor compliant to the IEEE 802.11a standard.

6.2 Test Techniques for GALS Systems

To propose an optimal test strategy for GALS systems, it is necessary to define test requirements. For a GALS application, the test must be clearly separated from the functional part in order that testing cannot affect the proper operation of the system. Additionally, the test should be easily initialized and controlled. The test circuitry should provide easy observability and controllability of all interesting points in the system. On the other hand, it is desirable to provide hierarchical testing, and testing at-speed of operation. Finally, all tests should show robustness towards the asynchronous behaviour of the wrapper and the highest possible fault coverage.

There are different possibilities how a test of GALS systems can be performed. In Table 6.1 a comparison of several testing techniques applicable to GALS is illustrated.

In general, BIST obviously separates the test circuitry from the functional part. On the other hand, in the case of a scan-based approach, the test structure is directly coupled with the functional components. BIST is also very useful allowing simplicity of the initialization and control procedure. For the scan- based approaches, test generation and management is not effortless but there are many tools which support the designer. In the case of the functional test, test vector generation is a critical point. There is a problem to access deeper pipelines in the system with functional tests. Furthermore, the number of vectors is usually huge and the cost of testing unacceptable. It is much easier to organise hierarchical testing based on BIST than on scan methods. Regarding observability and controllability it is clear that scan testing offers the best results. Functional testing has the problem that many errors can be masked inside the design and are not observable on the I/O pins. In general, the functional test and BIST test suffer from lower test coverage in comparison with scan-based approaches. However, compared to functional tests, BIST is much cheaper, regarding time for testing, for a certain level of test coverage. Finally, at-speed testing is not possible with scan methods. The only possibility for this type of testing is to use either BIST or functional testing.

Our main motivation for DFT introduction within the GALS baseband processor is to have a possibility to run very complex functional tests internally (without providing external test vectors). We use a hardware tester that is strictly cycle based and cannot react to asynchronous output signals of the circuit. The GALS arbitration processes preclude cycle level determinism. PVT variations further contribute to timing non-determinism. BIST significantly reduces the effort for generating a test program and enables us to use a synchronous tester. Furthermore, for datapath architectures (even for the pure synchronous one), it makes no sense to implement a full scan chain. Several reasons

support that: the number of scan elements is huge and the time for testing will be unacceptable, functional testing could verify correct system operation with high confidence, and scan-insertion will diminish the performance of the system. Additionally, the scan approach does not cover dynamic faults and race states. It requires a global test clock tree for the scan-cells that we wanted to avoid with GALS. However, applying a functional test for GALS will require expensive testers and increase the cost of testing. Therefore, we have decided to implement a BIST around the asynchronous wrappers for the testing of our GALS systems.

Table 6.1. Comparison of test approaches for GALS systems

Type of test / Requirements	Scan test	Functional test	BIST
Test circuitry separated from design	☆	☆☆☆	☆☆☆☆☆
Simple initialization & control	☆☆☆	☆	☆☆☆☆
Easy observability & controllability	☆☆☆☆☆	☆☆☆	☆☆☆☆
Mixed granularity	☆☆	☆	☆☆☆☆☆
Robustness, coverage	☆☆☆☆☆	☆	☆☆
At-speed testing	☆	☆☆☆☆☆	☆☆☆☆☆
Σ	☆☆☆	☆☆	☆☆☆☆

6.3 Proposed BIST Architecture

The architecture of one asynchronous channel is given in Figure 6.1. In order to test the operation of the channel, several test components are placed. A test pattern generator (TPG) is positioned at the output of the asynchronous GALS wrapper and the test data evaluator (TDE) is placed at the input of the asynchronous wrapper. In principle, any placement of TPG and TDE around the wrapper is conceivable. We decided to use the depicted configuration because the priority is testing of the asynchronous channel. However, the testing of the handshake signals is possible only implicitly by compacting the valid data transferred via the asynchronous channel. We will use one of the control signals (*REQ* or *ACK*) to trigger the respective TDE. With an additional TDE, as shown in Figure 6.1, it is possible to evaluate the operation of the locally synchronous part, too. The TDEs and TPGs must be designed in such way that they operate completely asynchronously and follow the handshake protocol.

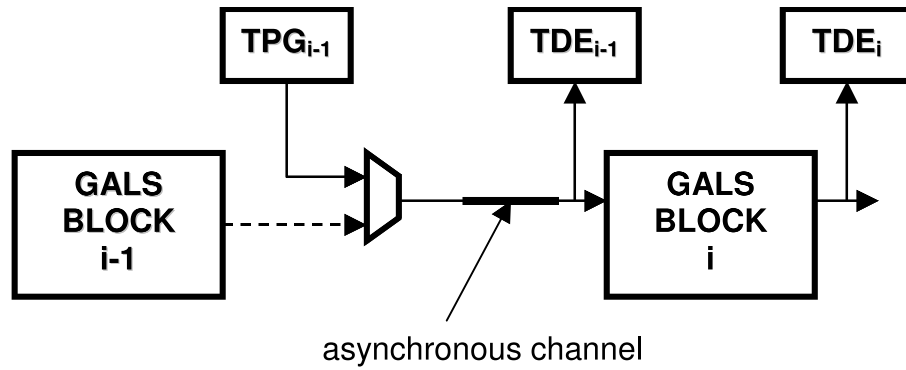

Figure 6.1. BIST configuration

The general architecture of the BIST building blocks is shown in Figure 6.2. Additionally, interfacing of several BIST blocks is illustrated in Figure 6.3.

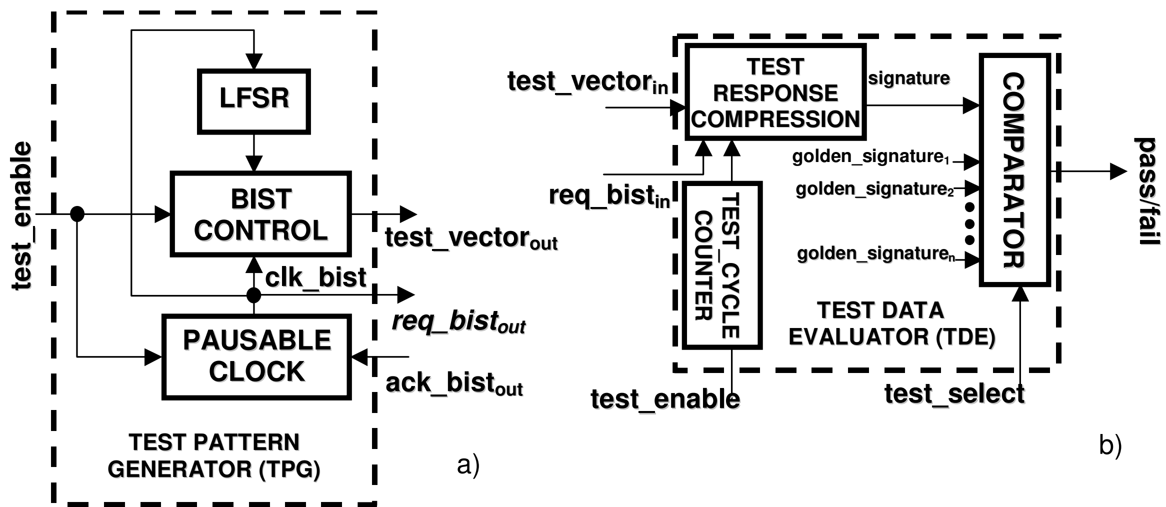

Figure 6.2. BIST components

A test pattern generator (TPG), given in Figure 6.2a, has to generate patterns that are suited to perform the testing of internal operations of an asynchronous wrapper and the operation of a locally synchronous module. The TPG consists of a Linear Feedback Shift Register (LFSR), a BIST controller and a pausable clock generator. The LFSR has to generate pseudorandom patterns. The BIST controller has to control the LFSR and to generate test vectors based on the LFSR output and, optionally, on some predefined test patterns. This predefined data structure is usually connected with the functional requirements of the specific GALS block. The purpose of this additional logic is to enable non-random test vectors that open the LS datapaths for particular operations.

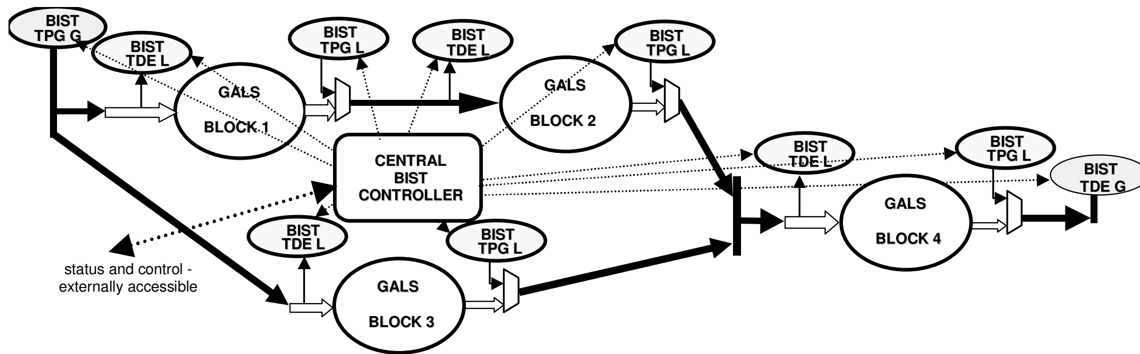

Figure 6.3. Global BIST configuration

For example, sometimes it is necessary to combine the pseudorandom data pattern with certain defined sequences in order to activate deeper datapath structures of a complex GALS system. Finally, a clock generator is needed in the TPGs, because our GALS blocks are request-driven. We need to trigger the synchronous part of the TPG in parallel with the token generation for the GALS block. However, in general it is not always easy to use an ordinary external clock source for this purpose because this clock cannot be stretched if the acknowledge on the GALS line does not arrive before the next rising edge of the clock. Therefore, we need a pausable clock, based on a tunable ring oscillator to drive the request and BIST clock. For better component utilization, it is proposed to use just one ring oscillator for all local TPGs, because only one of them is needed for a particular test procedure.

The test data evaluator (TDE) should check the output test data and indicate the result of the testing. The TDE consists of a test response compression circuit and a comparator, as shown in Figure 6.2b. The test response compression block is based on signature analysis and, accordingly, incorporates one LFSR in its structure.

The TDE is triggered with the respective handshake signal. It records the changes on the data lines only when a valid control token in present. Because of that, with this BIST approach, we can tolerate timing nondeterminism. For comparison, with the actual signature it uses one of n precomputed golden signatures. Which golden signature will be used depends on the performed test. A golden signature could be chosen via the test_select control signal, defined by the global BIST controller. The number of test cycles is defined in the test cycle counter. This circuit is counting the number of accepted valid test vectors and when this number is equal to the depth of the test vector set, the test response compression block will be disabled and a pass/fail signal is generated.

Two different types of BIST have been implemented: global and local. Such hierarchical BIST approach is not novel [HEAL04], but in the asynchronous and GALS area was not used until now.

The global BIST is initiated at the circuit boundaries and, therefore, is performed fully synchronously. Consequently it is based on the general well-known BIST strategy. Additionally, during the global test, it is possible to enable the local test compaction, placed between the GALS blocks, in

order to increase the number of check points and to achieve a better fault coverage. When the global test is finished, our test controller can activate the local tests in order to further evaluate various components of the GALS system. During every local test one local test pattern generator is activated and the results are stored in one or more test data evaluators. With such a strategy, testing may give broad and profound results about the correctness of the circuit operation and the operation of specific GALS blocks. With an increased number of test points and with local testing it is possible to isolate faulty components and possibly perform diagnostics of the fault.

For global testing of the system, a global test pattern generator (TPG G) is used. This generator may function completely synchronously. The generated data should propagate through the system and a global test data evaluator (TDE G) should process output data and compare it with the expected value. Additionally, there are several local TPGs and TDEs (TGG L and TDE L) that are driven completely asynchronously.

In order to control the BIST registers and to collect and process the results of test, one central BIST controller (CBC) is proposed. This CBC includes a pausable test clock generator in its structure. It is proposed that the BIST controller has its own autonomous reset signal in order to achieve decoupling from the system reset. In this case the CBC can be programmed to prepare a particular test during system reset when the GALS circuits are stable. Then the BIST can start immediately after the deactivation of the system reset. After some period, which should be sufficient for performing the test, the test status data is delivered.

In general, the purpose of the CBC is to allow a simple off-chip communication mechanism that can be used for test parameter setting and test result reading. From the signals in Figure 6.4, *Test_reset* is used for the reset of the testing circuitry, *Test_on/off* is used for the activation of the testing operation and *Test_select* should select which specific vector set is activated. Signal *Test_ok* indicates the success of the test, when all test vectors are processed. On the other hand, the CBC controls the execution of the particular test, based on the external setting. For that reason, *TPGi_en* and *TDEi_en* enable the respective TPGs and TDEs, *TDEi_golden* selects the golden signature value for a respective TDE, and $pass/fail_i$ should indicate the result of the testing. In addition to that, the CBC enables the operation of the pausable test clock generator (PTCG) when the setting of all parameters of the current vector set is finished and all TPGs and TDEs are ready to perform a particular test. Moreover, it is possible to adapt the CBC such that it supports some of the existing test standards as EJTAG etc. This activity of all BIST I/O pins is completely timing deterministic and synchronised with the external clock. Hence, a hardware tester can be used for automatic testing.

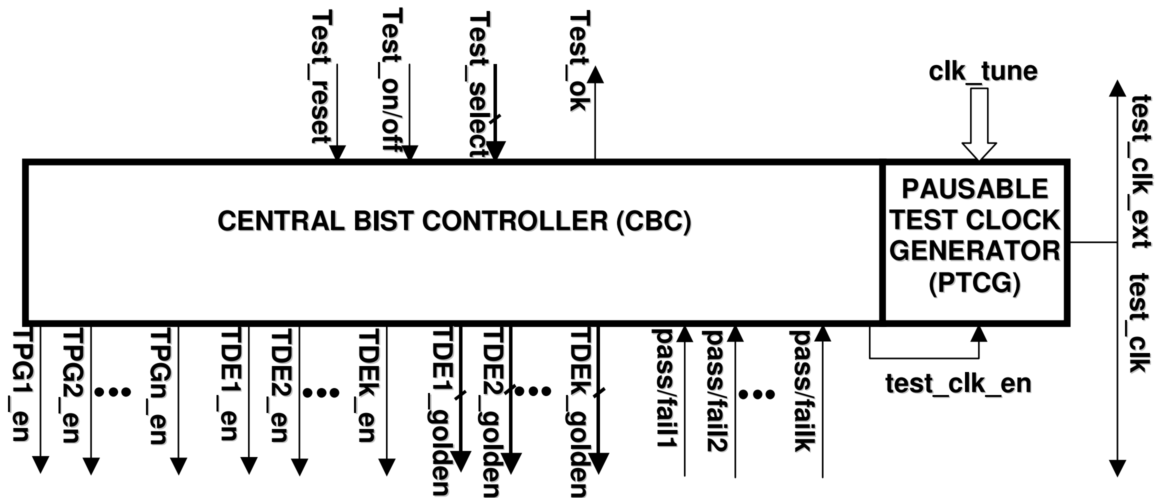

Figure 6.4. Central BIST Controller (CBC) configuration

6.4 Implementation of the BIST in the Baseband Processor

The described BIST structures are applied in the GALS baseband processor. A major task of the BIST implementation was to verify the functionality of the asynchronous control flow. A secondary task was to test, with reasonable coverage, the operation of the locally synchronous module. In order to fulfil those tasks, eleven different Test Data Evaluators (TDE) were incorporated in the system. Five of them are situated in the transmitter and the others are part of the receiver. TDEs are inserted at critical points of the GALS system. The most important positions are between the GALS blocks, where inter-block communication can be observed. The implemented TDEs are of different width (in the range from 10 to 256 bits) but with the same LFSR structure.

In order to more deeply investigate the correctness of the GALS baseband processor operation, five different Test Pattern Generators (TPG) and, accordingly, five different types of tests are implemented. The duration of all tests is around 300 μs, when the clock sources are set to support the real speed of system operation. One Central BIST Controller (CBC), specially designed for this GALS baseband implementation, performs the control of the complete test procedure. The CBC is designed according to the proposal presented in the previous section. When some BIST test is finished, the state of line *Test_OK* should indicate the success of the test. This value depends on the results of the global BIST test. Additionally, before this value becomes stable, we are using the same line to indicate the correctness at local test points. This information is indicated with the presence of pulses on the *Test_OK* line. This representation is shown in Figure 6.5. When the BIST test is performed, we will get at least one pulse indicating completion of the test procedure. The other pulses will indicate pass/fail of test result at the particular checkpoint. The final steady value will indicate the result of the compaction at the last TDE (TDE10) in the test chain. The complete structure of our GALS baseband processor with positions of all TPGs and TDEs is given in Figure 6.6.

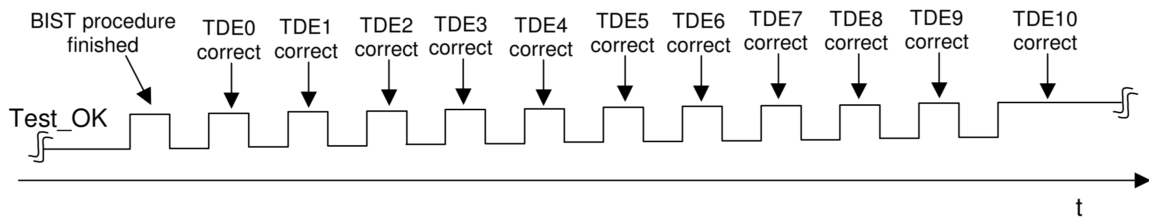

Figure 6.5. Representation of the test results

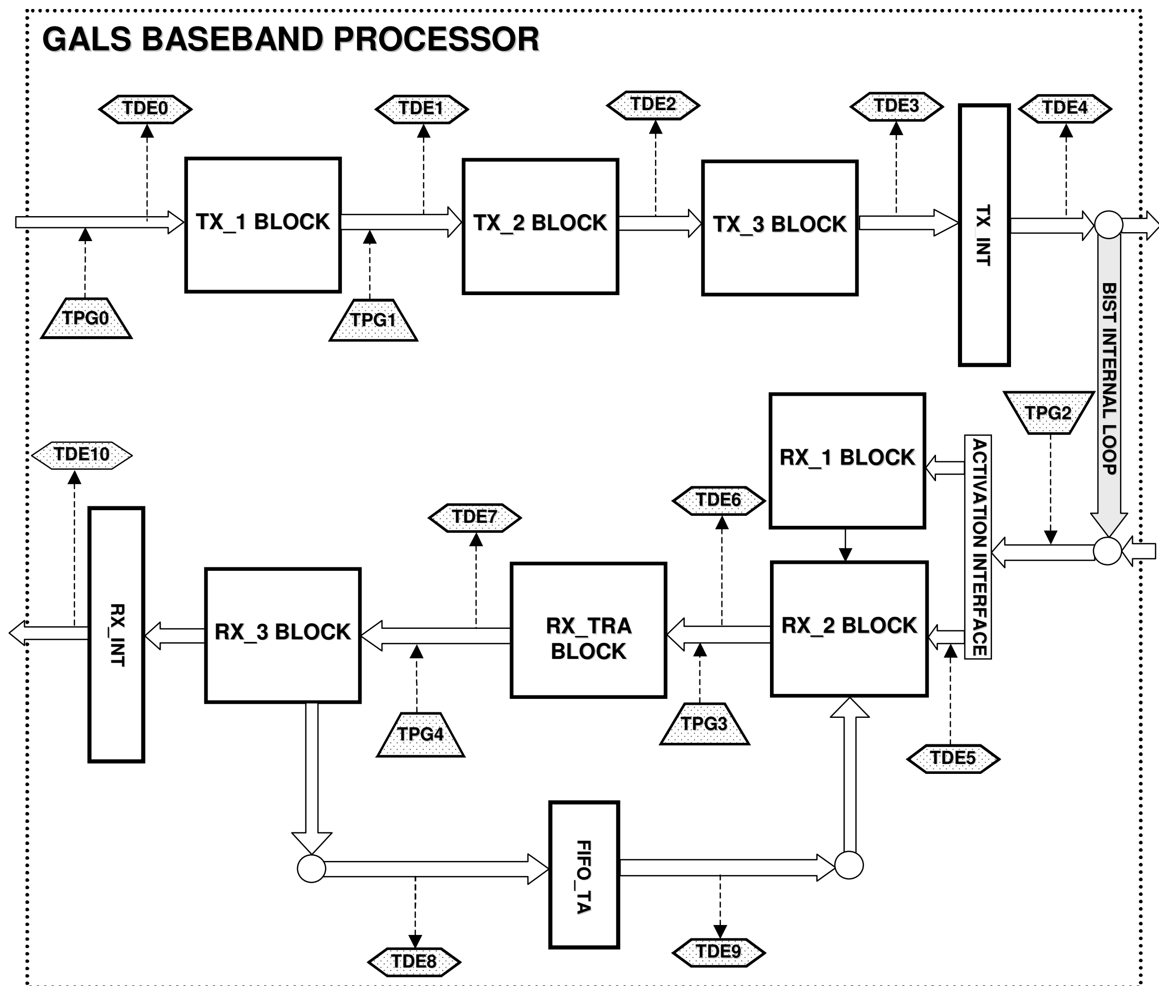

Figure 6.6. GALS baseband processor with BIST test points

With the proposed test structure we have the ability to run hierarchical tests. We have implemented a global test of the complete transceiver structure. This test is initiated from TPG0 in Figure 6.7. TPG0 has an additional control mechanism that performs initialization of the transmitter and sets the number of transmitted data words to 2kB. After that, random data is generated (2kB) and fed into the transmitter. Consequently, the transmitter sends an IEEE 802.11a compliant frame. During the global BIST test, an internal loop in the baseband chip is activated that directly feeds the data coming from the transmitter back into the receiver. With this solution, we can test both transmitter and receiver.

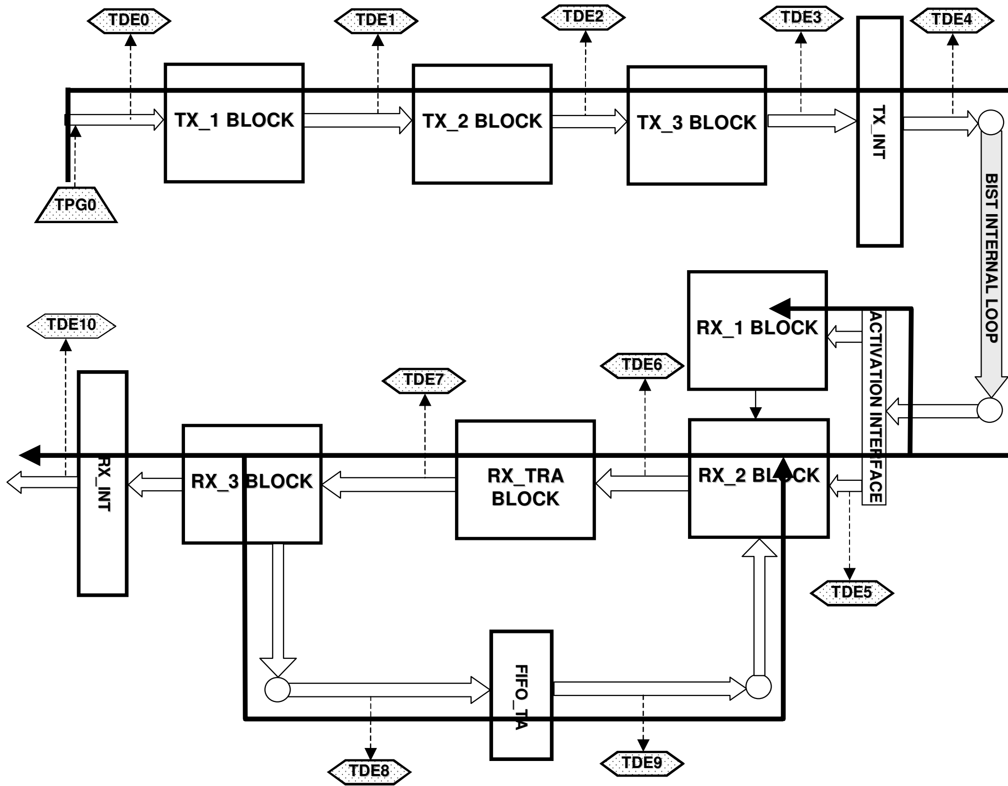

Figure 6.7. Global test of the baseband processor

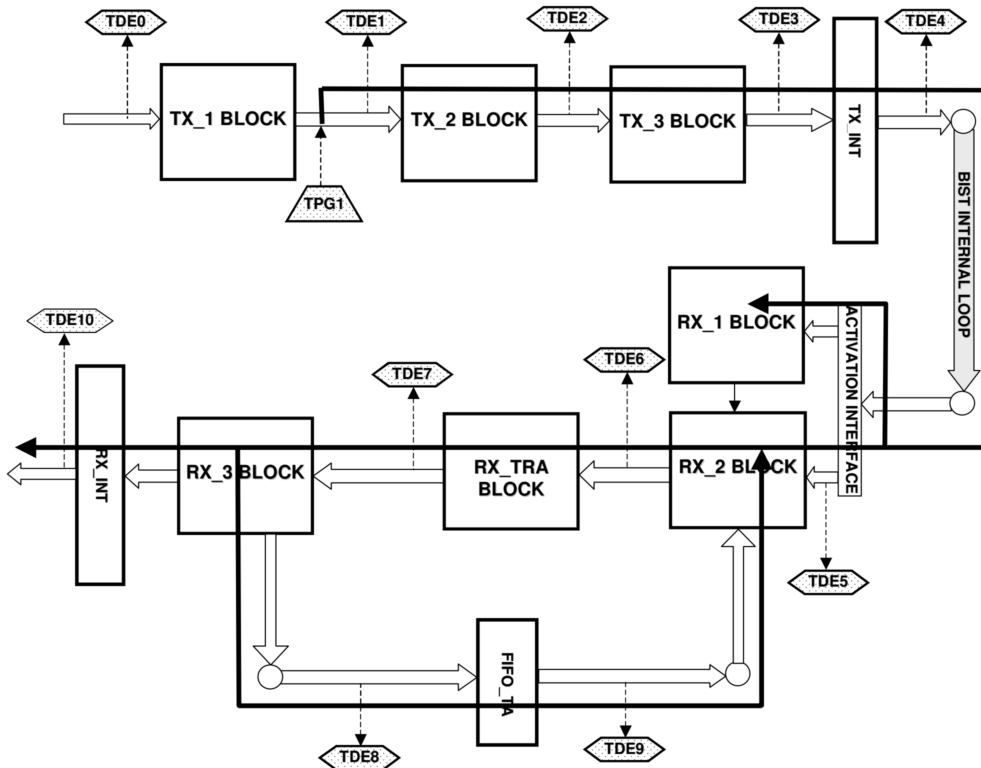

Figure 6.8. Reduced global test of the baseband processor

During the test, the complete datapath is in operation and all TDE blocks are activated. Hence, this test will yield information about the complete system. In most cases it will be sufficient to successfully perform just this global test in order to verify that the GALS system is correct.

In the case that some problems have occurred during this first test, the other local tests can be applied to detect and isolate possible problems. The local tests will increase the test coverage and strengthen the quality of the results. Those module tests are focused on parts of the dataflow or on particular GALS blocks. A second test (Fig 6.8) is initiated from TPG1 which is mainly focused on the transmitter operation. The receiver is also included into this testing chain. There are ten compactors that check the generated data during this test (TDE1 – TDE10). It will provide more information about the reliability of the data transfer between synchronous block Tx_1 and GALS block Tx_2.

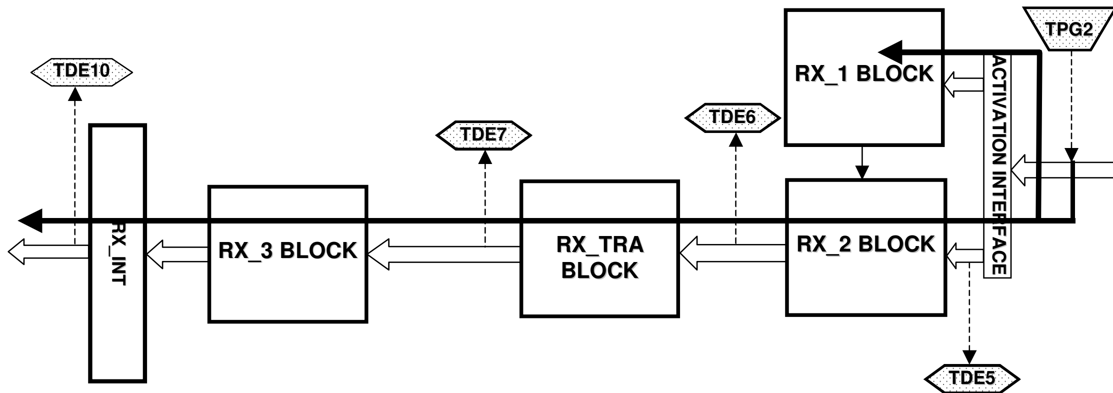

Figure 6.9. Receiver test

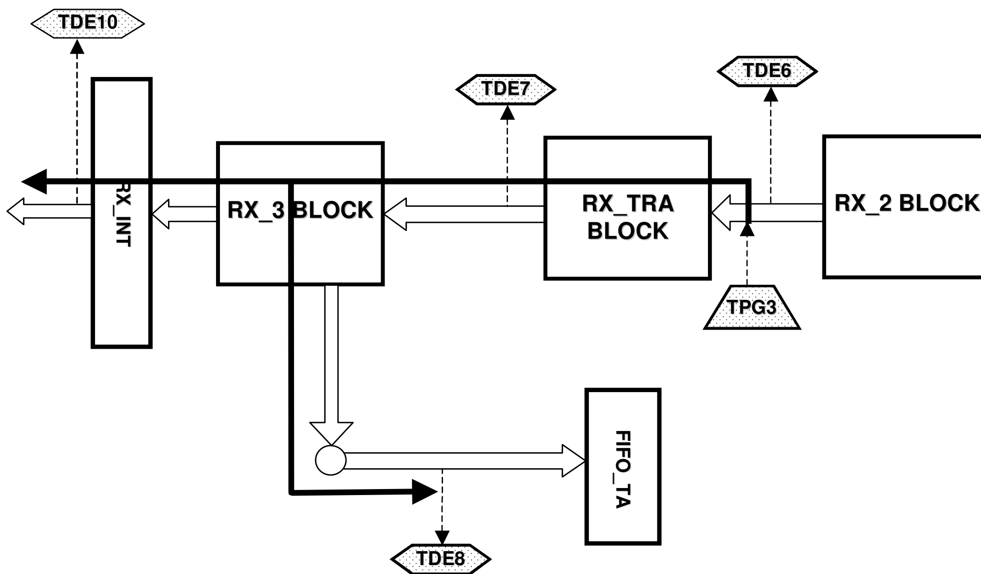

Figure 6.10. Test of the receiver internal loop

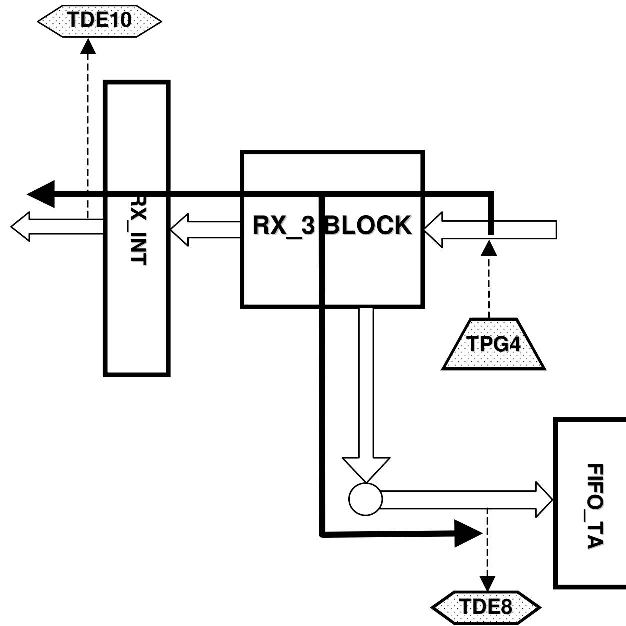

Figure 6.11. Test of the block Rx_3

The third test (TPG2 in Figure 6.9) is just a receiver test. Consequently, only compactors on the receiver side are able to collect some data (TDE5, TDE6, TDE7 and TDE10). The receiver feedback loop for this test is disabled. This test should give more detailed information on the forward datapath of the receiver.

The fourth test (generated from TPG3 in Figure 6.10) takes a broader look at the receiver feedback operation. Three TDEs are included in the process of testing (TDE7, TDE8, and TDE10).

The focus of the fifth test (TPG4 in Figure 6.11) is the operation of the very complex block Rx_3. Accordingly, only TDE8 and TDE10 are able to collect data. In this block some huge circuits are situated, such as Viterbi decoder and Deinterleaver. Here it is of utmost importance to provide a method that performs test with maximum coverage. A suitable combination of different tests can give us detailed and valuable information on the correctness of our GALS baseband processor.

For our GALS baseband processor, results of the synthesis show that the BIST circuitry occupies around 3.5% of the area. Hence, the insertion of BIST is paid with a relatively low area overhead. Additionally, BIST insertion did not lead to any significant performance drop and the functionality of the GALS system is preserved. Therefore, the BIST technique could be successfully used in the GALS systems.

Due to the lack of tool support, it is very hard to calculate the test coverage achieved with this BIST approach. Much of the synchronous design involves DSP pipelines. Those blocks are easy to test, because of the propagation of errors to the outputs. However, there are a few blocks that are only

implicitly part of the dataflow and are consequently hard to test. For example, in the tracking synchroniser (block Rx_1), the activity of autocorrelators and the plateau detector block is not visible from outside. Therefore, the fault coverage of those blocks is rather low. According to the switching analysis of the GALS baseband processor we can roughly estimate the system fault coverage to 90%. Nevertheless, further work in this direction is needed. However, the estimated test coverage that we can reach is significantly higher than with a pure functional test. The reason for increased test coverage compared to a functional test is increased observability and controllability. We can run hierarchical tests and start the test at different internal positions of the system. Unfortunately we cannot reach the test coverage of scan-based approaches. However, on the level of locally synchronous modules, scan-chains could be implemented.

Additionally, we have performed an indirect estimation of the achieved test coverage. We have used our BIST technique as a yield parameter for the after-production test of the GALS baseband processor chip. On the same wafer we fabricated some RAM blocks, which are exhaustively tested with a fault coverage of 100%. Having in mind that the average defect density is approximately constant on the same wafer, it is conceivable to perform the following analysis. According to Price's model [PRI70], the yield formula for exponential defect density distribution is given by:

$$Y = \frac{1}{1 + A\overline{D}}$$

where A is the area sensitive to defects and $\overline{D}$ is the average defect density.

From the yield measured for the RAM blocks, we interpolated the expected yield of the baseband processor to be 20.97%. The measured yield value according to our BIST scheme was 14.7%. Those numbers are based on an experimental technology run, and they are not relevant for process characterization and general yield figure of the qualified process technology.

We conclude that this BIST technique can be used as a suitable method for prototype verification. In combination with the scan approach for locally synchronous modules, we can even use this concept as a basis for the manufacturing test.

Chapter 7

Implementation and Evaluation of GALS Systems

7.1 Introduction

One of the main showstoppers for wider industrial application of systems with asynchronous components is the lack of asynchronous EDA tools. Therefore, it is very important to define a design-flow for GALS systems based on available tool support. In this chapter, a proposal for the design flow will be presented. On the basis of the design flow, an implementation of GALS systems is performed. Finally, some results of the implementation will be given. A comparison of the GALS approach with the pure synchronous implementation of the same system will conclude this chapter.

7.2 Design Flow

Defining a design flow for Globally Asynchronous Locally Synchronous systems is a difficult task. In general, designs that have asynchronous parts are relatively difficult to implement. A major problem is the lack of support for asynchronous logic designs in commercial EDA tools. Starting from behavioural simulation, the designer can experience that current HDLs (hardware description languages) such as VHDL and Verilog do not satisfy the needs of asynchronous datapaths. Even more problems will become apparent in the process of synthesis. The synthesis tools available on the market do not offer any support for asynchronous design. The analysis of critical paths is not possible. Furthermore, synchronous layout tools have special optimization procedures (as, for example, in-place optimization or timing-driven placement) that are not suited for asynchronous components.

However, several asynchronous EDA tools have been developed and are available to the users. There are two main categories of those tools. First, there are tools for synthesis of hazard-free asynchronous controllers. Some examples are Petrify [COR96], MINIMALIST [FUH01] and 3D [YUN99a]. In general, these tools are a good basis for the synthesis of relatively simple asynchronous controllers. They do not offer support for designing large systems, and they do not integrate any simulation tool. On the other hand, those tools are very useful help when some relatively simple asynchronous component has to be generated.

The other category of EDA tools offers a complete design framework for system description, simulation and synthesis. There are very few examples of such tools. Examples are TANGRAM (recently offered commercially to the market with the name HASTE) [BER91, HAN05], BALSA [BAR00] and TAST [DIN02]. Unfortunately, for GALS systems none of them is really useful. Due to immaturity, those tools achieve only sub-optimal synthesis results and they support the design-flow only up to the layout phase. The main obstacle for their application in the GALS area is that they are directed towards asynchronous designs and not to mixed synchronous-asynchronous designs.

Therefore, it was necessary to formulate our own framework for GALS system design. This design flow is based on existing tools, combined with our scripts. The proposed design flow for developing our GALS system is a combination of the standard synchronous design-flow with addition of specific asynchronous synthesis tools. However, most of the tools are taken from the pure synchronous world. The reason for that is simple: In general, the asynchronous part of the GALS circuitry is very small in comparison with the synchronous part. Accordingly, simple asynchronous circuits can be generated with the use of asynchronous synthesis tools and then embedded in complex synchronous blocks.

A graphical view to our design flow is shown in Figure 7.1. All synchronous circuits are designed in VHDL and synthesized for our in-house 5-metal layer 0.25 μm process with a FO4 delay of 140ps using a standard cell library and Synopsys Design Compiler.

All asynchronous controllers are modelled as asynchronous finite-state machines (AFSM) and subsequently synthesized using the 3D tool. 3D guarantees hazard-free synthesis of extended and normal burst-mode specifications. This tool is used because we predicted a need for extended burst mode (XBM) specifications. This tool is in the moment the only one that allows synthesis of the hazard-free XBM asynchronous controllers. Finally, the extended part of the specifications was not needed in the case of our asynchronous controllers. We have defined all controllers with burst mode specifications. On the other hand, it is also conceivable to specify the controllers with Signal Transition Graphs (STGs) or Petri-nets. Consequently, all tools that support hazard-free synthesis of asynchronous controllers can be used.

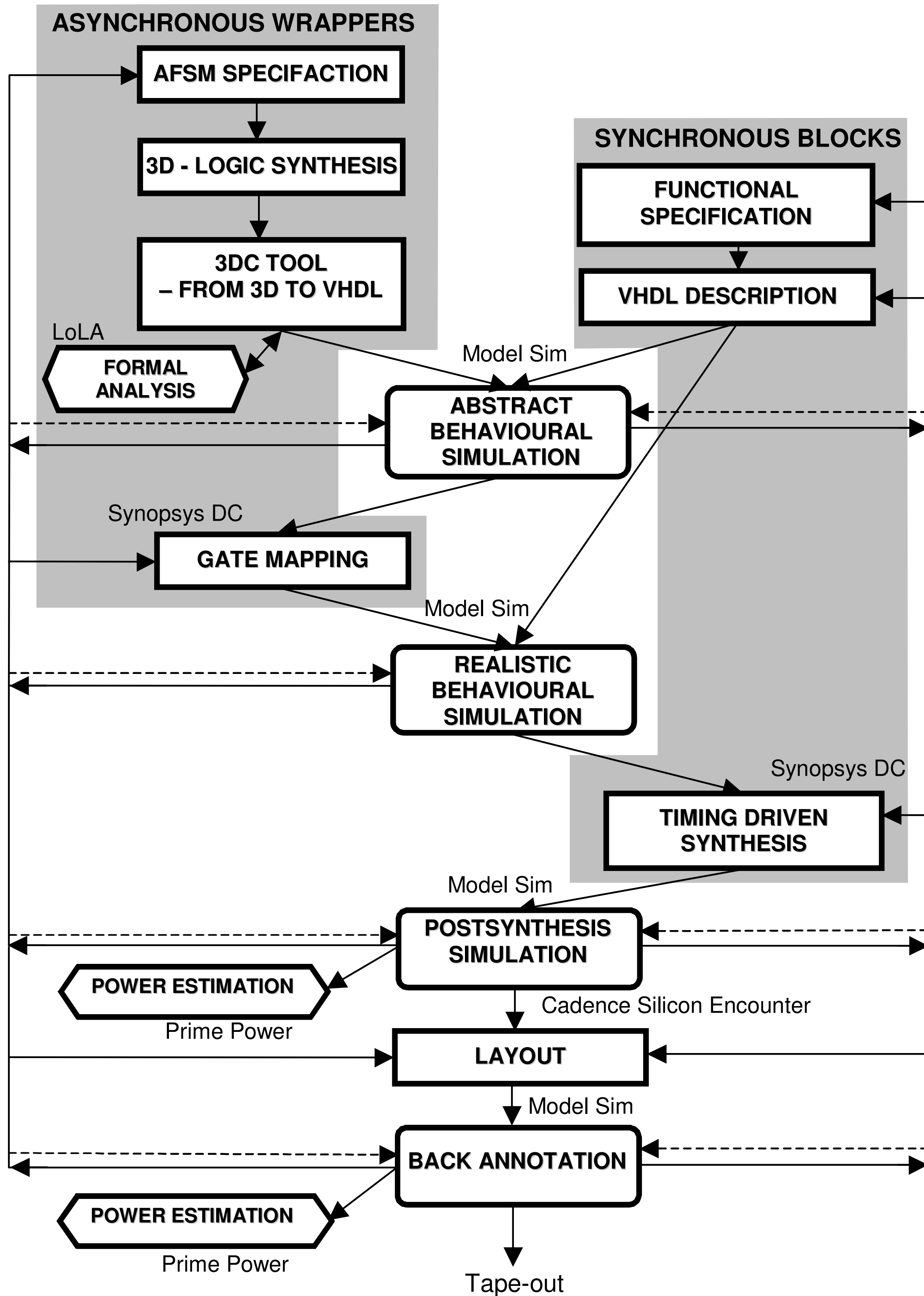

Figure 7.1. Design flow for GALS

The logic equations generated by 3D are automatically translated into structural VHDL using an own developed tool, called 3DC. The generated structural VHDL code represents a direct conversion from the logic equations generated from 3D. Further synthesis in Synopsys Design Compiler will

perform technology remapping. Consequently, Design Compiler cannot destroy the hazard-free behaviour of the AFSM but it will annotate realistic timing behaviour of the asynchronous block. After that, a formal analysis of the asynchronous controllers was performed in order to verify the hazard-free operation of the complete asynchronous wrapper.

In order to generate a complete and reliable flow for GALS design, two types of behavioural simulations are performed. Initially, a functional verification is performed at the level of VHDL descriptions. This simulation run should confirm the conceptual correctness of the system. Accordingly, for this simulation run, the complete synchronous part is modelled in behavioural VHDL code. Asynchronous wrappers are modelled in structural VHDL, which is the output of the 3DConverter tool. Additionally, in this structural code, some artificial delay is assigned to every logic operation. This is done in order to avoid races between the signals in the asynchronous part. During simulation, it is useful to enter realistic values for the clock signal. This allows correct adjustment of the system performance. Therefore, in this simulation the ring oscillators are described as real netlists and annotated with the appropriate SDF (Standard Delay Format) file.

On the basis of the synthesized asynchronous parts and VHDL description of the locally synchronous (LS) modules, a realistic behavioural simulation is performed after the abstract simulation. The complete asynchronous wrapper is annotated with a SDF file (based on the typical PVT values). This simulation will, with high probability, prove the correctness of the system function, but also can be performed very fast, because all LS modules are still modelled in behavioural VHDL.

Subsequently, a complete gate-level simulation is performed using both asynchronous and synchronous parts as synthesized netlists. This simulation is similar to any after-synthesis simulation in the purely synchronous world. Of course, due to the large number of annotated gates, this simulation run is rather slow in comparison with the one previously described. Accordingly, it should be performed only when all possible problems and hazards from the behavioural simulation have been resolved.

Layout is done using conventional synchronous layout tools. In particular, we have used two tools: Cadence Silicon Ensemble and Cadence SoC Encounter. The later one is much more advanced and better suited for complex designs. In the layout phase, we did not use advanced features as in-place optimization or timing driven placement and routing. The main reason for that was our desire not to unsettle sensitive timing issues in the asynchronous part of the system. However, an experienced user may use these features for the synchronous blocks only. The main feature, which is extensively used, is clock-tree generation. Specifically, clock-trees in the GALS blocks do not start from the clock pins but from internal gates inside the GALS block where the clock signal is generated. Furthermore, clock-tree generation is used for separate BIST domains and even for reset-tree generation.

After layout, back-annotation is performed. Behavioural, post-synthesis simulation, and back-annotation are performed using a standard VHDL-Verilog simulator.

One of the most important system issues is power consumption. Therefore, to efficiently design the circuit, some power estimation is needed. Power estimation is usually performed after synthesis (in order to create adequate power rings for floorplanning) and after layout (to estimate general power consumption of the chip). However, power figures may vary, depending on the application scenario of the circuitry. It is best when power estimation takes into account real switching activities of the circuit. During development of the GALS system, we have always used a realistic application scenario. Power estimation based on switching activities is done using the Synopsys Prime Power tool. Switching activities in this tool are annotated via a VCD (Value Change Dump) file, created during simulation.

7.3 System Integration with GALS

The original synchronous design flow for the baseband processor suffered from significant problems with global timing, clock tree-generation and clock-skew. Therefore, the design process was very prolonged and iterative.

The current CAD tool support is not sufficient to provide a fast and robust automatic design process, which is mandatory for this type of applications. GALSification provided some improvements for system integration. Challenges like global clock tree generation with an enormous number of leaves, clock divider and handling of clock gating simply disappeared. Clock skew within smaller clock domains was significantly reduced. For example, the maximum clock skew for the synchronous baseband processor was 660 ps. For the GALS design, we have reached 486 ps. However, with more stringent constraints even better results can be achieved. Since there is no global clock tree, timing closure of the complete design was achieved much more easily.

As this was the first complex GALS chip that we have designed, several new issues appeared. The main difficulties were lacking tool support for asynchronous components, or immaturity of asynchronous tools. For example, due to the limitations of the 3D tool, a direct gate-mapping of the generated logic equations was not possible. Therefore, many operations had to be performed manually. This degrades the performance of the final design and introduces additional delay in the design process. Additionally, the wrapper evaluation and improvement was performed in parallel to the GALS chip design. These issues caused some iterations of the GALS design process.

However, the general conclusion is that GALS led to a significant simplification of the system integration process. This simplification results in shorter design-to-market time and lower design cost.

7.4 Conceptual GALS Design Framework

In the previous section, our GALS design flow was presented. Although this proposal allows the relatively efficient integration of large synchronous blocks, the design flow is not completely automated. Many tasks must be performed manually. Some of the design steps require a lot of design experience in the area of asynchronous and synchronous design.

Therefore, there is a strong need to define a design flow that will incorporate all necessary requirements for a simple and effective implementation of GALS systems. This concept that we refer as "GALS Design Framework" (GALS DF) is shown in Figure 7.2.

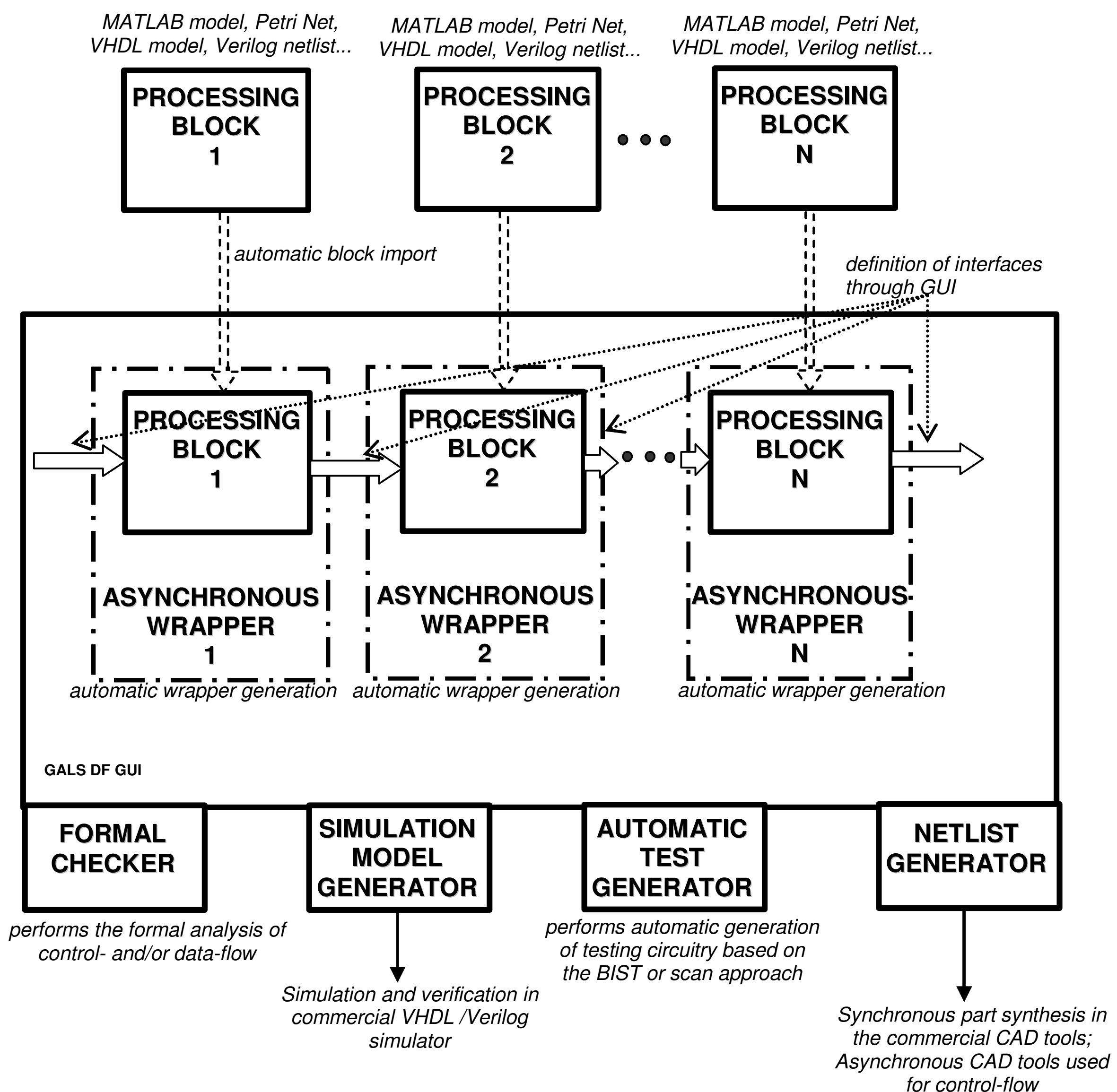

Figure 7.2. GALS design framework

The design framework is organised around a general graphical user interface (GUI). This environment has to support the multi-level description of the processing blocks. The highest level could be, for example, a MATLAB description (or any other high-level abstract model). However, the blocks could also be described by their control-flow (using a Petri-net model). Finally, at the RTL and gate level, it would be possible to use a VHDL model or a Verilog netlist. In the DF GUI it should be possible to import a specific processing block and to define its model for every abstraction level.

Additionally, the DF GUI allows the definition of interfaces and interface protocols between processing blocks. This should include the insertion of necessary interface blocks to support the communication mechanism, such as different join and fork circuits, or some bus structures.

Finally, it is possible to automatically generate the asynchronous wrapper around every processing block. Consequently, for every wrapper it should be possible to define the data width, time-out period, and other important parameters.

On the basis of the defined processing blocks and their mutual interfaces, the analysis of the defined complex system is performed. For example, at the level of the MATLAB description it is possible to invoke the MATLAB simulator and to perform the functional simulation of the abstract system model.

On the data-flow level, using Petri net models of the processing blocks, the formal analysis of the system using a formal model checker (in Figure 7.2) is facilitated. The design framework should automatically generate Petri-net models for wrappers and for interfaces. Using this analysis it should be possible to verify the hazard-free and deadlock-free behaviour of the system as well as liveness and reachability. Furthermore, a performance analysis is conceivable. Generally, the results from the formal analysis should give the designers hints about a possible GALS partitioning and the possibility to verify the control-flow for the proposed GALS partitioning in the system.

At the VHDL and the Verilog level, a simulation model generator (Figure 7.2) should be used in order to generate a complete model of the system. This tool should generate the VHDL model or a Verilog netlist for asynchronous wrappers and interfaces and integrate that code with the code for the processing blocks. From that tool, the standard VHDL/Verilog simulators are invoked.

To support the design, DFT features are mandatory. Therefore, an automatic test generator is included in GALS DF. The purpose of this tool is to automatically insert test circuitry and to perform automatic test pattern generation (ATPG) if needed. The test strategy could be based on BIST or on scan test and the structure should be defined by the user.

Finally, the netlist generator tool is required. It should generate the netlist on the basis of the VHDL description (or even a MATLAB model) of the system. This feature requests the combination of commercial CAD tools (as Synopsys Design Compiler) for processing blocks and special

asynchronous tools for asynchronous wrappers and interfaces (as 3D, Petrify, Minimalist, Balsa, Haste...). After generation of the system netlist, the after-synthesis simulation can be performed or the layout process may be invoked.

The development of such a design framework would be very important for automation of the GALS design process and it would shorten the time-to-market for GALS systems. However, this concept is not yet implemented and at the moment it is just a basis for future explorations and research.

7.5 Asynchronous Wrapper Implementation

Following the proposed design flow, the gate mapping of the developed asynchronous wrappers was performed. In Table 7.1 some synthesis results for our 0.25μ CMOS process are given. We have performed implementations of three basic types of wrappers: internally driven (based on ring oscillator), externally driven, and mixed internally/externally driven wrapper. All circuits are fitted with a reset logic, necessary for initialisation. The area for all circuits is given in μm^2 and equivalent inverter gates. The local clock generator for the internally driven wrapper is tuneable, similar to the one described in [MUT01]. For deriving the figures in Table 7.1, a 9-bit data latch connecting two asynchronous wrappers is included. However, for designing a GALS system, the latch width that corresponds to the application has to be taken into consideration. As can be seen in Table 7.1, the circuit area for the complete internally driven asynchronous wrapper is equivalent to about 1.3 k inverter gates.

Table 7.1 Asynchronous wrapper circuit area

Block \ Wrapper type	Int. wrapper		Ext. wrapper		Int-Ext wrapper	
	(μm^2)	**(gates)**	**(μm^2)**	**(gates)**	**(μm^2)**	**(gates)**
Input Controller	7100	118	7800	130	7800	130
Output Controller	3920	65	3900	65	3981	66
Time-out Detection	5179	86	5118	85	5632	94
Clock Control	4732	79	4631	77	4895	82
Clock Generation /Arbiter	55026	917	4489	75	58682	978
Total wrapper area	**79929**	**1332**	**29879**	**498**	**84682**	**1411**

For the internally driven wrapper, the largest area (917 inverter gates) is needed by the tuneable clock generation. This is due to the fact that for tunability it is needed to add a significant overhead to the clock generators. However, for an average size locally synchronous module of about 100,000 gates, this asynchronous wrapper would add just about 1% of the overall silicon area. An externally driven wrapper uses much less area (around 500 gates) due to the elimination of local clock

generator. The mixed-mode wrapper shows similar figures as the internally driven wrapper (1.4 K gates).

The throughput of the GALS system after synthesis was determined by simulation. The results of this evaluation are given in Table 7.2. In our technology, the wrappers are operational up to about 150 Msps in request-driven mode. However, this number depends very much on manual gate-resizing done after initial gate-mapping. Therefore, there is some unfairness of the results in the table. In local clock generation mode, the maximal throughput reached with internal clocking is in the range of 85 - 100 Msps. With external clocking we can reach a slightly lower speed of about 80-85 Msps.

However, this is fast enough for moderate speed applications. In any case, with the applied CMOS process (0.25µ), it is expected to cover operating frequencies of around 100 MHz. For more advanced technologies (0.18µ, 0.13µ, 0.09µ...) operating frequencies are higher. On the other hand, the maximum throughput of the asynchronous wrapper will be increased as well. That indicates that the application of the proposed asynchronous wrapper leads to very promising results, even for high-speed datapath applications.

We have also presented latency figures for the different wrapper structures. Latency is defined as the time that data needs to pass from the last register stage of one GALS block to the first register stage of the subsequent GALS block in the datapath. As we can see from the table, this time is in the order of 5 – 7.5 ns.

The power figures presented in Table 7.2 are extracted from simulation of different wrapper configurations in a realistic scenario of receiving, processing and transferring one data burst at 50 Msps datarate. The interesting fact is that the power consumption of the internally driven wrapper is only around 10% higher than the one from the externally driven wrapper. The reason is relatively low power consumption of the ring oscillator for these moderate frequencies. In general, a single wrapper consumes around 1 – 1.5 mW. Hence, normally the wrapper insertion should not lead to any significant power overhead in a GALS system.

Table 7.2 Evaluation of asynchronous wrapper performance and power consumption

Component / Parameter	Int. wrapper	Ext. wrapper	Int/Ext wrapper
Max. throughput - request mode (Msps)	119	133.9	148 / 148
Max. throughput - local mode (Msps)	86.9	79.4	96.2 / 82.6
Latency (ps)	7590	5150	6320 / 6290
Power (mW)	1.12	1.01	1.4 / 1.26

7.6 Experimental GALS Chip

In order to validate the request-driven GALS concept, we decided to implement a simple GALS demonstrator first. The main goal for this implementation was to check the feasibility of the proposed GALS technique. In this implementation we have employed asynchronous wrappers with an embedded ring oscillator, as described in the previous subsections. In this way, we wanted to check the operation of the wrappers and to identify potentially unknown hazard situations. Additionally, we were aiming to verify our design for testability strategy based on BIST.

The simple GALS demonstrator consists of a synthesised asynchronous wrapper fitted with a behavioural model of a 'dummy' synchronous FIFO. In Figure 7.3.a, a simulation trace for different modes of operation is shown. The chip structure is given in Figure 7.3.b. The 'dummy' synchronous block is a 21-stage FIFO. The simulated system consists of three cascaded GALS blocks that are fitted with an ideal token producer and a consumer at input and output respectively.

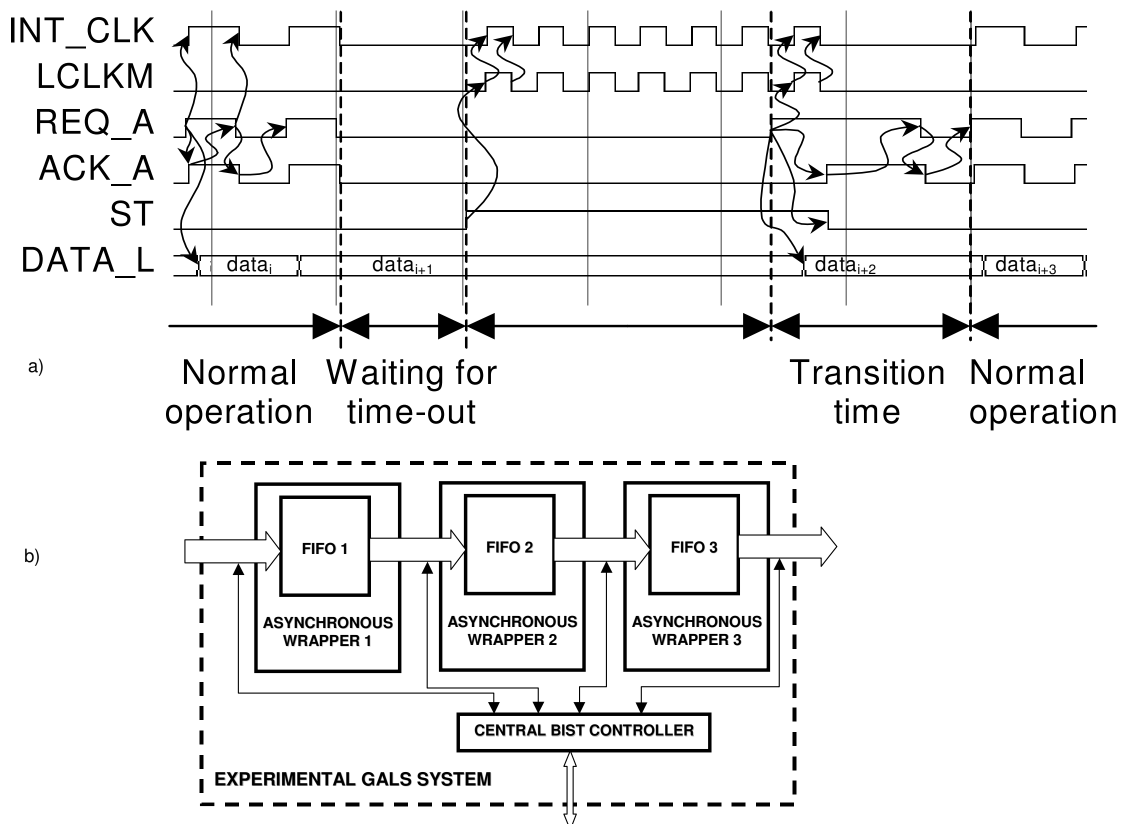

Figure 7.3. Asynchronous wrapper operation (a) in the experimental design structure (b)

Figure 7.3a shows the generation of signal *INT_CLK* within one asynchronous GALS wrapper. In this figure we can see that *INT_CLK* is generated from signals *LCLKM* and *REQ_A.* Four different modes of operation of the asynchronous wrapper are shown. First the 'normal' mode of operation can be observed, where a standard handshake is performed on the lines *REQ_A* and *ACK_A*. Every

request signal is interpreted as a new clock cycle. When *REQ_A* is kept at ‘0’, the circuit enters a second state: waiting for time-out. During this period the internal clock signal is disabled. The time-out event is indicated by activation of signal *ST*. Subsequently the transition into a third mode occurs. The local clock signal *LCLKM* will be activated and in turn drive signal *INT_CLK*. Further, the *REQ_A* line indicates the arrival of new data before deactivation of *LCLKM.* Now the system enters a transition mode. In this mode the initiated local clock cycle must be completed and subsequently control of the internal clock signal is handed over to the request line. In the last period of the simulation trace the circuit has resumed ‘normal’ request-driven operation.

This simple experimental GALS circuit was implemented as a chip in our in-house 0.25µm CMOS technology. The circuit was fitted with BIST logic in order to simplify testing. The structure and concept of the applied test logic was the same as described in the previous chapter. We have implemented a single TPG for test vector generation and four different TDEs for extracting test results. The cell area of the implemented chip is around 0.3 mm^2. This layout was pad limited, and the final chip area was 4.47 mm^2. This chip was fabricated and successfully tested. The test setup is shown in Figure 7.4. Test results are generated using a logic analyzer and illustrated in Figure 7.5. The pass of the test is indicated with three short pulses on the *BIST_OK* line, and then with a stable high value.

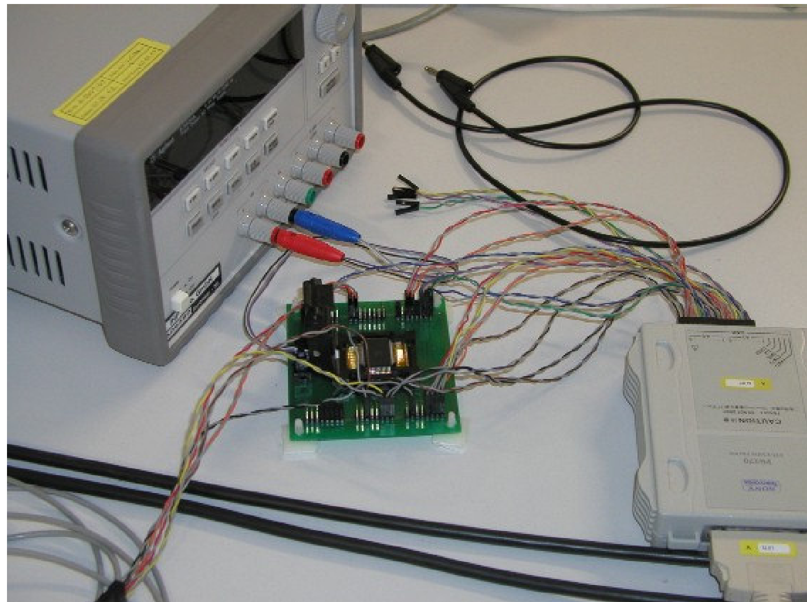

Figure 7.4. Testing experimental GALS chip

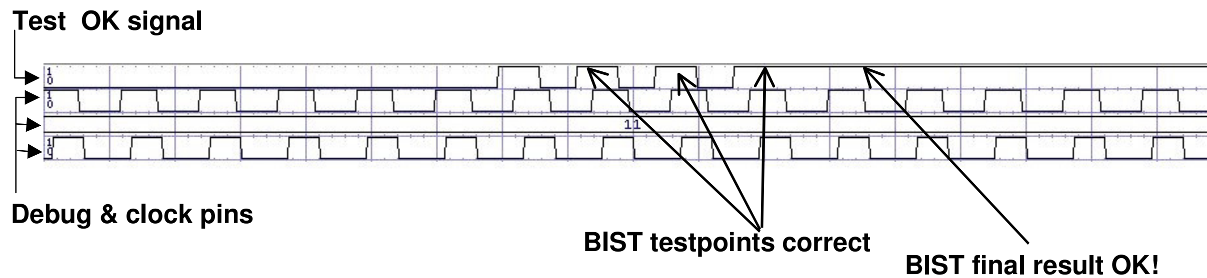

Figure 7.5. Test results collected from logic analyzer

7.7 GALS Baseband Processor Implementation

For the GALS baseband processor implementation, our previously described design flow was used. After GALS partitioning of the baseband processor, the created GALS system was behaviourally simulated and verified. After that, the synthesis of the individual synchronous blocks and the asynchronous components was performed and a unified netlist was generated. On the basis of the created netlist and a corresponding SDF file, a gate-level simulation was performed, which confirmed the results of the behavioural simulation.

The complete system was verified using two different sets of test vectors. One is based on the embedded BIST test and the other on a realistic transceiver application in the synchronous environment of the IEEE 802.11a modem. A set of five different BIST tests was performed as well as the functional test for eight different modulation schemes defined by the IEEE 802.11a standard. Results of the synthesis and power estimation are shown in Table 7.3. These results are based on the real transceiver application in the environment of an IEEE 802.11a modem. This functional test includes transmission of an IEEE 802.11a compliant data frame (100 B) and reception of a data frame of the same length.

It can be observed that synchronous functional blocks occupy almost 90% of the total cell area. The BIST circuitry requires around 3.5%, interface blocks (asynchronous interfaces and asynchronous to synchronous interfaces) 2.9%, asynchronous wrappers only 2% and the clock tree 1.7% of the area. The overall cell area is 22.78 mm^2. From these figures we conclude that the integration of the asynchronous wrappers did not result in any significant hardware overhead. The test logic contributed more to the overall silicon usage than the wrappers. However, since this was an experimental design, more effort is devoted to testing. For future GALS applications, the logic dedicated to test can be reduced. Additionally, some further investigations are required to reduce the complexity of interfaces in the GALS chip. Currently, those interface blocks are suboptimal and create much overhead.

Table 7.3. Area and power distribution in the GALS baseband processor

Component \ Results	Area [%]	Power [%]
BB chip	**100%**	**100%**
Clock trees	**1.7%**	**34.5%**
Sync. blocks	**89.5%**	**52.4%**
Rx_1 block	10.4%	6.9%
Rx_2 block	30.4%	15.9%
Rx_3 block	21.8%	17.2%
Rx_TRA	2.4%	0.7%
Tx_1 block	2.3%	1.1%
Tx_2 block	2.5%	0.6%
Tx_3 block	19.8%	10.0%
Asynchronous wrappers	**2%**	**2.9 %**
Rx1	0.2%	0.8%
Rx_TRA	0.3%	0.3%
Rx2	0.3%	0.6%
Rx3	0.3%	0.5%
Tx2	0.3%	0.1%
Tx3	0.6%	0.6%
Asynchronous interfaces	**1.6%**	**0.6%**
Join	0.0%	0.0%
FIFO_TA	1.5%	0.6%
As-sy interfaces	**1.3%**	**7%**
Rx_int	0.5%	4.4%
Tx_int	0.8%	2.6%
BIST	**3.5%**	**1.5%**
CBC	0.2%	0.5%
TDE	2.7%	0.2%
TPG	0.6%	0.8%

Based on the switching activity in a real transceiver scenario, dynamic power estimation with Prime Power was performed. In the simulation example the baseband processor receives one frame and transmits one frame. The synchronous blocks need most of the power (around 52.4%). A significant amount is also spent in the local clock trees (34.5%). Other important power consumers are asynchronous-to-synchronous interfaces with 7% and asynchronous wrappers with 2.9%.

The power consumption of the asynchronous wrappers including pausable clock generators is quite low with respect to the overall power consumption. On the other hand, the power consumption of the async-sync interfaces exceeds the usual limits for this type of circuitry. We have used complex FIFO structures in the interfaces to decouple blocks and increase the robustness of the system. However, these interfaces can be optimized, which would reduce their power consumption significantly. Additionally, some power is spent in the BIST circuitry, even during normal operation. The reason is the switching of small local clock trees in TPGs and TDEs, which are not disabled during ‘normal’ operation. With more careful design, this switching activity could also be avoided by gating these local clocks. Based on the final netlist after layout, the overall estimated dynamic power consumption is 324.6 mW.

Our GALS baseband processor was fabricated and the die photo of the produced chip is shown in Figure 7.6. In the floorplan it is noticeable that the receiver uses much more space than the transmitter. Furthermore, the most dominant blocks are as expected Rx_2 (processing synchroniser and channel estimator) and Rx_3 (dominated by the Viterbi decoder). The following blocks are also very complex: Rx_1 and Tx_3, both dominated by the FFT/IFFT processor.

The total number of pins is 120 and the silicon area including pads is 45.1 mm^2. For comparison, the equivalent synchronous baseband processor occupies around 34 mm^2. The significant increase in silicon area of around 32% for the GALS chip is mainly due to an architectural modification. For the GALS implementation, IFFT and FFT processors are realized as separate blocks in order to simplify the dataflow. In contrast, in the synchronous implementation, a single processor executes both operations. However, with reasonable effort it is possible to merge FFT and IFFT processors for the GALS implementation, as well.

To conclude, the design of the GALS baseband processor is an experimental design with potential for improvement. The main reason is that we focussed on robustness rather than efficiency. Furthermore, for this first complex design, some lack of experience was apparent. However, the results show very competitive numbers in some aspects. With some small optimizations in the future, we can hope for more effective results.

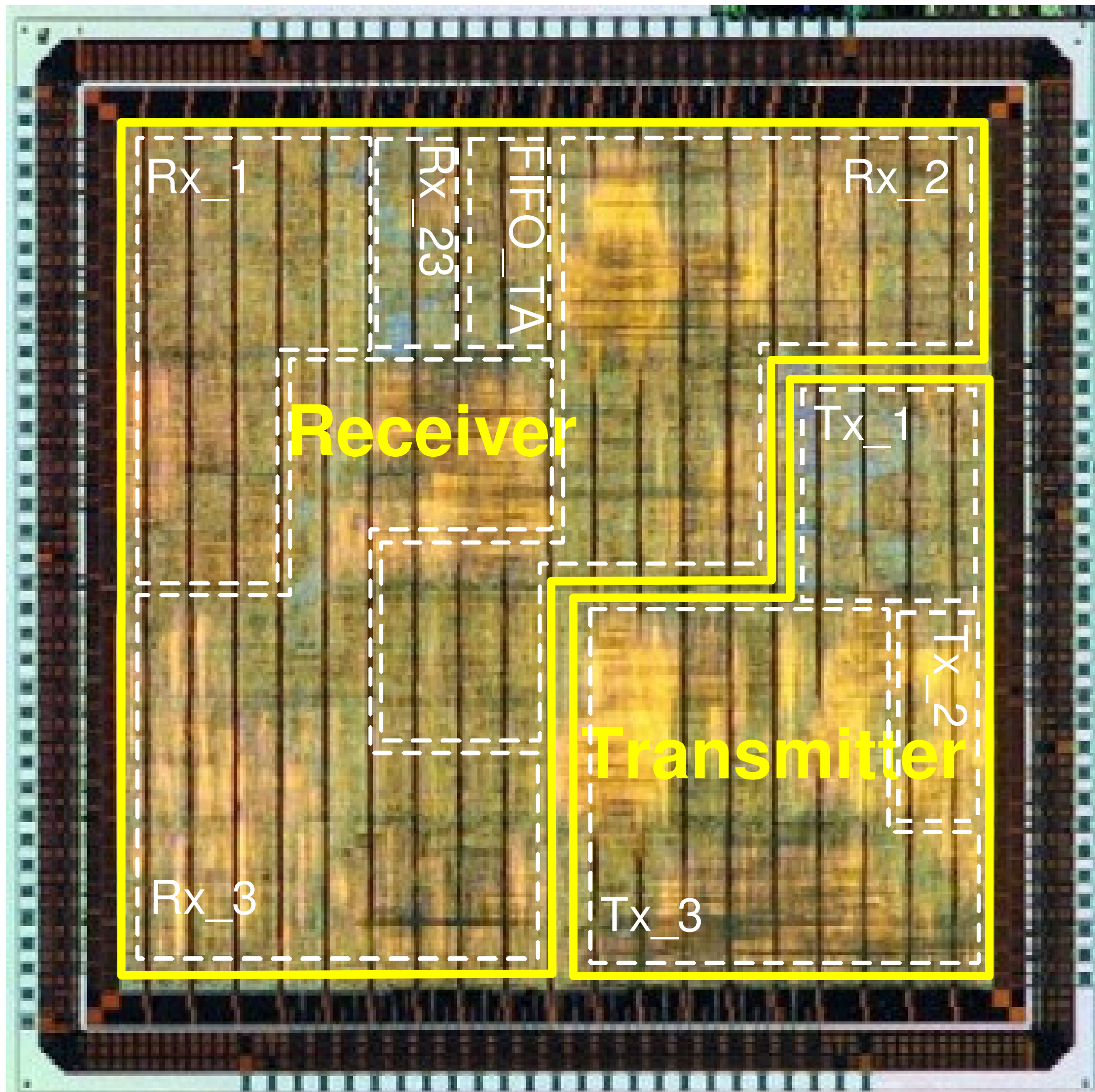

Figure 7.6. Die photo of the GALS baseband processor

7.7.1 Evaluation of Synchronous and GALS Baseband Processor

On the basis of a post-layout netlist and the corresponding power analysis, an interesting comparison of the pure synchronous with the GALS baseband processor can be performed. This analysis can quantify the influence of introducing GALS in a synchronous system, in terms of area or hardware overhead.

In Figure 7.7, the area allocation of the synchronous and the GALS baseband processor is shown. In the synchronous baseband processor, the BIST circuitry is not analyzed separately but it is a part of receiver and transmitter. From this figure, it is obvious that the decision to implement FFT/IFFT separately in the GALS processor leads to a significant change in the area profile. In the GALS implementation, FFT and IFFT use almost one third of the total cell area. Consequently, the share of the other components of receiver and transmitter decreases in comparison with the pure synchronous

solution. The introduction of asynchronous components in the system (asynchronous wrappers and different interfaces) resulted only in a comparatively small increase in area. Hence, GALSification as such, does not necessarily lead to a significant area increase. GALS circuits can limit the area overhead to 3-5% compared to its synchronous counterpart.

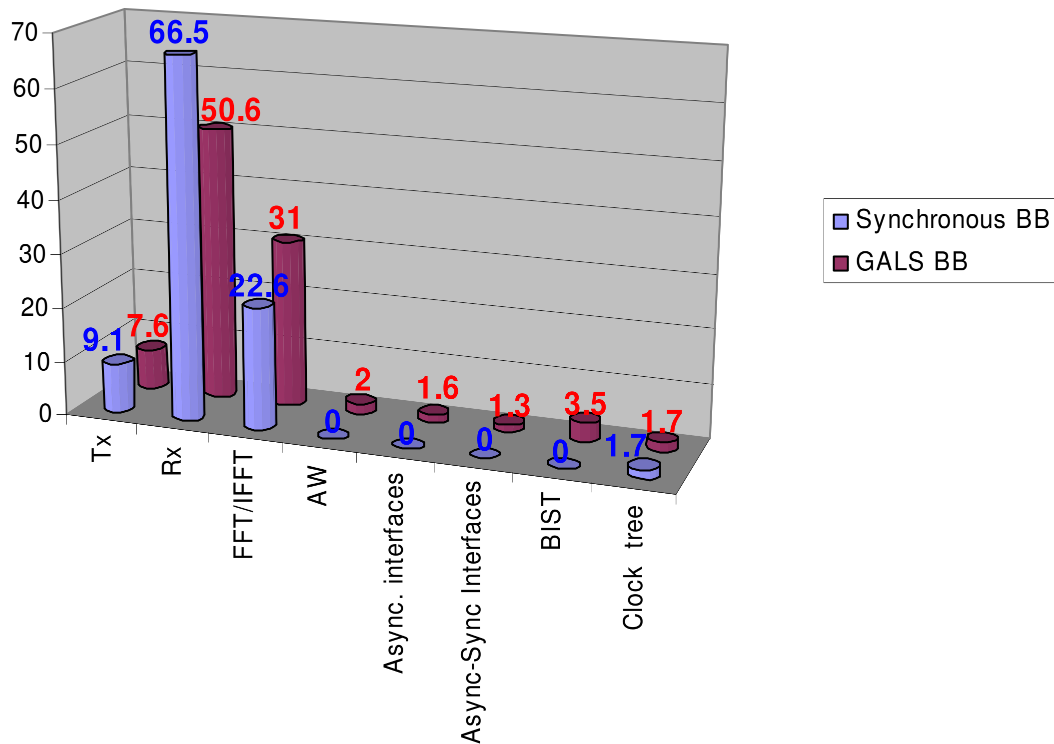

Figure 7.7. Area distribution in synchronous and GALS baseband processors

Even more interesting data can be generated from power profiling of the GALS and the synchronous processors. Results are illustrated in Figure 7.8. From this figure the greatest change in the power profile is observed in the transmitter. The power consumption of the GALS version is reduced due to the fine-grained clock gating mechanism. Additionally, the share of the clock-trees is reduced due to their smaller size. The power consumption in the GALS receiver is also reduced due to the improved power saving. The effective power spent in the FFT/IFFT is at the same level for the GALS and the synchronous processors. For the asynchronous component, as already mentioned, Async-Sync interfaces are the most dominant consumer. From their optimization, much better power results may be expected.

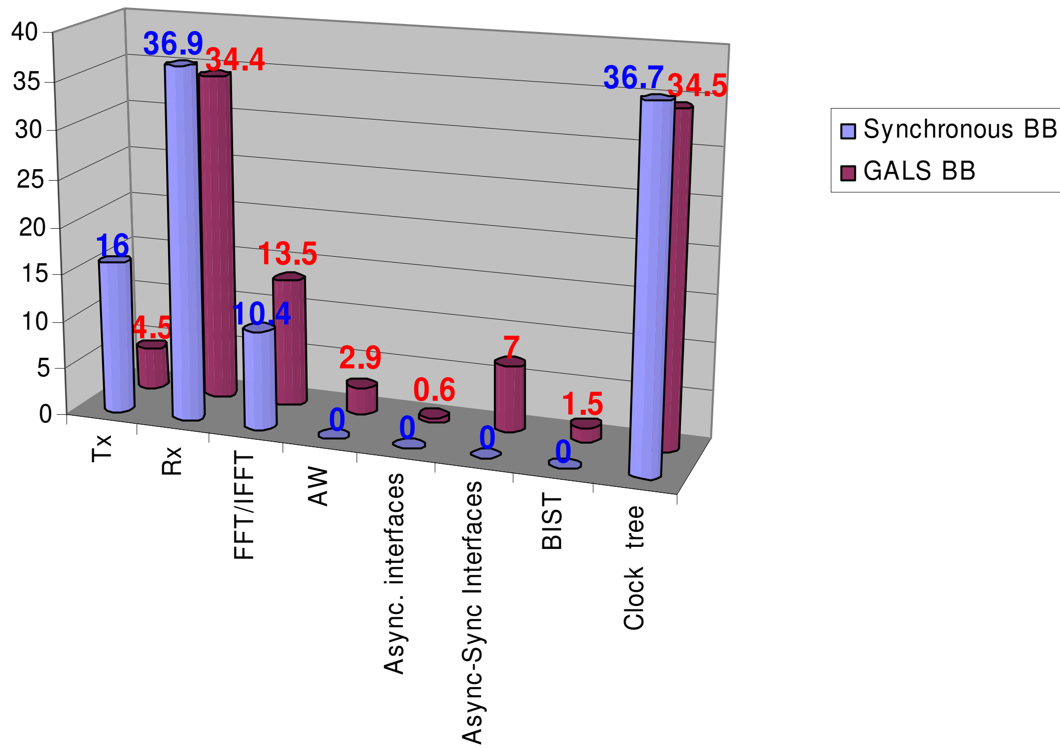

Figure 7.8. Power consumption for functional blocks in synchronous and GALS baseband processor

Chapter 8

Experimental Results

8.1 Introduction

In order to verify the behaviour of the fabricated chip extensive testing and measurements had to be performed. Three tasks were of major importance for us. The first goal was to verify the correct functional behaviour of the chip. Secondly, to justify the introduction the GALS technique we wanted to compare the GALS and the standard synchronous approach in respect to power consumption. Thirdly, the measurement of the noise level in different baseband processor chips was of special interest.

8.2 Functional Verification of the GALS Baseband Processor

An important task in the verification of any design is the after-production test. The first goal of the measurement and test procedure was to verify the functionality of the chip and to estimate the yield. For the after-production test, a hardware tester Agilent 93000 was used. However, the hardware tester is targeted to synchronous circuits. That means that the input and output patterns should be defined relative to tester clock cycles. Consequently, testing asynchronous logic is a very challenging task.

The GALS baseband processor has completely synchronously driven interfaces. Therefore, the generation of the test input vectors was not too difficult. However, the embedded ring oscillators and the arbitration units in the GALS lead to timing nondeterminism. The simulation of the system that is performed after back-annotation will not necessarily give the same results as the test of the fabricated circuitry. When correct setup is applied, the valid output tokens will eventually match both in the simulation and the testing case. However, the timing of token appearance in these two cases will probably be different. Therefore, although the outputs are also synchronised with the clock the nondeterminism in the GALS system complicates strobing of the output vectors. To solve this issue, we have used the embedded BIST circuitry in the baseband processor, as proposed in Chapter 6. The

output of the BIST circuitry is driven by an external clock and generates the result in predefined clock cycles. Having this in mind, the test can be performed in the standard way with the synchronous hardware tester. The results can be automatically or manually observed and verified.

The tester generates a simple input pattern which initiates the BIST. After that, the test operator should just observe the clock cycles where the *BIST_OK* line is activated. If the number of created pulses and the final value of this signal correspond to the simulated behaviour, the chip is correct. Otherwise, there is a malfunction in the chip. In Figure 8.1 a snapshot of the BIST test using our hardware tester is shown.

This test strategy makes the generation of the test program very easy. The BIST creates the test results at the moment when the BIST data compacting is already finished. Timing nondeterminism, always present in the GALS system, is avoided by externally clocking the BIST circuitry. However, the verification of the signals except the *BIST_OK* is not easily possible due to changes of the signal edges from chip to chip and from one test run to the next test run.

In our test suite we have used functional tests as well, in order to check the correct behaviour of the GALS chip in a realistic operational environment. This test was not part of the automatic test-flow for yield evaluation, but it was used for power estimation and noise characterization. The verification of the functional test was done manually.

Additionally, during the test we have noticed that the frequency of the ring oscillators is very sensitive to PVT variations. Therefore, some way of automatic calibration is needed. Otherwise, industrial application of this system will be limited. None of the commercial users of GALS systems would accept that in each produced chip the ring oscillators have to be manually tuned to the right frequency. Alternatively, for wider industrial application of GALS systems, it is desirable to avoid the use of ring oscillators and apply external clocks instead.

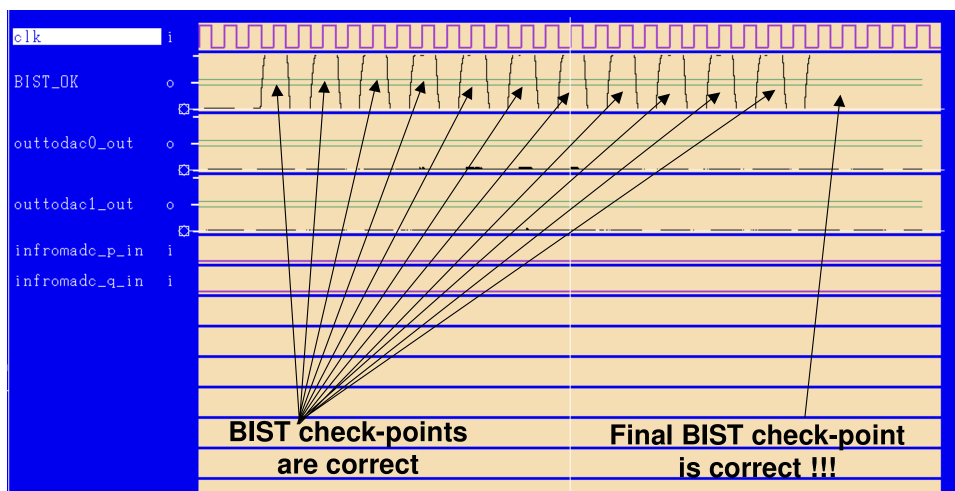

Figure 8.1. BIST testing on the Agilent hardware tester

8.3 Power Measurement

Performing the power measurements was relatively easy, because this feature is supported by the hardware tester. The measurement was done on the basis of a realistic scenario. This test lasts around 100 µs and represents the transmission of a 100 B frame followed by the reception of a 100 B frame at the transmission rate of 54 Mbps. For comparison, the same test was performed for the pure synchronous baseband chip, too. A statistical analysis was done on the basis of the results from the chips that have passed the after-production test. The results are shown in Table 8.1. One should have in mind that those numbers are derived from an experimental technology run, and they are not relevant for process characterization. The dynamic power dissipation of the pure synchronous baseband processor was around 332 mW, and for the GALS baseband processor slightly lower, at around 328 mW. Due to the much larger chip area the static power in the GALS baseband processor is higher. However, the static power consumption is normally in the order of µW. Due to some technology problems, this number was increased for several orders of magnitude for those runs where the baseband processors were fabricated.

Table 8.1. Power consumption of the synchronous and GALS baseband chip

Power / Design	Static power [mW]	Dynamic power [mW]	Overall power [mW]
GALS baseband chip	**58,4**	**328,4**	**386,8**
Sync. baseband chip	**40,8**	**332,0**	**372,8**

According to the after-layout power estimation, the expected dynamic power reduction for the GALS version was expected to be around 17 %. The actually measured reduction was a bit more than 1 %. The reason for this discrepancy is the power estimation of the synchronous baseband processor that created too pessimistic results. In general, the GALS and the pure synchronous baseband processors use similar power saving strategies. The difference is that in the synchronous implementation, this strategy is realized using a clock-gating scheme, and in the GALS implementation it is embedded into the operation of the asynchronous wrappers. The advantage of the GALS technique is the usage of the several low complexity clock trees instead of one global clock tree. In addition to that, finer granularity of the clock domains leads to lower power consumption. However, the considerable power consumption in the asynchronous-synchronous interfaces diminished those advantages. Hence, the difference in the dynamic power consumption is very small. Additionally, some power is consumed in the BIST circuit clock trees. This can be avoided in future designs.

The potential for lower power consumption could be better utilized if GALS was used in a more aggressive way. For example, asynchronous-to-synchronous interfaces could be realized in a more optimal manner, since due to robustness they have a significant overhead. Moreover, the most

complex block Rx_3 can operate at a much lower sample rate then the currently used 80 Msps. This conservative sample rate was chosen in order to re-use the design of locally synchronous modules. In principle, even 60 Msps are sufficient to fulfil the functional demands of the system. However, this normally requires certain modifications of the VHDL code for the locally synchronous modules.

To conclude, the GALS implementation led to small improvement in the dynamic power consumption in comparison with the pure synchronous solution. However, some parts of the implementation were not optimal which resulted in relatively small gain. For future implementations, with more careful planning of the design, the achievable power reduction can be significantly higher.

8.4 Supply Noise Measurement

Performing EMI measurements is generally quite complicated. Therefore, we decided to use the supply current profile as an indirect representation of EMI. Initially, we intended to directly measure the supply current profile via a small shunt resistor. However, this was not possible due to blocking capacitors on the board which are necessary to limit the supply voltage drop to an acceptable value.

Therefore, we decided to measure the variations of the supply voltage of the inner processor core using a functional output pin that was permanently set to logic high value. This is possible since, in our design, core supply voltage and I/O supply voltage are directly connected. During our measurements, the p-MOS transistor of the pad driver operates in the linear region (Figure 8.2) and can hence be modelled as a resistor, directly connected to the chip-internal supply voltage. This method proved very effective for our purpose. The voltage variations were analysed both for the synchronous and the GALS baseband processor operating in receive mode. After that, the spectrum of the power supply voltage was calculated.

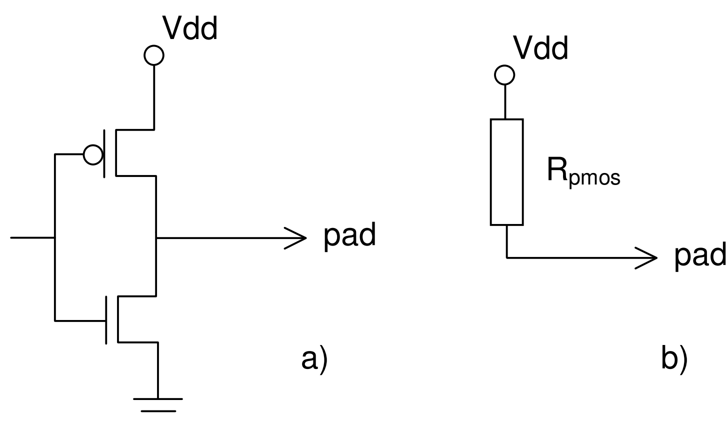

Figure 8.2. The final inverter stage of the output pad (a), and its simplified equivalent circuit for the p-MOS transistor in the linear region of operation (b)

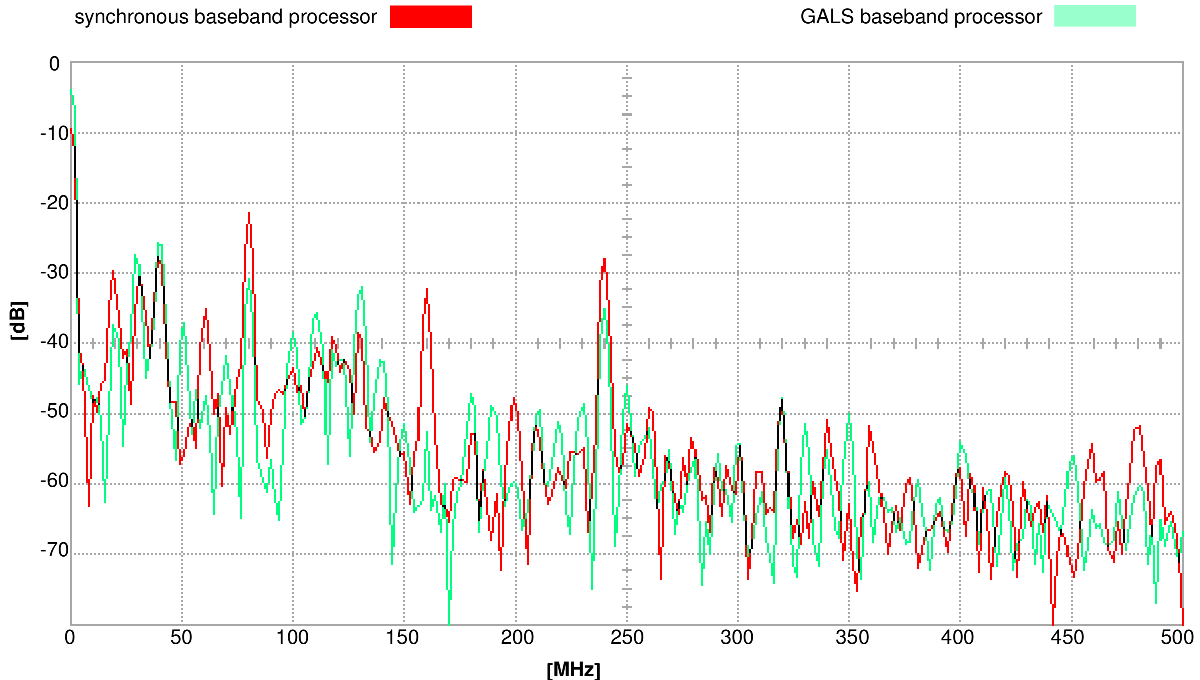

Figure 8.3. Spectral profile of the supply voltage (receiving scenario)

The most interesting case is to measure the spectral profile of the functional test when the greatest switching activity is generated. In our test, this is the part of the test when the baseband processor operates in receive mode and the Viterbi decoder is activated. From the oscilloscope waveform we confirmed that in this processing phase, the supply voltage showed maximum variations. The voltage spectrum is shown in Figure 8.3. This spectrum covers frequencies up to 500 MHz. Having in mind typical operating frequencies of the system, this is a sufficient spectrum. In Figure 8.3 there are strong spectral components at the frequencies of 20, 80, 160 and 240 MHz visible for the pure synchronous baseband processor. Obviously, the synchronous chip generates the peak spectral components at the frequencies of the clock sources (20 and 80 MHz). However, significant components are at the harmonic frequencies. In the GALS circuit, these components are less emphasised. The absolute maximum of the power spectrum of the GALS circuit (at 40 MHz) is about 5 dB lower than the absolute maximum for the synchronous circuit (at 80 MHz). However, even for the GALS chip, the components at 40 and 80 MHz are significant, due to the fact that the complete external shell operates synchronously. Additionally, less dominant peaks at frequencies of 50, 100 and 130 MHz are visible in the GALS spectrum. They may result from mixing products of the actual settings of our stoppable clock generators in the asynchronous wrappers.

In Figure 8.4, another measurement result is shown. Here, we have analyzed the transmit mode where the switching activity is not so strong. The most active part in this period should be the FFT in the transmitter operating at 20 MHz. However, for the synchronous chip one can still recognise the peaks at 20, 40, 80, 120 and 240 MHz. For the GALS version there are peaks mainly at 30, 40, 60 and 100 MHz. However, the synchronous peaks are more dominant. The difference is not that emphasized

as in the previous example. For the synchronous chip, the maximum of the power spectrum is at 40 MHz and it is around 2-3 dB higher than the most dominant power peak for the GALS circuit (at the same frequency). The reason for the not so big differences between the GALS and the synchronous version is that GALSification in the transmitter is not very deep. There are only two real GALS blocks and hence, we cannot expect any significant noise reduction.

One additional important aspect of the noise measurement is the estimation of the instantaneous current peaks on the supply line. In a mixed-signal environment, this fact is very important for successful integration of digital and analog components. For the case given in Figure 8.3, the instantaneous supply voltage peaks are reduced from 140 mV (synchronous design) to less than 100 mV (GALS).

However, from both examples, the general conclusion is that the GALS approach creates a smoother spectrum without emphasized and frequent peaks when compared to the pure synchronous approach. However, from the analysis presented in Chapter 3, the level of EMI reduction is expected to be more emphasized. On the other hand, in the implemented baseband processor chip the number of GALS blocks was very limited and the effect of the noise reduction as well. An application with fine-grained GALS partitioning can achieve more impressive results. Additionally, pads may also have an impact on our EMI measurement. However, the achieved results give us hope that GALS can reduce EMI and crosstalk in mixed-signal designs. Additional noise suppression is possible using clock phasing within every particular locally synchronous module. Also clock jittering can be applied as a method for spectral reduction of EMI.

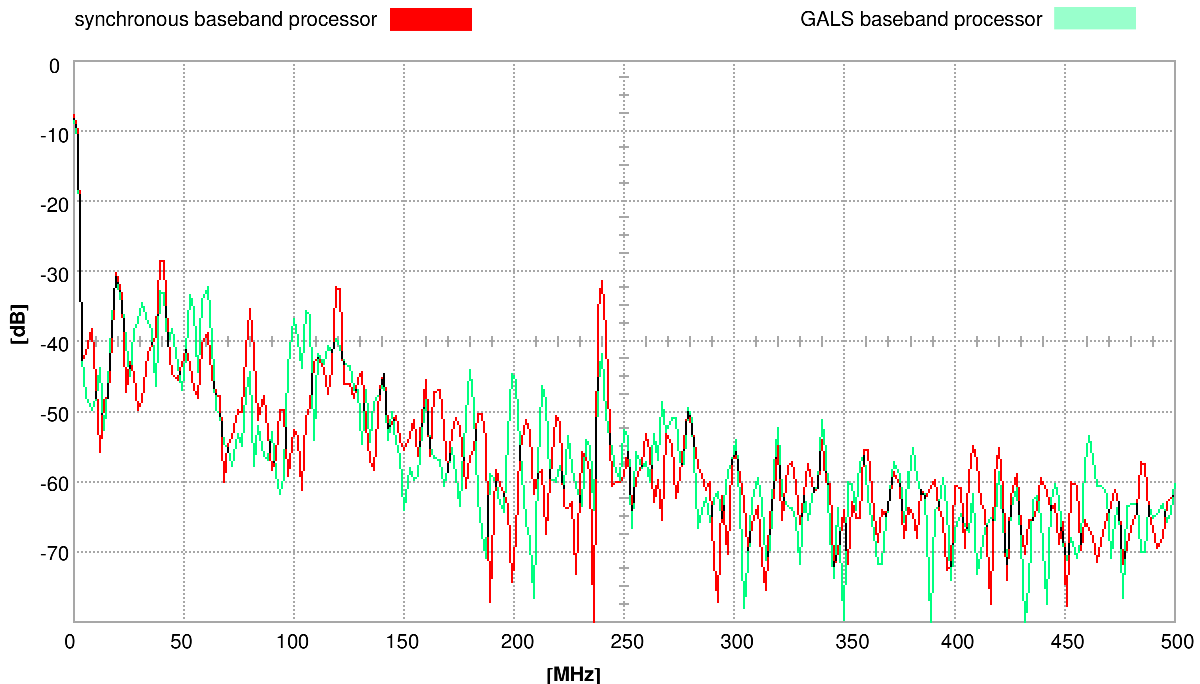

Figure 8.4. Spectral profile of the supply voltage (transmitting scenario)

Chapter 9

Conclusions

9.1 Achieved Results

In this thesis, a technique for asynchronous communication between synchronous blocks is presented. This novel GALS technique, reported here, allows easy integration of synchronous blocks into a complex system structure. The GALS concept is based on the application of an asynchronous wrapper, which is particularly optimized for datapath architectures with intensive and bursty data transfer between system blocks. The wrapper application is based on natural and simple rules for synchronous block design without notable hardware overhead. Our proposed method also offers explicit and effective power saving mechanisms. In this work, we presented one possible implementation of the proposed GALS technique. However, for a specific application, the circuits can be further optimised and hence made more area, speed, or power efficient.

As a feasibility study of the proposed GALS technique, the complete implementation of a complex IEEE 802.11a compliant GALS baseband processor for a WLAN modem has been undertaken. To our knowledge, this is one of the most complex GALS implementations on silicon reported so far. The challenging process of power-optimal GALS partitioning in the processor is described. The structure of the GALS baseband processor is completed with several additional blocks necessary for proper operation of the system and with suitable testing logic. The design flow used for this system is described and implementation results are presented.

By introducing GALS, our main goal of simplifying the system integration was achieved. Application of our GALS technique relaxed global timing, simplified clock tree-generation, and reduced clock-skew. Furthermore, the results of our measurements show that GALSification leads to slightly lower dynamic power consumption and a considerable reduction of supply noise.

9.2 Request-Driven GALS as a Solution – Pros and Cons

The proposed request-driven GALS technique differs significantly from other known approaches. This technique is optimized for point-to-point communication. There are some properties of the proposed technique that are unique. When the circuit is operating in request-driven mode, synchronisation at the input port is not needed. In this mode of operation, the incoming request signal drives the clock signal of the LS module. The determinism in the system is increased and the GALS block can be connected more easily to pure synchronous token consumers or producers.

However, there are some drawbacks as well. The hardware implementation of the wrapper is rather complex compared with some other GALS approaches and, in some cases, it can limit the performances of the system. System operation of our initial version of the GALS-wrapper still depended on ring oscillators, which require tuning. This will be too costly for most industrial applications. An advanced version with external oscillator was developed to fix this problem.

After the re-evaluation of the request-driven GALS concept, we can define the application area for this approach. The request-driven GALS technique is applicable for systems with complex datapaths based on point-to-point communication and bursty data transfer. The proposed GALS concept and the corresponding design-flow are based on a robust and fast design process with intensive tool support. On the other hand, this approach is not particularly well suited for high-performance systems (e.g. general purpose CPUs), due to possible performance degradation. Additionally, the concept is not best suited for applications with irregular and sporadic data-transfer between blocks.

We demonstrated that the proposed concept can be successfully applied in the area of wireless communication systems with datapath architectures. The measurements showed improvements in power consumption and a reduction of supply current variations. Therefore, this approach can be an efficient basis for complex digital system integration.

9.3 Future Work

The proposed GALS concept was successfully implemented and evaluated. However, many issues can be investigated further.

Our proposed implementation can be refined with an additional self-calibration of the local clock generators. This self-calibration can be achieved without any additional clock inputs, only by the use of the input request signal as a reference. In this way, the local clock generation and time-out period can be automatically tuned to precisely match the global clock frequency. This will lead to a PLL-like frequency generation resulting in a smoother data-flow. As a consequence, smaller FIFO buffers would be required in the system.

Furthermore, we are planning to expand the scheme by using ‘end-tokens’ to trigger emptying of internal pipeline stages. Instead of the time-out circuit proposed here, these ‘end-tokens’ can be used to start the local clock generator. This solution may be very effective. However, it would imply non-standard modifications to synchronous cores.

We need to optimize our clock jittering scheme in order to improve the possibilities for noise reduction. It is also very important to further explore theoretical limits of EMI reduction that can be reached with GALS application.

The GALS partitioning scheme has to be re-evaluated. The GALS techniques urgently need a systematic approach that can formalize the rules and algorithms for GALS partitioning. As a result of this investigation an automatic partitioning flow should be generated.

Additional work is needed to establish a complete user-friendly design framework that will allow easy integration of complex GALS blocks. Ultimately, application in a mixed-signal design will demonstrate the potential of our technique.

A subjective comparison of the GALS implementation of the OFDM baseband processor shows that the design process is easier, faster, and less error prone when compared with the equivalent synchronous design. This gives us hope that the scheme can be successfully deployed to simplify the design of datapath architectures for future mobile communication systems.

Chapter 10

References

[ALV98] Vladimir C. Alves, Felipe M. G. Franca, Edson P. Granja, A BIST scheme for asynchronous logic, *Proceedings of the Asian Test Symposium*, 1998.

[BAD04] Mustafa Badaroglu, Piet Wambacq, Geer Van der Plas, Stephane Donnay, Georges Gielen, Hugo De Man, Digital Ground Bounce Reduction by Phase Modulation of the Clock, *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04)*, Vol. I, pp. 10088, Paris, 2004

[BAR00] A. Bardsley, *Implementing Balsa Handshake Circuits,* PhD thesis, Department of Computer Science, University of Manchester, 2000.

[BLA02] B. Blaauwendraad, *TIR - Design and Testing of a Simple GALS Circuit*, Student thesis, Linköping University, 2002.

[BLU04] Ivan Blunno, Guy Alain Narboni, Claudio Passerone, An automated methodology for low electro-magnetic emissions digital circuits design, *Proceedings of the EUROMICRO Symposium on Digital System Design (DSD'04)*, pp. 540-547, Rennes, France, 31.08 – 03.09, 2004.

[BER91] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, F. Schailij, The VLSI-programming language Tangram and its translation into handshake circuits, *Proceedings of European Conference on Design Automation (EDAC),* pp. 384-389, 1991.

[BER02] K. van Berkel, F. de Beest, A. Peeters, Adding Synchronous and LSSD Modes to Asynchronous Circuits, *Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems*, pp. 161-170, April 2002.

[BOR97] David S. Bormann, Peter Y. K. Cheoung, Asynchronous Wrapper for Heterogeneous Systems, *Proceedings of International Conference on Computer Design (ICCD)*, October 1997.

[BUCK93] Joseph Buck, Edward A. Lee, The Token Flow Model, Presented at *Data Flow Workshop,* Hamilton Island, Australia May, 1992, also in *Advanced Topics in Dataflow Computing and Multi-threading,* ed. Lubomir Bic, Guang, Gao, and Jean-Luc Gaudiot, IEEE Computer Society Press, 1993.

[CAR03] Carlsson, W. Li, K. Palmkvist, L. Wanhammar, S. Zhuang, A Design Path for Design of GALS Based Communication Systems, *Proceedings of Swedish System-on-Chip Conference*, Eskilstuna, Sweden, April 8-9, 2003.

[CHAP84] Daniel M. Chapiro, *Globally-Asynchronous Locally-Synchronous Systems*, PhD thesis, Stanford University, October 1984.

[CHAN99] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, *Surviving the SoC Revolution*, Kluwer Academic Publishers, 1999.

[CHAK03] Ajanta Chakraborty, Mark Greenstreet, Efficient Self-Timed Interfaces for Crossing Clock Domains, *Proceedings of 9th International Symposium on Asynchronous Circuits and Systems (ASYNC'2003)*, pp. 78-88, Vancouver, Canada, 2003.

[CHE00a] Tiberiu Chelcea, Steven M. Nowick, Low-latency asynchronous FIFO's using token rings, *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 210-220, April 2000.

[CHE00b] Tiberiu Chelcea, Steven M. Nowick, Low Latency FIFO for Mixed-Clock Systems, *Proceedings of the IEEE Computer Society Workshop on VLSI (IWLSI'00)*, pp. 119-126, 2000.

[CHE01] Tiberiu Chelcea, Steven M. Nowick, Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols, *Proceedings of ACM/IEEE Design Automation Conference*, pp. 21-26, Las Vegas, USA, June 2001.

[COR96] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers, *Proceedings of XI Conference on Design of Integrated Circuits and Systems*, Barcelona, November 1996.

[DAM02] Ø. Damhaug, T. Njølstad, Arbitration And Meta-Stability Management In Globally Asynchronous Locally Synchronous Circuits, *Proceedings of 5th Nordic Signal Processing Symposium*, On board Hurtigruten from Tromsø to Trondheim, Norway, 2002.

[DIKE99] C. Dike and E. Burton, "Miller and Noise Effects in a Synchronizing Flip-Flop," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 849-855, 1999.

[DIN02] A. V. Dinh Duc, J. Rigaud, A. Rezzag, A. Siriani, F. Fragoso, L. Fesquet, M. Renaudin, TAST CAD tools*, Collection of regular and poster presentations at the 2nd Asynchronous Circuit Design Workshop ACiD 2002*, Munich, Germany, 28-29 January, 2002.

[DOB04] R. Dobkin, R. Ginosar, C. Sotiriu, Data Synchronization Issues in GALS SoCs, *Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems,* pp. 170-179, Crete, Greece, 19 - 23 April 2004.

[FRA04] U. Frank, R. Ginosar, A Predictive Synchronizer for Periodic Clock Domains, *In Proceedings of International Workshop on Power And Timing Modeling Optimization and Simulation (PATMOS),* pp. 402-412, Santorini, Greece, September 2004.

[FUH01] R. Fuhrer, S. Nowick, *Sequential Optimization of Asynchronous and Synchronous Finite State Machines: Algorithms and Tools*, Kluwer Academic Publishers, Boston, 2001.

[GEN92] H.J. Genrich, R.M. Shapiro. Formal Verification of an Arbiter Cascade, *In Jensen, K., editor, Proceedings of 13th International Conference on Application and Theory of Petri Nets (ICATPN'92)*, Sheffield, UK, Springer LNCS, Vol. 616, June 1992.

[GIN02] Ran Ginosar, Application of the Async design to the Sync world: synchronization and arbitration, *Presentation at Summer School on "Asynchronous Circuit Design",* Grenoble, France, July 2002.

[GIN03] Ran Ginosar, Fourteen Ways to Fool Your Synchronizer, *Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems,* pp. 89-96, May 2003.

[GRA01] Eckhard Grass, Klaus Tittelbach-Helmrich, Ulrich Jagdhold, Alfonso Troya, Gunther Lippert, Olaf Krüger, Jens Lehmann, Koushik Maharatna, Kai Dombrowski, Norbert Fiebig, Rolf Kraemer, Petri Mähönen, On the Single-Chip Implementation of a Hiperlan/2 and IEEE 802.11a Capable Modem, *IEEE Personal Communications*, Vol. 8, No. 6, pp. 48 – 57, December 2001.

[GRA05] E. Grass, F. Winkler, M. Krstić, A. Julius, C. Stahl, M. Piz, Enhanced GALS Techniques for Datapath Applications, *Proceedings of International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS)*, LNCS 3728, Springer Verlag, pp. 581-590, Leuven, Belgium, 2005.

[GUR02] Frank Gürkaynak, Thomas Villiger, Stephan Oetiker, Norbert Felber, Hubert Kaeslin, Wolfgang Fichtner, A Functional Test Methodology for Globally-Asynchronous Locally-Synchronous Systems, *Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems*, pp. 181-189, April 2002.

[HAN05] *Handshake Solutions web page*, http://www.handshakesolutions.com/.

[HEM99] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olson, P. Nilsson, J. Öberg, P. Ellervee, D. Lindqvist, Lowering power consumption in clock by using globally asynchronous locally synchronous design style, *Proceedings of ACM/IEEE Design Automation Conference, 1999.*

[HEAT03] M. Heath, I. Harris, A Deterministic Globally Asynchronous Locally Synchronous Microprocessor Architecture, *Proceedings of 4th International Workshop on Microprocessor Test and Verification (MTV)*, Austin, TX, May 29-30, 2003.

[HEAL04] James T. Healy, Shotgun Wedding Needed: BIST and ATE, *Chip Scale Review*, pp. 11-13, Jan/Feb 2004.

[IEEE99] IEEE P802.11a/D7.0, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High Speed Physical Layer in the 5 GHz Band*, July 1999.

[ITRS03] *International Technology Roadmap for Semiconductors*, 2003.

[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.

[KES97] J. Kessels, P. Marston, Designing asynchronous standby circuits for a low power pager, *Proceedings of the International Symposium of Advanced Research in Asynchronous Circuits and Systems*, pp 268-278, April 1997.

[KOL98] Rakefet Kol, Ran Ginosar, Adaptive Synchronization for Multi-Synchronous Systems, *Proceedings of International Conference on Computer Design (ICCD)*, October 1998.

[KON01] Xiaohua Kong, Radu Negulescu, and Larry Weidong Ying. Refinement-based formal verification of asynchronous wrappers for independently clocked domains in systems on chip, *Proceedings 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME)*, September 2001.

[KRS03a] M. Krstić, E. Grass, New GALS Technique for Datapath Architectures, *Proceedings of International Workshop on Power And Timing Modeling Optimization and Simulation (PATMOS)*, pp. 161-170, Turin, Italy, September 2003.

[KRS03b] M. Krstić, K. Maharatna, A. Troya, E. Grass, U. Jagdhold, *Implementation of the IEEE 802.11A Compliant Low-Power Baseband Processor*, *Proceedings of 6th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, Nis, Serbia, Vol I, pp. 97-100, 2003.

[KRS03c] M. Krstić, E. Grass, Request-driven GALS Technique for Datapath Architectures, *Collection of regular and poster presentation of ACiD-WG Workshop*, Crete, Greece, January 2003.

[KRS03d] M. Krstić, A. Troya, K. Maharatna, E. Grass, Optimized Low-Power Synchronizer Design for the IEEE 802.11a Standard, *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP),* Vol II, pp. 333-336, Hong Kong, April 6-10, 2003.

[KRS04a] M. Krstić, E. Grass, GALSification of IEEE 802.11a Baseband Processor, *Proceedings of International Workshop on Power And Timing Modeling Optimization and Simulation (PATMOS)*, pp. 258-267, Santorini, Greece, September 2003.

[KRS04b] M. Krstić, E. Grass, GALS Baseband Processor for WLAN, *Collection of regular and poster presentation of ACiD-WG Workshop*, Turku, Finland, June 2004.

[KRS05a] M. Krstić, E. Grass, C. Stahl, Request-driven GALS Technique for Wireless Communication System, *Proceedings of 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC 2005)*, pp. 76-85, New York, Mar 2005.

[KRS05b] M. Krstić, E. Grass, BIST Technique for GALS Systems, *Proceedings - 8th EUROMICRO Conference on Digital System Design (DSD 2005)- Architectures, Methods and Tools*, Porto, Portugal, pp. 10-16, August 30th - September 3rd, 2005.

[MAH05] K. Maharatna, S. Banerjee, E. Grass, M. Krstić, A. Troya, Modified Virtually Scaling Free Adaptive CORDIC Rotator Algorithm and Architecture, *IEEE Trans. on Circuits and Systems for Video Technology (CSVT)*, Vol. 15, No. 11, November 2005, pp. 1463-1474.

[MED02] *MEDEA+ Design Automation Roadmap*, March 2002.

[MEI99] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Öberg, T. Olson, P. Nilsson, D. Lindqvist, H. Tenhunen, Evaluating Benefits of Globally Asynchronous Locally Synchronous VLSI architecture, *Proceedings of International Symposium on Circuits and Systems, 1999.*

[MIG96] V. Mignone, A. Morello, CD3-OFDM: A Novel Demodulation Scheme for Fixed and Mobile Receivers, *IEEE Transactions on Communications*, vol. 44, no. 9, pp. 1144-1151, Sept. 1996.

[MOO98] Simon Moore, Peter Robinson, Rapid prototyping of self-timed circuits, *Proceedings of International Conference on Computer Design (ICCD)*, pages 360-365, October 1998.

[MOO00] Simon Moore, George Taylor, Robert Mullins, Paul Cunningham, Peter Robinson, Self Calibrating Clocks for Globally Asynchronous Locally Synchronous System, *Proceedings of International Conference on Computer Design (ICCD)*, September 2000.

[MOO02] Simon Moore, George Taylor, Robert Mullins, Peter Robinson, Point to Point GALS interconnect, *Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems*, pp. 69-75, April 2002.

[MUT99] Jens Muttersbach, Thomas Williger, Hubert Kaeslin, Norbert Felber, Wolfgang Fichtner, Globally-Asynchronous Locally-Synchronous Architectures to Simplify the Design of On-Chip Systems, *Proceedings of 12th IEEE International ASIC/SOC Conference*, Washington DC, pp. 317-321, Sept. 1999.

[MUT01] Jens Muttersbach, *Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems*, Doctor of Technical Sciences Dissertation, ETH Zurich, Switzerland, 2001.

[NOW02] S. Nowick, T. Chelcea, *Minimalist CAD Tool Tutorial*, Summer School on “Asynchronous Circuit Design”, Grenoble, France, July 15-19, 2002

[PRI70] Price, J.E., A new look at yield of integrated circuits, *Proc. IEEE*, 58, 1290, 1970.

[SCHM00] K. Schmidt. Lola -- a low level analyser. *In Nielsen, M. and Simpson, D., editors, International Conference on Application and Theory of Petri Nets*, LNCS 1825, pp. 465 ff. Springer-Verlag, 2000.

[SCHW01] L. Schwoerer, H. Wirz, VLSI Implementation of IEEE 802.11a Physical Layer, *Proceedings of 6th Int'l. OFDM Workshop*, Hamburg, Germany, pp. 28.1 – 28.4, 2001.

[SEITZ80] Charles Seitz, System Timing. *In Carver A. Mead and Lyn Conway, editors*, *Introduction to VLSI systems*, chapter 7, Addison-Wesley, 1980.

[SEIZ94] Jakov Seizovic, Pipeline Synchronization, *Proceedings of the International Symposium of Advanced Research in Asynchronous Circuits and Systems*, pp 87-96, Nov 1994.

[STA05] C. Stahl, W. Reisig, M. Krstić, Hazard Detection in a GALS Wrapper: a Case study, *In Desel, J. and Watanabe, Y., editors, 5th International Conference on Application of Concurrency to System Design (ACSD'05),* pages 234-243, IEEE Computer Society, 2005.

[TAL05] Emil Talpes, Diana Marculescu, Toward a Multiple Clock/Voltage Island Design Style for Power-Aware Processors, *IEEE Transactions on Very Large Scale Integrations (VLSI) Systems*, Vol. 13, No. 5, pp. 591-603, May 2005.

[TAY00] George Taylor, Simon Moore, Steve Wilcox, Peter Robinson, An on-chip dynamically recalibrated delay line for embedded self-timed systems, *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 45-51, April 2000.

[TRO02] A. Troya, K. Maharatna, M. Krstić, E. Grass, R. Kraemer, OFDM Synchronizer Implementation for an IEEE802.11a Compliant Modem, *Proceeding of IASTED International Conference Wireless and Optical Communications*, Banff, Canada, ISBN 0-88986-344-X, pp. 152-157, 2002.

[TRO03] A. Troya, M. Krstić, K. Maharatna, Simplified Residual Phase Correction Mechanism for the IEEE 802.11a Standard, *Proceedings of IEEE VTC Conference*, Orlando, USA, 2003.

[YUN96] Kenneth Y. Yun, Ryan P. Donohue, Pausible Clocking: A first step toward heterogeneous systems, *Proceedings of International Conference on Computer Design (ICCD)*, October 1996.

[YUN99a] Kenneth Yun, David Dill, Automatic synthesis of extended burst-mode circuits: Part I and II, *IEEE Transactions on Computer-Aided Design*, 18(2), pp. 101-132, February 1999.

[YUN99b] K. Y. Yun and A. E. Dooply, Pausible clocking based heterogeneous systems, *IEEE Transactions on VLSI Systems*, Vol. 7, No. 4, pp. 482-487, Dec. 1999.

[VIL02] T. Villiger, F. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, W. Fichtner, Multi-point Interconnect for Globally-Asynchronous Locally-Synchronous Systems, *Collection of regular and poster presentations at the 2nd Asynchronous Circuit Design Workshop ACiD 2002*, Munich, Germany.

[VIL03] 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.

[ZHU02a] S. Zhuang, W. Li, J. Carlsson, K. Palmkvist, L. Wanhammar, An Asynchronous Wrapper with Novel Handshake Circuits for GALS Systems, *Proceedings of International Conference on Communications, Circuits and Systems (ICCCAS)*, Chuengdu, China, 2002.

[ZHU02b] S. Zhuang, W. Li, J. Carlsson, K. Palmkvist, L. Wanhammar, Asynchronous Data Communication with Low Power for GALS Systems, *Proceedings of IEEE International Conference on Electronics, Circuits, and Systems (ICECS) 2002,* Dubrovnik, Croatia, 2002.

[ZHU02c] S. Zhuang, W. Li, J. Carlsson, K. Palmkvist, and L. Wanhammar, The VLSI Implementation of 1-D DWT Based On GALS Systems, *Proceedings of IEEE Nordic Signal Processing Symposium,* Hurtigruta, Tromsö-Trondheim, Norway, Oct. 4-7, 2002.

Acronyms and Symbols

3D	-	tool for synthesis of hazard-free asynchronous controllers
3DC	-	tool that converts the output of the 3D tool into VHDL
4G	-	fourth generation of mobile communications
Ack	-	acknowledge line of the handshake protocol
ACKC	-	gated internal acknowledge signal
ACKEN	-	acknowledge enable signal
ACK_A	-	input acknowledge signal
ACK_B	-	output acknowledge signal
ACK_INT	-	internal acknowledge signal
ACKI_1	-	clock stretching acknowledge
ADC	**-**	Analog to Digital Converter
AFSM	**-**	Asynchronous Finite State Machine
AFE	**-**	analog front-end
ASIC	**-**	Application Specific Integrated Circuit
ATPG	-	Automatic Test Pattern Generation
BIST	**-**	Built-In Self-Test
BPSK	-	Binary Phase Shift Keying
CBC	-	Central BIST Controller

CC	-	coarse control signal for delay element
Clk_grant	-	enable for new clock cycle generation
clk	-	clock signal
CMOS	-	Complementary MOSFET
CMU	-	Clock Management Unit
CORDIC	-	COordinate Rotation DIgital Computer
DAC	-	Digital to Analog Converter
DATAV_IN	-	valid data present at the input stage of the LC block
DATAV_OUT	-	synchronous data valid signal
DFT	-	Design for Testability
DLE	-	data latch enable signal
DOV	-	data output valid signal
DONV	-	data output not valid signal
DSP	-	Digital Signal Processing
ECLK	-	external clock signal
EDA	-	Electronic Design Automation
EJTAG	-	Enhance Joint Test Action Group
EMI	-	Electro-Magnetic Interference
EN	-	enable for join circuitry
f_c	-	clock frequency
FF	-	flip-flop
FFT	-	Fast Fourier Transform
FIFO	-	First-In First-Out memory
FIFO_TA	-	FIFO memory used for token alignment in the receiver
FIR	-	Finite Impulse Response
FSM	-	Finite State Machine
GALS	-	Globally Asynchronous Locally Synchronous
GALS DF	-	GALS Design Framework
GUI	-	Graphical User Interface

HDL - Hardware Description Language

IEEE 802.11a - IEEE standard for Wireless LAN

INT_CLK - Clock signal of the LS block

IP - Intellectual Property

IFFT - Inverse Fast Fourier Transform

LAN - Local Area Network

LCLKM - locally generated clock signal

LCLK - output of the ring oscillator

LFSR - Linear Feedback Shift Register

LS - Locally Synchronous

ME - Mutual Exclusion element

Msps - Mega samples per second

MTBF - Mean Time Between Failures

MUTEX - Mutual Exclusion element

NCO - Numerically Controlled Oscillator

OFDM - Orthogonal Frequency Division Multiplex

pass/fail - signal that indicates the result of the test

PCC - Pausable Clocking Control

ppm - part per million

PTCG - Pausable Test Clock Generator

PVT - process, voltage, temperature

QAM - Quadrature amplitude modulation

QPSK - Quadrature phase shift keying

RCLK - inverted local clock signal

RCLKD - delayed inverted local clock signal

Req - request line of the handshake protocol

REQ_A - input request of the asynchronous wrapper

REQ_A1 - hazard-free derivation of the input request

REQ_A - output request of the asynchronous wrapper

REQ_INT	-	internal request line in the asynchronous wrapper
REQI_1	-	request for clock stretching generated from input port
Rot_val_ok	-	control signal that indicates coarse synchronisation
RST	-	time-out circuitry reset
RTL	-	Register-transfer level
Rx_TRA	-	token rate adaptation circuit of the receiver
SDF	-	standard delay format
SoC	-	System on Chip
ST	-	time-out indication signal
STG	-	Signal Transition Graph
Start_m	-	activation signal for receiver feedback loop
STOPI	-	internal stop signal
STOP	-	signal that indicates clock generator disabling
Stretch	-	signal that controls stretching of the pausable clock generator
T	-	clock period
Test_on/off	-	activation signal for BIST
Test_reset	-	reset for test circuits
Test_select	-	selection signal for the specific BIST test
TDE	-	Test Data Evaluator
TDEi_en	-	enable of *i*-th Test Data Evaluator
TDEi_golden_select	-	selection of the golden value for *i*-th TDE
$\mathbf{T_{time_out}}$	-	timing period of request line inactivity that activates local clock
TPG	-	Test Pattern Generator
TPGi_en	-	enable of *i*-th Test Pattern Generator
VCD	-	Value Change Dump file format
VHDL	-	Very high speed integrated circuits Hardware Description Language
WLAN	-	Wireless Local Area Network
Write_fifo	-	data output valid of the receiver
XBM	-	Extended Burst Mode

Curriculum Vitae

Miloš Krstić received his Dipl.-Ing. degree in Electronics & Communications, and Master of Science (M.Sc.) degree in Electronics from the University of Niš, Serbia and Montenegro, in 1997 and 2001, respectively. In 2001 he joined the IHP GmbH in Frankfurt (Oder) as a Research Associate in the Wireless Communication Systems department. For the last few years in IHP, his work was mainly focused on low power digital design for wireless applications and Globally Asynchronous Locally Synchronous (GALS) methodologies for large digital system integration. The developed concepts and the respective results are presented in this thesis.