

# System integration by request-driven GALS design

M. Krstić, E. Grass, C. Stahl and M. Piz

**Abstract:** A novel request-driven globally asynchronous locally synchronous (GALS) technique for the system integration of complex digital blocks is proposed. For this new GALS technique, an asynchronous wrapper compliant is developed and evaluated. This proposed GALS technique is applied to a baseband processor compatible with the wireless LAN standard IEEE 802.11a. The developed GALS baseband processor chip is fabricated and measured. Besides improvements of the system integration process, a 5 dB reduction in electromagnetic interference, 30% reduction in instantaneous supply current variation, and similar dynamic power consumption as in the synchronous baseband processor is achieved.

## 1 Introduction

Modern mobile communication devices contain highly complex digital structures. An ever-increasing system complexity is leading to growing challenges in the area of system integration, power management and noise reduction. In order to cope with these problems, different methods have been proposed. One of the most promising techniques at present is the globally asynchronous locally synchronous (GALS) methodology for system integration.

The GALS methodology was first proposed over twenty years ago by Chapiro [1], and in the last few years has regained in popularity with several new methods being developed. A GALS circuit consists of several modules. Each module works synchronously and is surrounded by an asynchronous wrapper (AW) that has an asynchronous interface to other modules. Together, a wrapper and its enclosed module form a GALS block. Most GALS proposals are based on a pausable clocking scheme as can be seen in the works of Muttersbach [2], Moore *et al.* [3], and Bormann and Cheung [4]. Those GALS proposals offer a stable framework for system integration and some of them have been successfully demonstrated on silicon. However, the proposed methods are oriented towards a general system architecture and are not optimised for particular applications. Additionally, they have not been optimised for power saving, electromagnetic interference (EMI) reduction and bursty data transfer. Consequently, there is scope for a novel GALS proposal that supports particular applications better than the previous techniques.

Our architectures of interest are systems with very complex digital circuitry, and one of our main aims was to improve the integration of complex blocks and ease timing closure. A GALS structure that allowed data transfer at every cycle of the local clock was also a consideration. For mobile applications, lowering power consumption is a

critical requirement, and along with the lowering of EMI, which is of great importance for mixed-signal applications, is also focussed upon in this work.

## 2 Request-driven GALS technique

### 2.1 Concept of the novel GALS methodology

GALS techniques are about to be adopted into standard design practice. However, most are not optimised for special system architectures. We have targeted GALS applications in point-to-point (datapath) architectures, and have assumed that in our applications, the data transfer is very intensive (with every clock cycle) but bursty. As a consequence, data bursts are followed by long periods of communication inactivity.

As proposed by Krstić and Grass [5] and Krstić *et al.* [6], GALS blocks can be clocked for such systems by the incoming request signals when receiving a data burst. Consequently, in this request-driven mode, synchronisation is not needed and the clock frequency of the receiving (slave) GALS block will be the same as the clock frequency of the transmitting (master) GALS block. The only difference will be a different clock phase due to the delays on the request line. When a complete burst is received, some more cycles are usually needed to process and transfer the data to the succeeding GALS block. In order to perform those tasks, the clock generation is handed over to a local clock generator. In order to detect the inactivity of the input handshake line, a time-out generator is needed. This functional block starts the local clock generation after a complete burst has been received. This mode is called a local clock generation mode. After processing and transferring all tokens, the clock activity of the module needs to be stopped. This will then save the power consumed by unnecessary clock transitions. By 'token', we mean a single data-item, transferred between two GALS blocks, and accompanied by the corresponding handshake signals.

### 2.2 Potential gain of the novel architecture

As a result of the transformation of the synchronous system into GALS (i.e. GALSification), we expected improvements in several areas. Our primary aim was to achieve an easier and simpler system integration of complex digital blocks. For example, a global clock-tree generation would not

© The Institution of Engineering and Technology 2006

*IEE Proceedings* online no. 20050210

doi:10.1049/ip-cdt:20050210

Paper first received 16th December 2005 and in revised form 2nd June 2006

M. Krstić, E. Grass and M. Piz are with IHP, Im Technologiepark 25, Frankfurt (Oder) 15236, Germany

C. Stahl is with the Humboldt-Universität zu Berlin, Rudower Chaussee 25, Berlin 12489, Germany

E-mail: krstic@ihp-microelectronics.com

longer be needed, the timing closure between blocks would be relaxed and clock skew in general would be reduced. As a result of these enhancements, we expect a significantly reduced time-to-market and design costs.

The request-driven GALS technique provides a power-saving mechanism similar to clock gating. When the triggering of the local block is not needed, clock generation is disabled. This feature is inherent to this GALS technique, and it is not needed to generate any additional circuitry in order to save power. However, the limits of this low-power approach are similar to those of clock gating.

In order to estimate the limits for EMI reduction with the request-driven GALS, we performed the investigation described here. The analysis was based on the provisional numbers and assumptions of a system that was similar to our baseband processor. The purpose of this investigation was to identify potential EMI reductions. We used a triangular shape as a model for the instantaneous current consumption, as described by Badaroglu *et al.* [7] and Blunno *et al.* [8]. The current profile of a synchronous system was modelled as a triangular shape with a frequency of 50 MHz and a peak of 1 A. It was assumed that the profile was asymmetric, with a rise time of 5 ns and a fall time of 10 ns. For comparison, the supply current profile after the GALSification of this synchronous system was modelled. It was assumed that the synchronous circuit was split into 10 GALS blocks with the peak consumption of 200 mA (two blocks), 100 mA (four blocks) and 50 mA (four blocks). In order to model a request-driven GALS system, we applied random phases to different GALS blocks and also introduced a frequency offset of  $\pm 2\%$ . This frequency offset was taken as a typical value for the frequency resolution of our pausable clock in the AW. Fig. 1a illustrates the current spectrum for the pure synchronous design, and Fig. 1b the request-driven GALS design. The figure also shows that the application of the request-driven GALS technique leads to a spectral noise suppression of up to 20 dB. Additionally, in the time domain, the current peaks were reduced by 30–50%. That indicated that this GALS technique could be successfully used for EMI reduction.

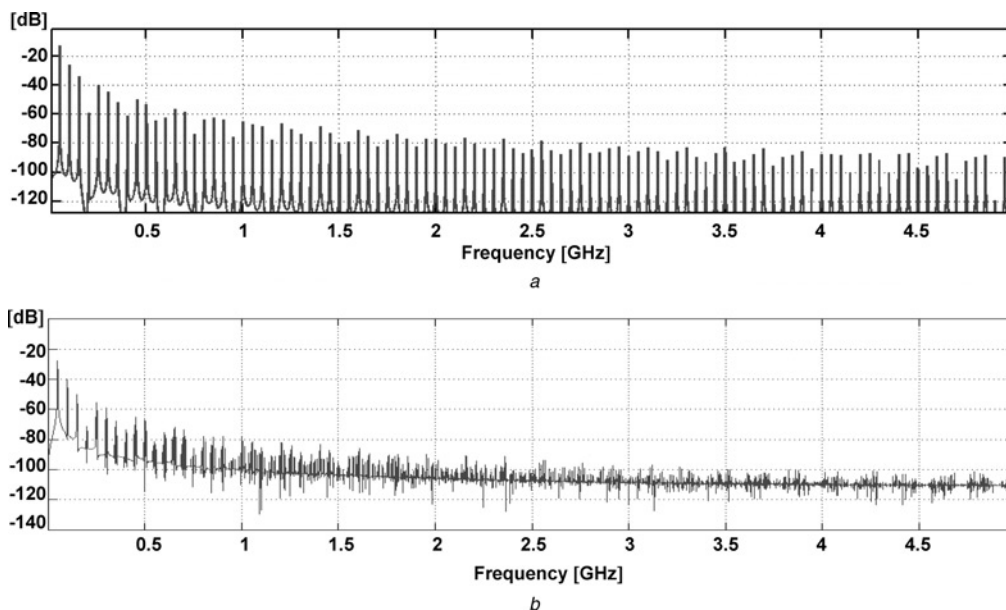
Our request-driven GALS technique has a few advantages when compared with other existing GALS proposals.

For instance, we can perform a data transfer between GALS blocks at every clock cycle of the local clock. The timing determinism is increased in comparison with completely locally driven GALS architectures as in the works of Muttersbach [2], Moore *et al.* [3] and Bormann and Cheung [4]. Synchronisation and arbitration is performed only at a single point in the handshake chain (output port of the AW) and not at several places (input and output port in other solutions). In request-driven mode, the clock speed of the local module is determined by the speed of communication of the master. Therefore this can be very useful to increase the performance. The design style and the interface to the locally synchronous (LS) module is simple and easily adoptable by synchronous designers. However, there are some limitations as well. The implementation of the wrapper is rather complex and the maximum throughput is not very high. Additionally, further delays can be generated in the system when non-bursty data are applied.

Consequently, the application fields for our technique are datapath architectures with moderate performance and intensive data transfer between blocks. In contrast, it is hard to imagine an application of this technique in the area of high-performance CPUs or systolic arrays.

### 3 Implementation of the GALS wrapper

The concept, described in the previous section, can be implemented in many different ways. In the following, one possible implementation is presented. Fig. 2 shows the block diagram of the AW compliant with the proposed request-driven concept. The proposed AW consists of several components including the input and output ports, the clock-control block, the time-out generator and the tunable clock generator. Additionally, a data latch is placed in the dataflow in order to prevent metastability on the register stages of the LS modules. Three handshake processes are performed in a single AW. One is performed by the input port that is connected to the predecessor GALS block. This handshake provides for safe data reception. Furthermore, an internal handshake is performed between the input and output ports. It is used for the clock generation in the request-driven mode. The third handshake is carried



**Fig. 1** Supply voltage spectrum

a Synchronous

b GALS baseband processor

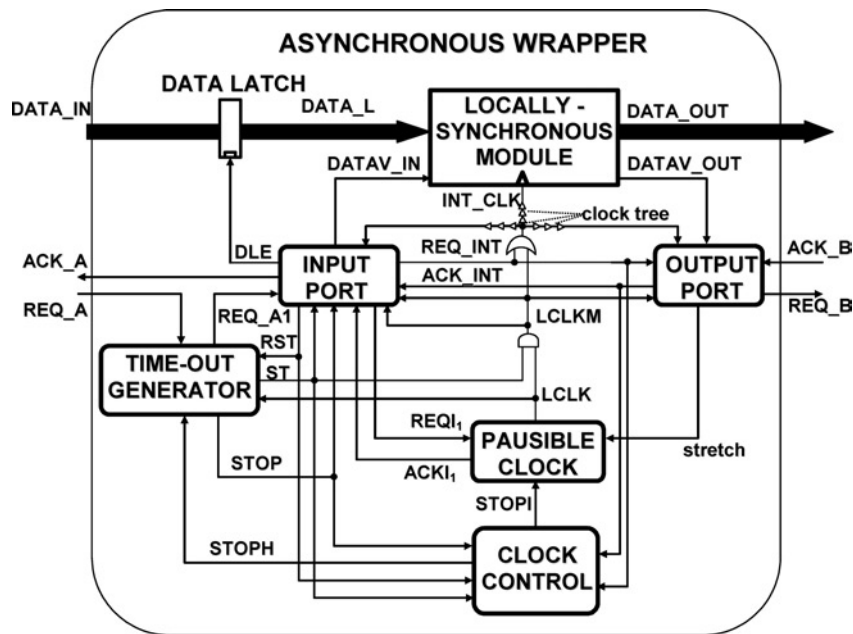


Fig. 2 Structure of the AW

out by the output port. It has to guarantee safe transfer of the output data. The following interface signals support the operation of the wrapper. On the input port, handshake control signals (REQ\_A and ACK\_A) adhering to the broad four-phase protocol [9] correspond to the data signal DATA\_IN; on the output side, the signal DATA\_OUT carries the data and the respective control signals are REQ\_B and ACK\_B. In addition, our wrapper provides the data (DATA\_L) and data validity (DATAV\_IN) signals for the LS block. At the output port, control signal DATAV\_OUT is generated. This signal indicates the data validity for the subsequent and not the current clock cycle of the synchronous block.

Although the request-driven concept is invented for point-to-point interfaces between blocks, a more complex dataflow architecture can be constructed. Standard asynchronous elements such as ‘joins’ and ‘forks’ can be employed. 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 because of 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. Similarly, the output port generates a single data stream. Therefore if the output data stream has to be distributed to several sinks, a fork construct must be implemented. It is possible to adapt the proposed wrapper architecture in order to support multi-output-port structures. However, in our opinion, it is better to deploy single-output wrappers. If multiple inputs or outputs are required, standard asynchronous joins and forks, respectively, can be used as separate components.

### 3.1 Pausable clock generator and clock-control block

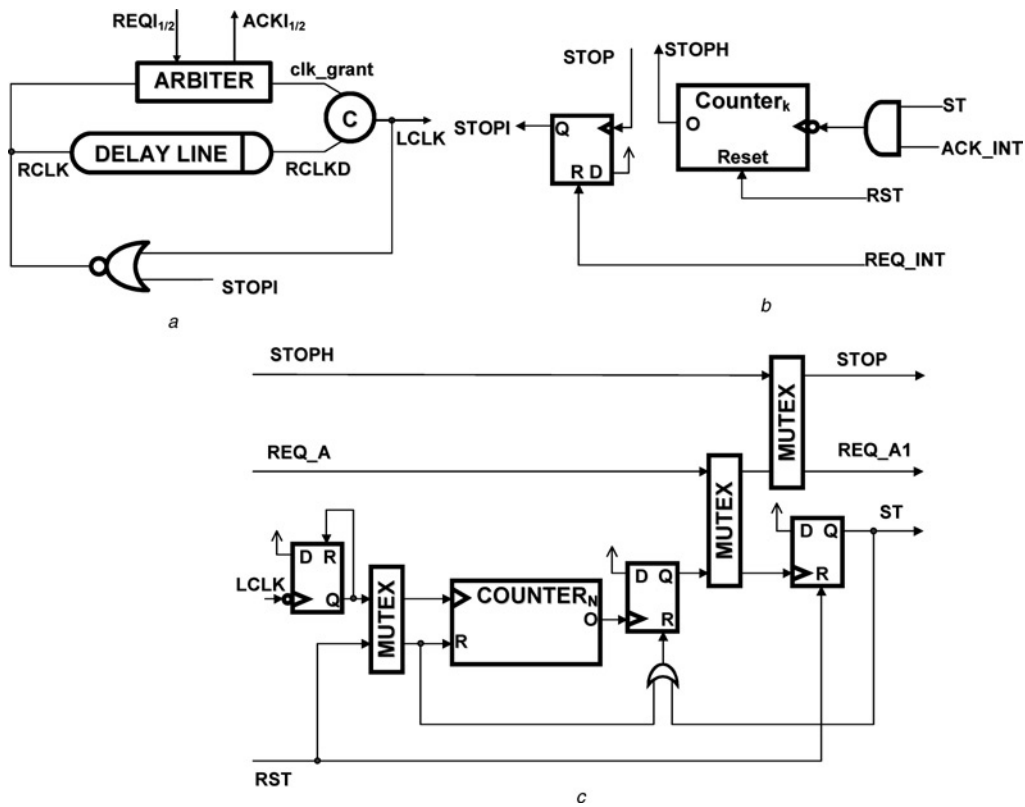
Fig. 3a gives the structure of the pausable clock generator. The architecture of this block is the same as proposed in the works of Muttersbach [2]. The pausable clock generator includes arbitration units with two mutual exclusion (mutex) elements and a tunable ring oscillator. Such an architecture allows simple stretching of the clock signal.

The purpose of the clock-control block, depicted in Fig. 3b, is to stop the clock when all internally stored

tokens are processed and transferred to the succeeding GALS block. The clock-control block mainly contains a counter that counts the number of valid clock cycles. When the number of counted clock cycles is equal to the one needed for emptying the local pipeline, the clock activity has to be stopped. As a result, the signal STOPH is activated. Because signals STOPH and REQ\_A are concurrent, they have to be arbitrated in order to avoid a possible deadlock in the input port of the AW. After arbitration, signal STOP is fed back to the clock-control block. Finally, signal STOP activates signal STOPI that directly controls the ring oscillator. For some systems, it is possible that the duration of their activity depends on the data and accordingly, the clock control cannot be hardwired. For such cases, it is feasible to make a modification where the clock-control unit is a part of LS the block and it is possible to dynamically change the number of cycles sufficient to process some data stream. Alternatively, it is possible to set the number of local clock cycles to the maximal number needed, independent of the data content.

### 3.2 Time-out generator

The time-out generator, illustrated in Fig. 3c, has two basic functions. The first task is to generate the time-out signal ST. This signal has to be generated if, for some period of time, there is no activity on the input handshake lines. This time can be specified by setting a counter inside the time-out generator. The counter counts the number of locally generated clock cycles without the activity of the handshake. In order to safely reset this counter whenever some handshake is performed, we had to put a flip-flop in front of the clock input of the counter. This flip-flop will transform possibly longer periods of active high into short pulses. Short pulses on the clock line result in reliable propagation of the RST signal through the mutex element. However, the design of such pulse generator has to be carefully verified to avoid timing violations. The second task of the time-out generator is to arbitrate the concurrent signals REQ\_A and ST, and REQ\_A and STOP in order to avoid hazards in the wrapper. Therefore some mutex elements are implemented in this block.

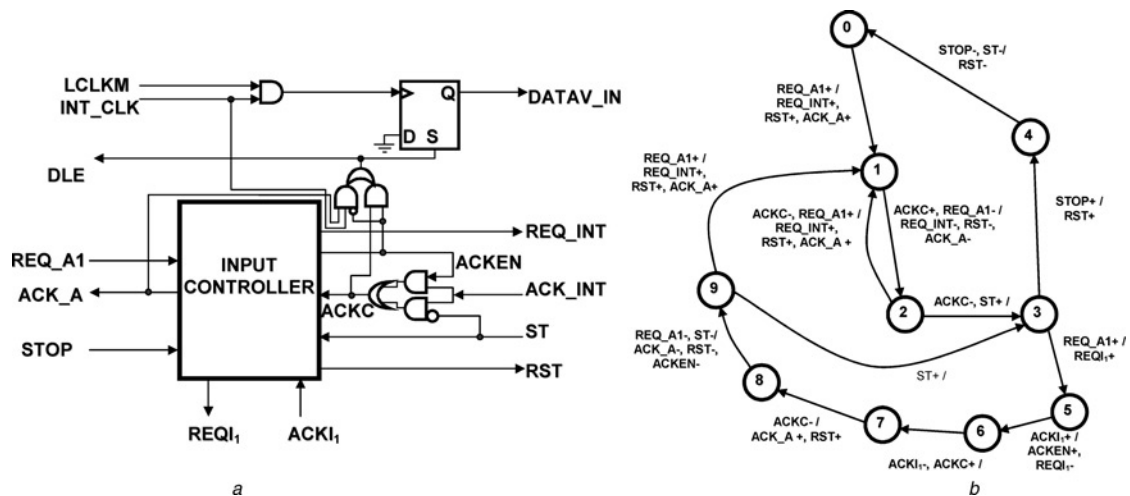


**Fig. 3** *AW components*  
 a Plausible clock generator  
 b Clock-control block  
 c Time-out generator

### 3.3 Input port

Fig. 4a depicts the structure of the input port including the input controller together with supporting circuitry. The function of the input controller is to receive data coming from the preceding GALS block. Additionally, in request-driven mode, the input port generates the local clock. In Fig. 4b, the asynchronous finite state machine (AFSM) specification of the input controller is given. From this figure, we can define the following operation modes of the input port. Firstly, states 1 and 2 support the request-driven mode. In this mode, the input port handles

the handshake protocol. State 3 covers the time-out mode and the local clock generation. In this mode, the input port has a passive role. There are two possible state transitions from the time-out mode. Either the signal STOP will indicate an empty pipeline; as a result, the GALS block will go into idle mode, waiting for a new input request. Or, a new request signal appears and then the transitional mode is initiated (state changes  $3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$ ). In this mode, the current local clock will be processed first and then the control of the clock will be handed over to the request signal. The synthesis of the input port AFSM generates the following



**Fig. 4** *Input port*  
 a Structure  
 b Input controller AFSM specification

equations

$$\begin{aligned}
 \text{REQ\_INT} &= \text{REQ\_A1} \cdot \text{REQ\_INT} + \overline{\text{ACKC}} \cdot \text{REQ\_INT} \\
 &\quad + \text{REQ\_A1} \cdot \overline{\text{ACKC}} \cdot \text{ACKEN} \cdot \overline{\text{ST}} \\
 \text{ACK\_A} &= \overline{\text{ACKC}} \cdot \text{REQ\_INT} + \text{REQ\_A1} \cdot \text{RST} \\
 &\quad + \overline{\text{ACKC}} \cdot \text{ST} \cdot \overline{\text{ACKI1}} \cdot \text{ACKEN} \cdot \overline{\text{Z0}} \\
 &\quad + \text{REQ\_A1} \cdot \overline{\text{ACKC}} \cdot \overline{\text{ST}} \cdot \overline{\text{ACKEN}} \\
 \text{ACKEN} &= \text{ACKI1} + \text{REQ\_A1} \cdot \text{ACKEN} + \text{ST} \cdot \text{ACKEN} \\
 \text{RST} &= \text{STOP} + \overline{\text{ACKC}} \cdot \text{REQ\_INT} + \text{REQ\_A1} \cdot \text{RST} \\
 &\quad + \text{ST} \cdot \text{RST} + \overline{\text{ACKC}} \cdot \text{ST} \cdot \overline{\text{ACKI1}} \cdot \text{ACKEN} \\
 &\quad \times \overline{\text{Z0}} + \text{REQ\_A1} \cdot \overline{\text{ACKC}} \cdot \overline{\text{ST}} \cdot \overline{\text{ACKEN}} \\
 \text{REQ\_A1} &= \text{REQ\_A1} \cdot \overline{\text{ACKC}} \cdot \text{ST} \cdot \text{ACKEN} \\
 \text{Z0} &= \text{ACKI1} + \overline{\text{REQ\_A1}} \cdot \text{ACKC} + \overline{\text{REQ\_A1}} \cdot \overline{\text{ST}} \\
 &\quad \times \text{Z0} + \overline{\text{ACKC}} \cdot \text{ACKEN} \cdot \text{Z0} + \text{ACKC} \\
 &\quad \times \overline{\text{ACKEN}} \cdot \text{Z0}
 \end{aligned}$$

where Z0 is an internal signal, added to provide hazard-free operation of this AFSM.

Additional gates shown in Fig. 4a are used to perform the controller signal changes according to the burst mode specification [9] and to generate the signal for data-latch activation. In principle, the purpose of these gates is to block unwanted signal transitions that are not according to the expected AFSM protocol. As a result, the AFSM is simplified by limiting the number of state transitions. Otherwise, this AFSM would be too slow and too complex.

### 3.4 Output port

Fig. 5a presents the block diagram of the output port and in Fig. 5b, the AFSM specification of the output controller is shown. The output port has to support the delivery of data from the GALS block. In general, the AFSM for the output controller has a simple structure (Fig. 5b). In states 1 and 2, the output controller performs the internal and output handshake. This mode is activated when there is data to be transferred to the succeeding GALS block (indicated with data valid signal DOV). When there is no valid output token (indicated with activation of signal DONV),

the output controller just passively responds to the internal handshake. According to Fig. 5a, it is obvious that our suggested output port requires certain changes of the synchronous block. In particular, signal data valid (DATAV\_OUT) has to be generated. However, this is also the case with the other GALS solutions [2]. In addition, the required change is relatively small and simple. Either the data valid signal should be activated one cycle before the validity of the output, or a small part of the wrapper that generates the signals DOV and DONV has to be embedded in the synchronous block. The logic equations for a hazard-free AFSM implementation of the output controller are listed below

$$\begin{aligned}
 \text{REQ\_B} &= \overline{\text{ACK\_B}} \cdot \text{REQ\_B} + \text{DOV} \cdot \overline{\text{Z0}} \\
 &\quad + \overline{\text{ACK\_B}} \cdot \text{DOV} \\
 \text{ACK\_INT} &= \overline{\text{ACK\_B}} \cdot \text{REQ\_B} + \text{DOV} \cdot \overline{\text{Z0}} \\
 &\quad + \overline{\text{ACK\_B}} \cdot \text{DOV} + \overline{\text{ACK\_B}} \cdot \text{DONV} \\
 \text{Z0} &= \text{ACK\_B} \cdot \text{Z0} + \text{ACK\_B} \cdot \overline{\text{DOV}} \\
 &\quad + \overline{\text{DOV}} \cdot \overline{\text{DONV}} \cdot \text{Z0}
 \end{aligned}$$

Z0 is an internal signal added to achieve hazard-free behaviour of the AFSM. The additional flip-flops and logic gates are implemented in the output port to support the correct operation of the AFSM and to generate the data valid signals.

### 3.5 Evaluation of the AW

We have performed extensive simulations of the AW. One trace of the request-driven GALS wrapper operation is shown in Fig. 6. From this figure, different modes of operation can be observed. The 'normal operation' is the request-driven mode when the wrapper is in the slave state and responds to the request of the master block. When the master is no longer active, the local clock generation mode will be activated. When a new request from the master arrives during the local clock generation, a transitional mode starts. In this mode, clock generation has to be handed over from the local source to the request line.

In order to validate the correctness of our proposed concept, and to investigate the possible hazards in the system, we have decided to perform a formal analysis of the AW. Our modelling approach presented by

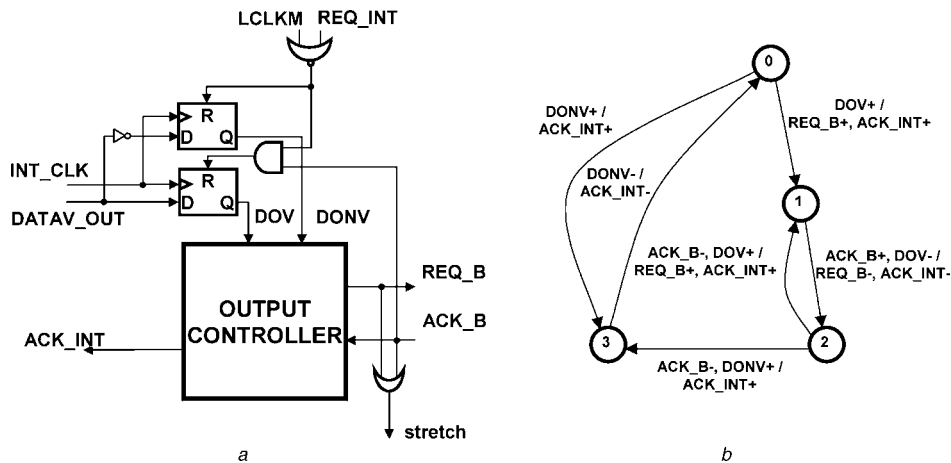


Fig. 5 Output port

- a Structure
- b AFSM specification

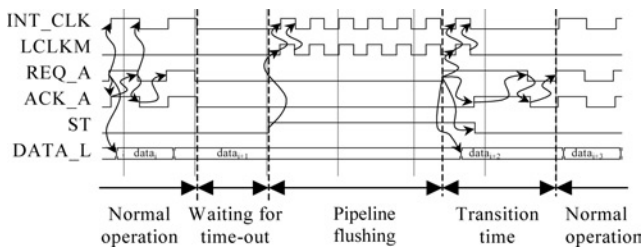


Fig. 6 Timing diagram of the data transfer

Stahl *et al.* [10] is a combination of event-based and level-based modelling [11]. For each gate type of the given wrapper, a Petri net pattern is built which describes edges and levels of all signals in the pattern. This way, a pattern preserves all information needed to detect hazards. The wrapper is a connected composition of several instances of the corresponding patterns. The question whether a hazard can occur in a gate is reduced to a model-checking problem: the reachability of a particular marking in the Petri net.

To verify the wrapper's model we have employed LoLA [12], an explicit model checker that features powerful state space reduction techniques. We have also used the known reduction technique of abstraction to alleviate the problem of state-space explosion [10]. As a result, we have detected and corrected a number of potential hazards, signal races and a deadlock.

In Table 1, the area profiling of the AW is presented [13]. For the synthesis, we have used our internal 0.25  $\mu\text{m}$  CMOS library. The tunable clock generator uses most of the area of the AW. The other blocks of the wrapper are very small. However, with an average size of 100 k gates for a GALS block, the GALSification adds an overhead of only 1–1.5%. Table 2 shows the performance and power estimation results of the AW. In request-driven mode, we can reach around 120 Msps depending on the implementation. In contrast, in the locally driven mode, the limit is 80–100 Msps. The average power consumption of a wrapper, driven by a 50 MHz data stream, is around 1 mW.

## 4 Design of the GALS baseband processor for the wireless LAN application

### 4.1 Synchronous baseband processor

Our work in the GALS area is connected to a project that aimed to develop a single-chip WLAN modem compliant to the IEEE 802.11a standard [14]. First, we have implemented the baseband processor for the WLAN modem as a standard synchronous design, as reported by Krstić *et al.* [15, 16] and Grass *et al.* [17]. It has a complexity of around 700 k gates and uses the clock gating as a power reduction technique. The global structure of the

Table 1: AW circuit area

Block/wrapper	Area, $\mu\text{m}^2$	Gates
Input controller	7100	118
Output controller	3920	65
Time-out detection	5179	86
Clock control	4732	79
Clock generation/arbiter	55026	917
Total wrapper area	79929	1332

Table 2: Evaluation of the AW performances and power consumption

Parameter/component	AW
Maximum throughput – request mode, Msps	119
Maximum throughput – local mode, Msps	86.9
Latency, ps	7590
Power, mW	1.12

baseband processor (including the GALS blocks) is given in Fig. 7. However, the synchronous processor is based on the same architecture and built with the same functional blocks. The baseband processor consists of two basic datapaths: receiver and transmitter.

The transmitter is a simple point-to-point structure with the FFT processor as the dominant area and power consumer. The other blocks include encoder, mapper, interleaver, pilot and guard interval insertion.

The receiver has a more complex architecture that includes forward and backward dataflow. The input block consists of a synchroniser that performs frame synchronisation and carrier offset correction. This block includes several autocorrelators, CORDIC processors, a peak detector and a cross-correlator. The subsequent block is a channel estimator followed by a residual phase correction unit. The channel estimator is based on a decision-directed feedback

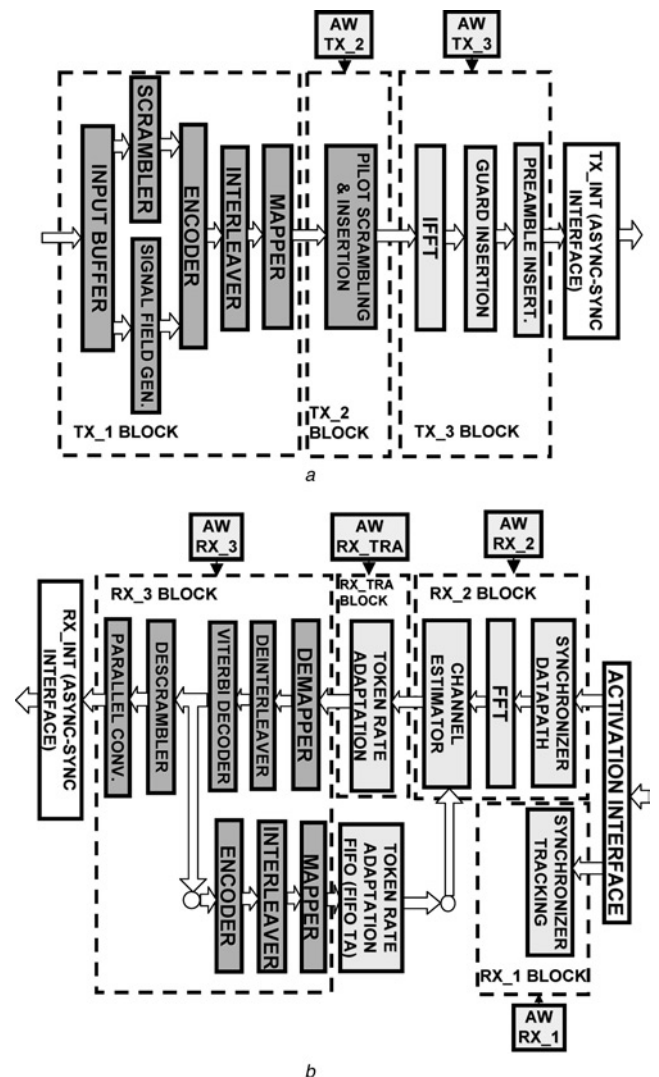


Fig. 7 Structure of the GALS baseband processor

[18]. In this method, for the channel correction of the current symbols, the demodulated data from the previous symbols is needed. Therefore the receiver includes loop processing and the output data is reprocessed and fed back to the channel estimator. This concept significantly complicates the dataflow in the receiver. The final stage of the receiver includes demapper, deinterleaver and finally a very complex and extremely power-intensive Viterbi decoder.

During the development of the synchronous baseband processor, we faced several problems. The processor has a very complex design including two clock domains (80 and 20 MHz), both derived from the same source. Therefore the design includes also a clock divider for the generation of the 20 MHz clock. Furthermore, the clock gating created an additional clock separation into different domains. In the Viterbi decoder, we even had to introduce an additional 10 MHz clock domain for the trace-back logic. In general, the complete baseband processor is RAM-free and strictly flip-flop based with 36 k leave cells. Consequently, a very complex clock-tree is required with the potential for high clock skew and significant problems with the inter-domain timing closure. As a result, even for experienced designers, this design process was long and iterative. In contrast, the design of such system must be very cheap, automated and the time-to-market must be short. Therefore we introduced the GALS technique in order to relax design challenges and to perform the complete design process fast and successful. Additionally, GALS can support our efforts in reducing noise (EMI) and power consumption.

## 4.2 GALS partitioning

On the basis of the developed synchronous blocks, the baseband processor design was initially partitioned into separate transmitter and receiver datapath structures as shown in Fig. 7. We mainly used the same architecture for both the synchronous and the GALS design. The only major change is a separation of IFFT and FFT processors in order to simplify the dataflow for the GALS implementation. In the synchronous implementation, a single processor executes both operations. However, it is possible to merge both processors for the GALS implementation as well.

The GALS transmitter is organised as a point-to-point structure. It is divided into three modules, as shown in Fig. 7a. This block division takes into account the functionality, operating frequency and the complexity of the existing blocks.

Block Tx<sub>1</sub> comprises 80 MHz components and the complexity of the block is about 17 k gates (inverter equivalent). Block Tx<sub>1</sub> is directly driven from the external clock source, that is, it is not GALSified because of a complex off-chip interface protocol. The intermediate block Tx<sub>2</sub> (20 k gates) collects data in request-driven mode at 80 Msps and waits to receive a complete burst before sending data to block Tx<sub>3</sub>. When a complete data burst for one symbol is collected, the local clock generator, set to approximately 20 Msps, initiates data transfer to the complex block Tx<sub>3</sub> (150 k gates). The communication between those two blocks Tx<sub>2</sub> and Tx<sub>3</sub> is performed only in bursts of eight data tokens. Then, block Tx<sub>3</sub> generates another 72 clock cycles to process this data without accepting any new incoming data.

The GALS receiver is significantly more complex and it has more than twice as many gates than the transmitter. Additionally, the receiver's dataflow is more complicated

and inside the datapath, there is a token ring as can be seen in Fig. 7b.

For most of the time, the receiver will just search for the synchronisation sequence. Therefore, to reduce the power consumption, we decided to implement the tracking synchroniser as a separate GALS block. This block (Rx<sub>1</sub>) is comparatively small (80 k gates). Restricting the switching activity of the baseband processor to this block for most of the time reduces power consumption. However, a similar effect is feasible in the synchronous processor using clock gating. Block Rx<sub>2</sub> is activated when coarse synchronisation is reached. This block is very complex (235 k gates) and all operations are performed at 20 Msps. Block Rx<sub>3</sub> (168 k gates), in the pure synchronous implementation, operates at 80 MHz. The application of the GALS approach theoretically allows reducing the nominal local clock frequency of this block down to the absolute limit of 54 Msps. However, block Rx<sub>3</sub> in this GALS solution uses a rate of 80 Msps. This conservative sample rate was chosen in order to re-use the design of the LS block.

One key challenge of the receiver is to connect blocks Rx<sub>2</sub> and Rx<sub>3</sub>, and to arrange a token rate adaptation 20 Msps → 80 Msps (in forward data transfer Rx<sub>2</sub> → Rx<sub>3</sub>), and 80 Msps → 20 Msps (in backward data transfer Rx<sub>3</sub> → Rx<sub>2</sub>). An additional problem is to synchronise the backward data transfer with the forward data transfer. A possible solution can be achieved by combining two types of interface blocks: GALS block Rx\_TRA and the asynchronous FIFO\_TA, as shown in Fig. 8. Block Rx\_TRA is implemented as a synchronous FIFO. This GALS block has the function to adapt the rate from 20 to 80 Msps. 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<sub>3</sub> at a datarate of 80 Msps. Block FIFO\_TA is implemented as a latch-based asynchronous FIFO. This block stores 48 tokens per OFDM symbol from block Rx<sub>3</sub> at a datarate of 80 Msps. Similar blocks exist in the synchronous processor implementation too. However, the asynchronous FIFO\_TA is much less complex than its synchronous counterpart. Using such interfaces, the backward dataflow is treated as an independent dataflow that creates tokens asynchronously. Additionally, we have implemented a join circuit for the alignment of tokens entering block Rx<sub>2</sub>. The predecessor of the join circuit, block FIFO\_TA responds to incoming tokens immediately. Therefore the rate of the joined dataflow is determined by the activation interface rate (20 Msps).

The GALS baseband processor will be normally used in a standard synchronous environment. In transmit mode, it has to deliver data tokens at every clock cycle (20 Msps) to the DACs. In receive mode, it should absorb data tokens from the ADCs at every clock cycle (20 Msps). For the connection between a synchronous producer and an asynchronous consumer, we have just connected the request line of the wrapper with the clock source that delivers synchronous data. However, it must be guaranteed by the AW that all incoming tokens are absorbed. This is safely achieved by the insertion of decoupling circuitry in blocks Rx\_TRA and Tx<sub>2</sub>. To connect the asynchronous producer with a synchronous consumer, we have used pipeline synchronisation [19] in blocks Rx\_int and Tx\_int from Fig. 7.

The GALS introduction did not cause significant changes and challenges for the system integration. The same AW architecture was used for all blocks. All synchronous blocks remained unchanged. In principle, interface block Rx\_TRA already existed in the synchronous design, whereas other interfaces had to be additionally designed.

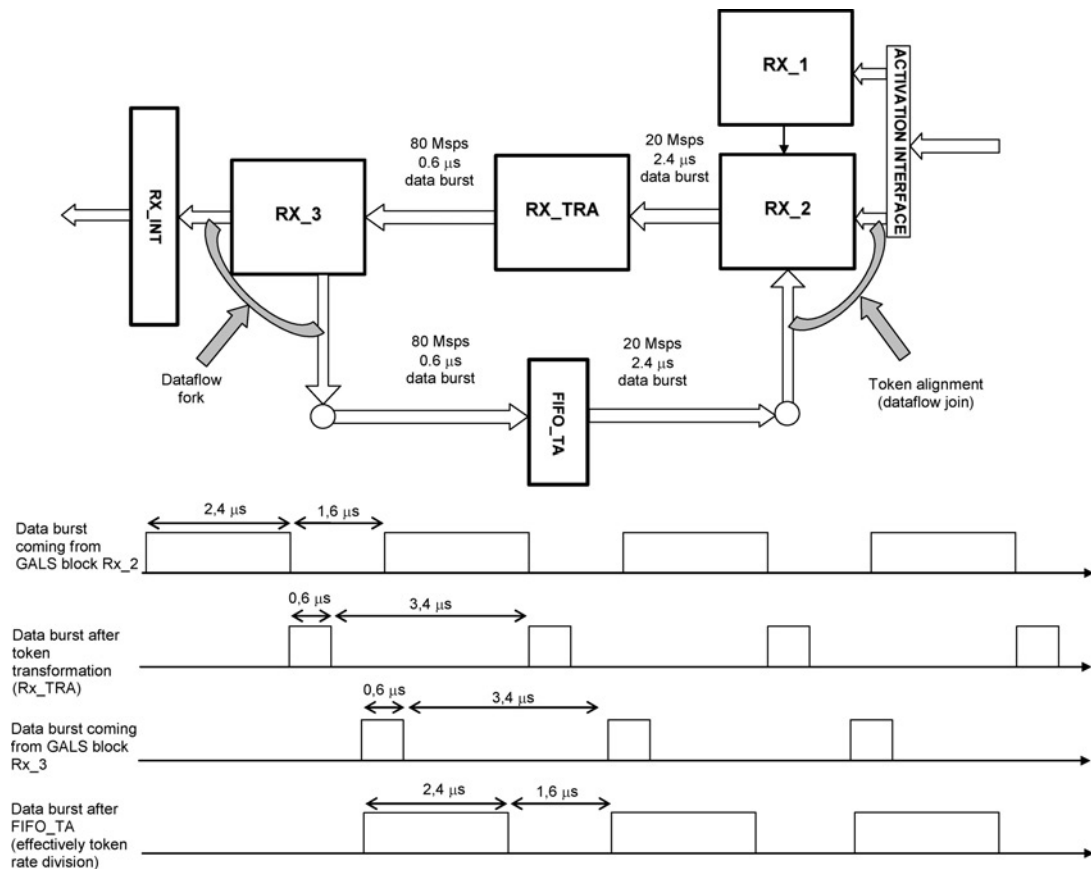


Fig. 8 Receiver dataflow organisation

However, for future GALS designs all developed blocks can be re-used.

## 5 Implementation of the GALS baseband processor

A GALS circuit is usually dominated by its synchronous modules. Therefore the GALS design flow is mostly based on synchronous CAD tools. Our proposed design flow for GALS is shown in Fig. 9. Asynchronous controllers are modelled as AFSMs and synthesised using the 3D tool [20]. Subsequently, they are converted into structural VHDL code with our tool 3D converter. For system modelling, the application of VHDL is proposed. However, VHDL is not the perfect choice for the asynchronous components as it lacks support for modelling asynchronous channels. Therefore for the system verification, it is recommended to use the synthesised Verilog-netlist for the AW and behavioural VHDL for the synchronous components in order to achieve reliable and fast simulation. For the synchronous blocks, it is possible to use standard CAD tools. Therefore during the layout phase, the advanced utilities such as in-place optimisation, or timing-driven placement and routing are limited to the synchronous components in the system.

We have applied the proposed design flow to implement the GALS baseband processor. The area and power distribution of the GALS baseband processor are given in Table 3. In terms of area, the synchronous blocks are dominant with around 90%. The test logic requires some additional overhead (around 3.5%) but this can be significantly reduced because much of it is owed to the experimental nature of this GALS design. The AWs require 2% and additional asynchronous interfaces (joins, forks and so on) are using 1.6% of area. A considerable share of 1.3%

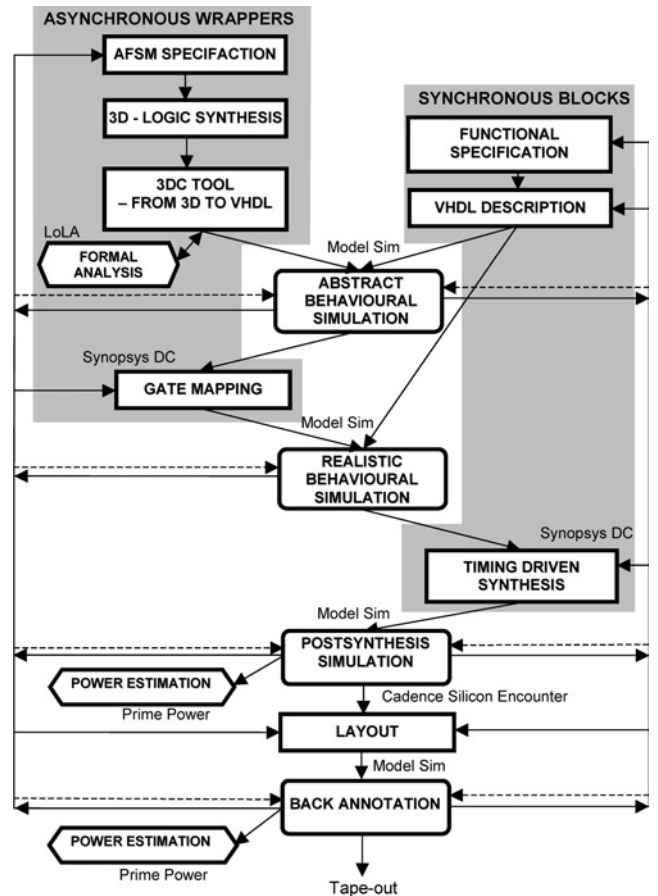


Fig. 9 Design flow for GALS

**Table 3: Area and power distribution in the GALS baseband processor**

Component/results	Area, %	Power, %
BB chip	100	100
Clock-trees	1.7	34.5
Synchronous 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
AW	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
Asynchronous-synchronous interfaces	1.3	7
Rx_int	0.5	4.4
Tx_int	0.8	2.6
Test logic	3.5	1.5

is used by the asynchronous-to-synchronous interfaces. The power estimation is based on a realistic application scenario of receiving a 100 byte frame and then transmitting a 100 byte frame. The result shows that the local clock-trees and the logic of the synchronous blocks consume nearly 90% of the power. The AW and asynchronous interfaces consume 3.5%, together. However, asynchronous-synchronous interfaces consume around 7%. This relatively large power consumption is the result of the oversizing these blocks in order to achieve robustness of FIFO-like structures. The power reduction for those interfaces will significantly affect the overall power consumption. Additionally, the test logic contributes to the power dissipation because of the switching activity of small local clock-trees. However, in the next redesign this can be easily fixed.

In the previous section, we have described the design problems of the synchronous baseband processor. The conversion of the synchronous design into the GALS design leads to certain improvements of the system integration and relaxed timing closure. Contrary to the synchronous design, with GALS, we did not have any problems with the large number of clock leaves, with the clock divider or clock gating. Additionally, because of the smaller clock domains, the clock skew was significantly reduced. For example, the estimated clock skew of the synchronous baseband processor was 660 ps. For the GALS design, the maximal clock skew was estimated to 486 ps. With more stringent clock-tree generation constraints (than currently defined 500 ps, for GALS design) an even better result can be achieved. GALS wrappers took care of the data transfer between blocks and consequently, timing closure between the clock domains was much easier.

However, due to the lack of design experience with complex GALS systems, we faced several new challenges. The major problem was the lack of support for the asynchronous design style by commercial CAD tools as well as the immaturity of the asynchronous university CAD tools. For example, the 3D tool does not support direct gate-mapping from the generated AFSM logic equations. Therefore many steps in the design process had to be performed manually. Finally, the AW optimisation has been performed in parallel to the GALS chip design. Those issues caused some iterations of the GALS design process as well. However, the general conclusion is that our GALS technique potentially offers significant simplifications of the system integration process. Eventually, the GALS application will result in shorter design cycles and lower design cost.

The GALS baseband processor was fabricated in our internal 0.25  $\mu\text{m}$  CMOS process and successfully tested. The die photo of the processor chip is shown in Fig. 10. The floorplan of the receiver and the transmitter is outlined in this figure.

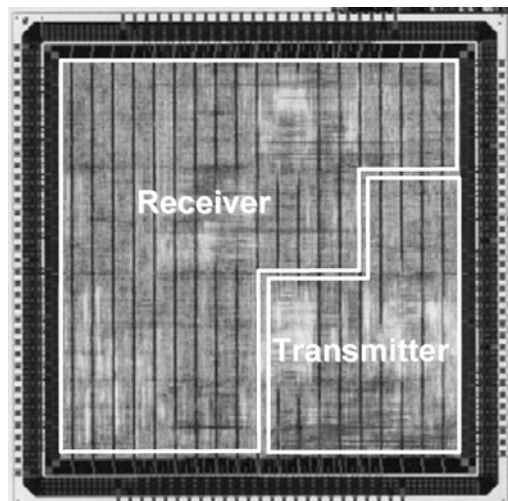
## 6 Measurement results

The fabricated GALS baseband chip was thoroughly tested. Two issues were the focus of our interest: measurements of power dissipation and supply voltage variations to gain information on spectral noise and EMI.

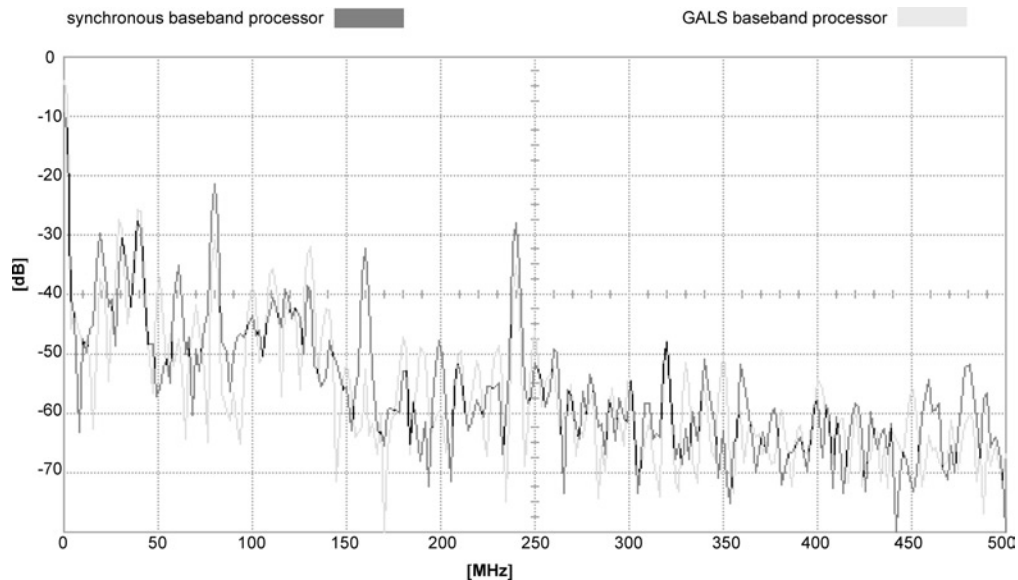
### 6.1 Measurement of power dissipation

Measuring the power dissipation was relatively easy, because our tester supports this feature. This was done on the basis of a realistic application scenario. For comparison, the same test was performed on the pure synchronous baseband chip. The dynamic power dissipation of the pure synchronous baseband processor was on an average 332 mW, and for the GALS baseband processor slightly lower, at 328 mW. Those results are based on the measurements of all chips that passed functional and BIST tests. Static power dissipation of the GALS baseband processor is slightly higher because of the larger chip area.

According to the after-layout power estimation, the power reduction with the introduction of GALS was expected to be around 17%. However, the actually measured gain is just around 1%. The reason for this discrepancy is that the power estimation of the synchronous



**Fig. 10** Die photo of the baseband processor



**Fig. 11** Spectral profile of the supply voltage in the GALS baseband processor

baseband processor was too pessimistic. In general, the GALS and the pure synchronous baseband processor deploy similar power saving strategies. For the synchronous implementation, deactivation of blocks is realised by clock-gating, whereas for the GALS version it is embedded in the operation of the AWs. The advantage of the GALS circuit is that a number of low-depth clock-trees can be used instead of one global tree. Additionally, finer granularity of the clock domains potentially leads to lower power consumption. However, the considerable power consumption in the asynchronous-to-synchronous interfaces diminished those advantages. Therefore the difference in the dynamic power consumption is very small.

The potential for lower power consumption can be higher if the potential of GALS were exploited more aggressively. For example, asynchronous-to-synchronous interfaces could be realised in a more efficient manner. In our design, they have a significant overhead in order to enhance their robustness.

## 6.2 EMI measurement

Estimations and measurements of the EMI and noise characteristics are very difficult. For our test, we have used a board that is equipped with many blocking capacitors. Therefore the direct time-domain measurement of the voltage drop was not possible. Consequently, we have decided to measure the voltage profile of one output signal pin that was set to high during the entire test. From the output pin circuitry, it is obvious that the voltage variations measured at this pin against ground are proportional to the noise on the power supply line. The measurement was performed during active operation of the baseband processor in the trans-receiver scenario. The spectral profile of the voltage for the GALS and the pure synchronous processor designs are presented in Fig. 11. For the synchronous baseband processor, the spectral components at the operating frequency are very high (20 and 80 MHz) and so are their harmonics (40, 160, 240 MHz). For the GALS processor, dominant peaks are at 30 and 40 MHz. The gain for our GALS processor can be estimated as a 5 dB reduction of the maximum spectral peak. Additionally, instantaneous peaks were reduced by 30%. However, the spectral reduction achieved here is significantly smaller

than the one we estimated in Fig. 1. The reason is the smaller number of GALS blocks in our baseband processor. However, with finer GALS partitioning, an additional noise reduction is possible.

## 7 Conclusions

In this paper, we have presented a novel GALS technique which is particularly suitable for the design of wireless communication systems. Our request-driven GALS technique is optimised for point-to-point architectures often used in communication systems. The proposed GALS technique is applied to a baseband processor for WLAN IEEE 802.11a. As a result, we have achieved improvements in the system integration by simplifying the timing closure. Additionally, the GALS design shows significant improvements of clock skew and a noticeable reduction of EMI. Because of the experimental nature of our GALS implementation, its power dissipation is very similar to the synchronous counterpart.

However, further optimisation of the GALS baseband processor is possible and more compelling results can be achieved. For example, the interface between a synchronous and an asynchronous domain can be improved. Dynamic voltage scaling can be introduced for additional power saving; EMI can be further reduced with finer partitioning, clock phasing and jittering.

Although we have achieved good results in reducing the EMI with our proposed GALS technique, further improvements are possible. Therefore we will investigate the introduction of additional jitter in the GALS clock and its effect on EMI reduction. In the area of formal verification, some additional work is also needed. Furthermore, we are considering a redesign of the current GALS wrapper in order to allow external clocking instead of local clock generation. With such solution, we can avoid the problems of clock tuning and power consumption of the ring oscillators. Finally, additional work is needed to automate the design flow for GALS and consequently to make GALS more suitable for conventional synchronous designers. We believe that GALS circuits can facilitate the efficient re-use of IP-blocks and will be more widely used in future.

## 8 References

- 1 Chapiro, D.M.: 'Globally-asynchronous locally-synchronous systems'. PhD thesis, Stanford University, October 1984
- 2 Muttersbach, J.: 'Globally-asynchronous locally-synchronous architectures for VLSI systems'. Doctor of Technical Sciences Dissertation, ETH Zurich, Switzerland, 2001
- 3 Moore, S., Taylor, G., Mullins, R., Cunningham, P., and Robinson, P.: 'Self-calibrating clocks for globally asynchronous locally synchronous system'. Proc. Int. Conf. on Computer Design (ICCD), Austin, TX, USA, September 2000
- 4 Bormann, D., and Cheung, P.: 'Asynchronous wrapper for heterogeneous systems'. Proc. Int. Conf. on Computer Design (ICCD), Austin, TX, USA, October 1997
- 5 Krstić, M., and Grass, E.: 'New GALS technique for datapath architectures'. Proc. Int. Workshop Power and Timing Modeling Optimization Simulation (PATMOS), Torino, Italy, Sept. 2003 (*Lect. Notes Comput. Sci.*, 2003, **2799**), pp. 161–170
- 6 Krstić, M., Grass, E., and Stahl, C.: 'Request-driven GALS technique for wireless communication system'. Proc. 11th IEEE Int. Symp. Asynchronous Circuits Systems (ASYNC 2005), IEEE Computer Society, New York, March 2005, pp. 76–85
- 7 Badaroglu, M., Wambacq, P., Van der Plas, G., Donnay, S., Gielen, G., and De Man, H.: 'Digital ground bounce reduction by phase modulation of the clock'. Proc. Design, Automation and Test in Europe Conf. Exhibition (DATE'04), Paris, 2004, vol. I
- 8 Blunno, I., Narboni, G.A., and Passerone, C.: 'An automated methodology for low electro-magnetic emissions digital circuits design'. Proc. EUROMICRO Symp. on Digital System Design (DSD'04), Rennes, France, 2004, pp. 540–547
- 9 Sparsø, J., and Furber, S.: 'Principles of asynchronous circuit design' (Kluwer Academic Publishers, 2001)
- 10 Stahl, C., Reising, W., and Krstić, M.: 'Hazard detection in a GALS wrapper: a case study', in Desel, J., and Watanabe, Y. (Eds.), 5th Int. Conf. Application of Concurrency to System Design (ACSD'05), IEEE Computer Society, 2005, pp. 234–243
- 11 Yakovlev, A., and Koelmans, A.M.: 'Petri nets and digital hardware design', in 'Lectures on Petri Nets II: Applications', (*Lect. Notes Comput. Sci.*, **1492**, 1998)
- 12 Schmidt, K.: 'Lola – a low level analyser', in Nielsen, M., and Simpson, D. (Eds.), Proc. ICATPN, (*Lect. Notes Comput. Sci.*, 2000, **1825**), p. 465
- 13 Grass, E., Winkler, F., Krstić, M., Julius, A., Stahl, C., and Piz, M.: 'Enhanced GALS techniques for datapath applications'. Proc. Int. Workshop Power and Timing Modeling, Optimization Simulation (PATMOS), Leuven, Belgium, 2005, (*Lect. Notes Comput. Sci.*, **3728**, 2005), pp. 581–590
- 14 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
- 15 Krstić, M., Maharatna, K., Troya, A., Grass, E., and Jagdhold, U.: 'Baseband processor for IEEE 802.11a standard with embedded BIST', *Facta Univ., Ser.: Electron. Energ.*, 2004, **17**, pp. 231–239
- 16 Krstić, M., Troya, A., Maharatna, K., and Grass, E.: 'Optimized low-power synchronizer design for the IEEE 802.11a'. Proc. Int. Conf. Acoustics, Speech and Signal Processing (ICASSP), Hong Kong, 2003, vol. II, pp. 333–336
- 17 Grass, E., *et al.*: 'On the single-chip implementation of a Hiperlan/2 and IEEE 802.11a capable modem', *IEEE Pers. Commun.*, 2001, **8**, (6), pp. 48–57
- 18 Mignone, V., and Morello, A.: 'CD3-OFDM: a novel demodulation scheme for fixed and mobile receivers', *IEEE Trans. Commun.*, 1996, **44**, (9), pp. 1144–1151
- 19 Seizovic, J.: 'Pipeline synchronization'. Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, Salt Lake City, UT, USA, November 1994, pp. 87–96
- 20 Yun, K., and Dill, D.: 'Automatic synthesis of extended burst-mode circuits: Part I and II', *IEEE Trans. Comput. Aided Des.*, 1999, **18**, (2), pp. 101–132