

Request-driven GALS Technique for Wireless Communication System

Miloš Krstić¹, Eckhard Grass¹, Christian Stahl²

¹*IHP, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany*

²*Humboldt Universität, Rudower Chaussee 25, 12489 Berlin, Germany*

E-mail: krstic@ihp-microelectronics.com

Abstract

A Globally Asynchronous - Locally Synchronous (GALS) technique for application in wireless communication systems is proposed and evaluated. The GALS wrappers are based on a request-driven operation with an embedded time-out function. A formally verified GALS wrapper is deployed for the 'GALSification' of a baseband processor for WLAN. Details of the GALS partitioning, implementation and the design-flow are discussed. Furthermore, a test strategy based on built-in self-test (BIST) is suggested. The described baseband processor was fabricated and successfully tested. The GALS design is compared with a clock-gated, synchronous version. Advantages for system integration are achieved along with a 1% reduction in dynamic power consumption, a 30% reduction in peak power supply current, and 5 dB reduction in spectral noise.

1. Introduction

The process of integration and miniaturization of mobile communication systems imposes a number of challenges. Even though digital systems design is supported by very advanced tools, the problems of block integration, clock domain synchronization, clock tree generation, clock gating and timing closure are not trivial.

The deployment of GALS techniques can make system-on-chip (SoC) integration easier by circumventing the problems above. GALS has been studied in the scientific community for a number of years and there have been many valuable proposals [1, 2, 3]. The current level of GALS development facilitates a reasonable framework for complex digital system implementations. However, existing GALS techniques are oriented towards general system architectures and introduce drawbacks in performance, some additional system constraints and hardware overhead.

In this paper, we will investigate a GALS technique designed to alleviate integration challenges for architecturally complex but medium performance SoCs. By introducing GALS, we limit the common design challenges to the level of local blocks. In [4] we introduced a request-driven GALS technique, intended for a datapath architectures with bursty and very intensive

data transfer. The applicability of our proposed request-driven GALS technique was investigated in a complex datapath application. A GALS baseband processor, compliant to the IEEE 802.11a standard [5], was designed, fabricated and measured in order to evaluate the application of our GALS technique in communication systems.

The baseband processor integrates sophisticated processing modules such as a Viterbi decoder, FFT/IFFT-, and different CORDIC- processors. Implementing this design in a pure synchronous environment was a challenging process due to its gate count, multiple clock-domains, clock-gating and complex clock tree [6,7]. With the GALS application, we simplified those issues. The design was partitioned into several sub-blocks of lower complexity. Consequently, the clock-tree generation and timing-closure for the blocks was significantly easier. The asynchronous wrapper inherently implements power saving by reduction of switching activity. On the other hand, block interfacing and global data-transfer was facilitated with asynchronous wrappers.

The test methodology for the baseband processor is based on hierarchical, built-in self-test (BIST). The proposed test solution is applicable in any GALS system with datapath structure. Our test method allows the use of a synchronous digital tester and gives good controllability and observability of the test procedure.

The fabricated baseband processor was thoroughly tested and measured using BIST and functional test vectors. The measurements showed that GALSification leads to slightly improved power and EMI results.

In Sections 2 and 3, we will describe the concept of our request driven GALS technique and the formal analysis of our asynchronous wrapper respectively. In Section 4, the GALS partitioning of the baseband processor will be presented. Section 5 is dedicated to design for testability (DfT) for GALS. Design flow issues are discussed in Section 6, and measurement results are presented in Section 7.

2. Request-driven GALS Technique

The principle architecture of our proposed asynchronous wrapper around a locally synchronous (LS) module is shown in Fig. 1. Our asynchronous wrappers

have been described in [4], which also gives detailed information about the specific individual blocks of our GALS system.

The main prerequisite for the following discussion is that the communication between blocks is very intensive (i.e. every clock cycle of the local clock) but bursty, with longer periods of inactivity. 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.

The key idea behind our request-driven approach is that a module can use the request input signal as its clock while receiving a burst of data. Therefore, in the mode of operation in which data is continuously received at the input port, the *request-driven clock* (Fig. 1) is directly derived from the request signal at the input port. However, after an input burst is received and there is no activity on the input handshake lines, the data stored inside the locally synchronous pipeline has to be processed and flushed out. This can be achieved by switching to a mode of operation in which a local clock generator drives the GALS block independently. To control the transition from request driven operation to the local clock generation mode, a time-out function is proposed. The time-out function is triggered when the input request line of the GALS block is idle for a certain period of time (defined as a $T_{\text{time-out}}$), but data that has to be processed is still stored in internal pipeline stages. Then the circuit enters a new state where it can internally generate clock cycles using a local ring oscillator. We usually use 2-3 clock cycles of the request driven clock as a time-out delay. Shorter delays may create problems in distinguishing the real time-out from the jitter on the request signal. Longer delays may reduce performances. 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 new data token has appeared at the input.

More complex and demanding is the scenario when after a time-out and starting of the local-clock generation, a new request appears before the synchronous pipeline is emptied. In this case, first it is necessary to complete the present local clock cycle. Subsequently, it is possible to safely hand over clock generation from the local ring oscillator to the input request line. The activity of the *request-driven clock* and the *locally generated clock* is mutually exclusive and hence hazards are avoided.

Communication between different AWs is based on the simple 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 employing standard asynchronous elements such as ‘joins’ and ‘forks’.

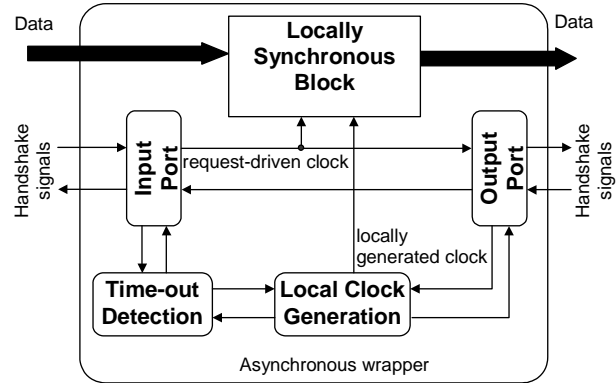


Fig. 1 Asynchronous wrapper structure

The proposed architecture leads to a simple framework for integration of complex digital systems. It promises several advantages over other GALS techniques. In particular, reliable and fast transfer of large bursts of data is achieved. Data transfer is possible at every clock cycle of a synchronous block. The clock speed is determined by the clock speed of the master and not by the slower participant in the communication. In the request-driven mode, synchronization is not needed and incoming requests are automatically considered as clock signals. Therefore, data transfer latency is decreased and non-deterministic behavior is limited. The proposed GALS approach follows the usual design style of synchronous blocks and doesn't require significant changes in the locally synchronous components.

There are issues that limit the applicability of the proposed technique. There is a latency increase due to the time-out measurement. The introduction of the ‘end-tokens’ to trigger emptying of internal pipeline stages, as we proposed in [4], can improve this weakness. The applicability of our wrapper is limited to architectures with bursty data-streams. Processing of single tokens can diminish performance. The proposed implementation of this GALS concept is more complex than other GALS techniques.

We can expect following 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. Additionally, the local clock generators of the local blocks will run at different frequencies. In this way the generation of substrate noise, supply noise, crosstalk and the amplitude of instantaneous supply voltage peaks can be reduced.

This GALS technique offers a 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. This mechanism is embedded into the

proposed GALS concept. The depth of the clock tree is reduced due to decoupling of the global tree into a set of independent local trees. Additionally, fine-grained clock gating can be applied within locally synchronous blocks. However, the general power saving limit of any GALS technique due to clock pausing is very much comparable with clock gating, and it is not realistic to expect significant change in the power consumption in comparison with a synchronous system with clock gating.

GALSification also enables the efficient application of other power reduction techniques. For example, in pure synchronous systems, blocks are often clocked at higher frequencies than actually needed. This is done to avoid problems of synchronization with neighboring 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. For each GALS block, the optimal supply voltage could be applied which would enable further power reductions.

3. Formal Analysis of the Wrapper

In order to validate the correctness of our proposed concept and to investigate possible hazards in the system, we performed a formal analysis of the asynchronous wrapper. For each gate in the wrapper, we built a model that describes the relation between signal transitions, gate delays and logic level. For analysis purposes we used Petri nets. The wrapper consists of only a few different elements (basic logic gates, flip-flops, counters, mutex, c-elements). We created a Petri net pattern for every element. For modelling those elements, we used a methodology similar to the one described in [8]. We aimed at modelling all properties in the patterns; thus the complete wrapper can be modelled by plugging the respective patterns together.

To verify the wrapper we used the Petri net based model checker LoLA [9] that features powerful state space reduction techniques. In the first step, we reduced the net size. This can be done through introduction of causalities into the model, because the initial model completely ignored the timing relationships caused by switching and propagation delays. Secondly, combinations of simple gates are merged, e.g. two AND gates whose output signals are the input signals of an OR gate. Our current wrapper model is a Petri net with 246 places and 423 transitions. The full state space exceeds the memory of the PC used. Nevertheless, due to property specific techniques available in LoLA, it was possible to check for hazards.

To find possible hazards, all markings for a hazard have to be calculated and their reachability has to be checked. For each potential hazard, LoLA constructs a sequence of transitions from the initial marking to the

hazard marking. We analyzed each of these critical sequences of transitions. Afterwards we tried to simulate these sequences, in order to confirm whether any ‘dangerous’ state transitions can occur in reality. During the analysis we found a glitch in the Input port and another potential hazard in the Time-out generator. Fortunately, the glitch cannot fire any false token and consequently cannot lead to a malfunction of the system. The latter hazard can be ignored, because it can never occur in the proposed application scenario. In summary, the formal analysis was very useful to confirm the correctness of the proposed GALS method and to clearly formulate the timing constraints for the wrapper.

4. GALSification of the Baseband Processor

The IEEE 802.11a [10, 11] standard defines wireless broadband communication systems in the 5 GHz band with data rates ranging from 6 to 54 Mbit/s. Initially, the required baseband processor for this standard was implemented as an ASIC with complexity of about 700k gates working synchronously. This design is deeply pipelined and register based without memory structures.

The standard [10] defines the functionality for receiver and transmitter datapath processing. Our transmitter comprises an input buffer, scrambler, signal field generator, encoder, interleaver, mapper, 64-point IFFT and circuitry for pilot insertion, guard interval insertion, and preamble insertion. Fundamental units of the receiver are the synchronization block, including a 64-point FFT, and the channel estimation block. Other blocks of the receiver are the demapper, deinterleaver, descrambler, Viterbi decoder, and additional buffers.

The design of the synchronous baseband processor involves a number of challenges: global clock tree generation with two different clock domains (80 and 20 MHz, and a clock-divider on-chip), an internally generated 10 MHz clock domain for the Viterbi trace-back logic, timing closure between the different modules, clock gating, and clock skew handling for approximately 36 kflip-flops. The design process should be fast and automated, due to the short time to market. However, due to the CAD tool limitations and design complexity, the complete process was very prolonged and iterative. As we plan to integrate a complete WLAN modem into single chip, the integration problems will be even worse.

The request-driven GALS architecture was applied as a possible solution for those integration problems. Based on the existing synchronous functional blocks, the GALS design was partitioned into separate transmitter and receiver datapaths as shown in Figure 2. The architecture of the GALS and the synchronous baseband processor are mainly the same. The main change in the GALS design is a separation of IFFT and FFT processors in order to simplify the dataflow and consequently the

implementation. In contrast, for 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.

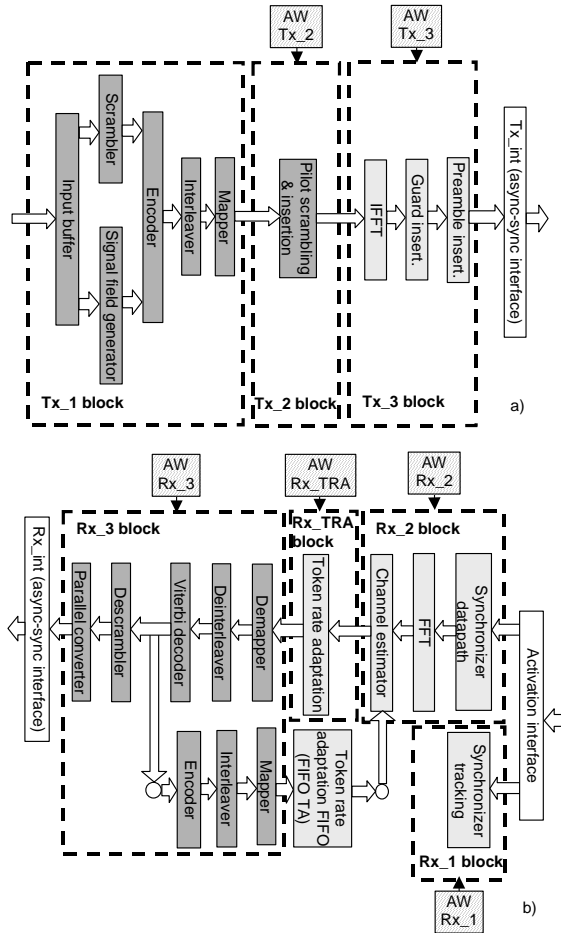


Fig. 2 GALS transmitter (a) and receiver (b)

The GALS transmitter is a natural point-to-point architecture, based on a token-flow design technique. It is divided into three modules, as shown in Fig. 2a. This block division takes into account the functionality and complexity of the existing blocks. Two of those blocks are fitted with an asynchronous wrapper (AW).

Block Tx_1 comprises 80 MHz components and the complexity of the block is about 17k gates (inverter equivalent). Block Tx_1 is directly driven from the external clock source, i.e. it is not GALSified to avoid a complex off-chip interface protocol. The intermediate block Tx_2 (20k gates) performs pilot insertion and token rate adaptation. It receives tokens at a frequency of 80 Msp/s and generates tokens at 20 Msp/s for the subsequent block. This is performed in the following way: when block Tx_2 is in request-driven mode, it collects data at 80 Msp/s and waits to receive a complete burst before

sending data to block Tx_3. When a complete data burst for one symbol is collected the local clock generator, set to approximately 20 Msp/s, initiates data transfer to block Tx_3. Block Tx_3 contains the very complex (150k gates) IFFT processor. A communication between those two blocks Tx_2 and Tx_3 is performed only in bursts of 8 data tokens and then block Tx_3 generates the next 72 cycles to process this data without accepting any new incoming data.

The GALS receiver is significantly more complex. The gate count is more than double that of the transmitter. Additionally, the data-flow is much more complicated. The channel estimator is based on a feedback loop. Accordingly, inside the receiver there is a token ring between the blocks as can be seen in Fig. 2b.

The activation scheme of the baseband processor given in Fig. 3 shows that for most of the time, the receiver will just search for the synchronization sequence. Therefore, we chose to implement the tracking synchronizer as a separate GALS block. This block (Rx_1) is comparatively small (80k gates), and with limitation of the switching activities inside the baseband processor to this block for most of the time, the power consumption is kept low. Block Rx_2 is activated when coarse synchronization is reached. This block is very complex (235k gates) and all operations are performed at 20 Msp/s. Block Rx_3 (168k) performs the most timing critical deinterleaving and Viterbi decoding. In the pure synchronous implementation, those components operate at 80 MHz. However, application of GALS allows reducing the nominal local clock frequency down to the absolute limit of 54 Msp/s.

One of the main challenges in the receiver is to connect blocks Rx_2 and Rx_3, and to arrange token rate adaptation 20 Msp/s \rightarrow 80 Msp/s (in forward data transfer Rx_2 \rightarrow Rx_3), and 80 Msp/s \rightarrow 20 Msp/s (in backward data transfer Rx_3 \rightarrow Rx_2). An additional problem is how to synchronize 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 Figure 2. Both blocks have the same purpose as in the synchronous processor implementation. However the asynchronous FIFO_TA is much less complex than the synchronous counterpart. This way the backward dataflow is treated as an independent dataflow that creates asynchronous tokens. A critical point in this solution is the control of the asynchronous token ring. Since the latency in this feedback loop can vary, joins and forks could potentially disrupt the smooth operation of the system. Therefore, we have implemented the join circuit for the alignment of tokens entering block Rx_2. FIFO_TA block performs the handshake very fast. Therefore, the rate of the joined dataflow is determined by the activation interface rate (20Msp/s).

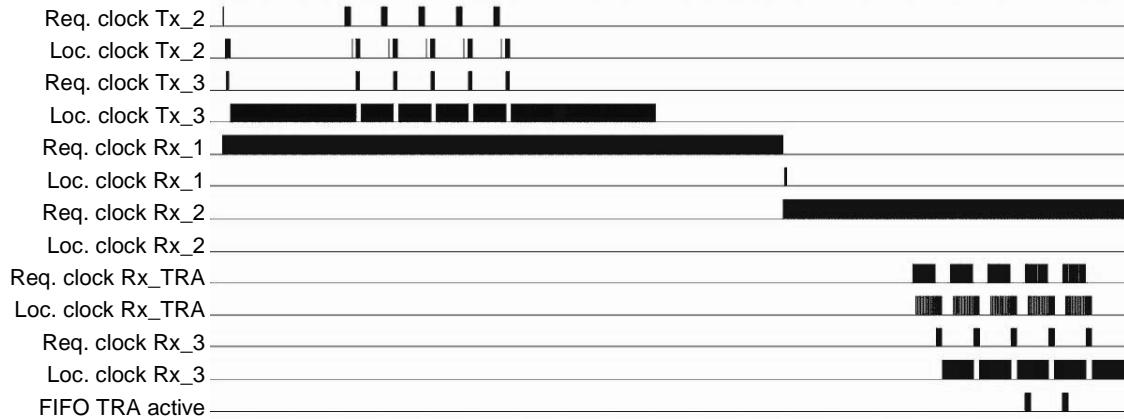


Fig. 3 Activation scenario in the Baseband processor

The GALS baseband processor will normally be used in a standard synchronous environment. In transmit mode it has to send data tokens every clock cycle (20 Msps) to DACs. In receive mode it should absorb data tokens from the ADCs at every clock cycle (20 Msps). The connection between a synchronous producer and an asynchronous consumer is achieved in a simple manner. Due to the request driven property of the asynchronous wrapper, we have just connected the request line of the wrapper with the clock source that delivers synchronous data. It must be guaranteed by the asynchronous wrappers that all incoming tokens are absorbed. This is safely achieved with the insertion of decoupling circuitry in blocks RX_TRA and Tx_2. To connect the asynchronous producer with a synchronous consumer, we have used pipeline synchronization [12] (blocks Rx_int and Tx_int in Fig. 2). The addition of more pipeline stages leads to an increase of the system robustness but also in significantly higher power consumption.

Using the proposed partitioning inside the baseband processor it is possible to efficiently implement power saving mechanisms. As shown in Fig. 3, after reset at the very beginning of the trace, the baseband processor shows no switching activity. Activity starts when the first data from outside arrives or transmission is initiated. After that, in receive mode, most of the time only block Rx_1 will be active, trying to find the synchronization pattern. This block is relatively small, and the power consumption level is acceptable. When synchronization 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 Fig. 2 from ‘tracking’ to the ‘datapath’ synchronizer. As shown in Fig. 3, block Rx_2 is never activated. This is due to the nature of the processor operation. In general, local clocks of blocks Rx_1 and Rx_2 are operating for very few cycles after the activation mechanism disables the data flow from the respective block. During these few cycles, the synchronous block is reinitialized and prepared for the

next data burst. Block Rx_3 is activated only when a frame is detected and the channel estimation is performed. Then, the Viterbi decoder processes the data, and Rx_3 becomes active. Blocks Rx_TRA and FIFO_TA are active only for short periods of time. During operation of the receiver, the transmitter is inactive. 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 didn’t cause significant changes and challenges for the system integration. The same asynchronous wrapper architecture was used for all blocks. All synchronous blocks remained unchanged. In principal, interface block Rx_TRA already existed in the synchronous design, whereas other interfaces had to be additionally designed. However, for future GALS designs all blocks can be re-used.

5. Design for Testability in GALS Systems

Testability has always been a weak point of asynchronous systems and one of the most important reasons why asynchronous techniques have never gained industrial popularity.

DfT of synchronous digital systems is usually based on the scan chain approach. There are similar proposals for asynchronous systems [13]. On the other hand, there are claims that functional testing is sufficiently effective for GALS wrapper circuits [14]. However, it is also possible to combine efficient functional testing with embedded random pattern generation. BIST could be a possible compromise between the functional and scan-based testing. In several papers [15, 16], BIST is reported as a suitable technique for testing GALS systems. However, particular test strategies are not reported.

Our main motivation for DfT introduction in the GALS baseband processor is to have a possibility to run very complex functional tests internally (without providing ex-

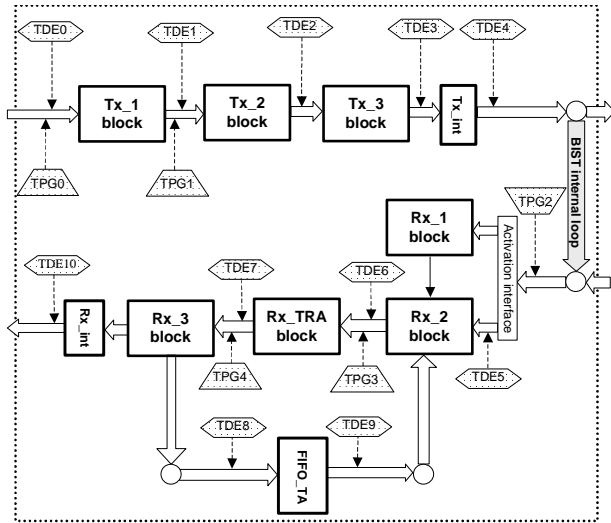


Fig. 4. BIST test points in GALS design

ternal test vectors). We are using a hardware tester which 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.

The BIST structures were applied in the GALS baseband processor as shown in Fig. 4. In order to more thoroughly investigate the correctness of the GALS baseband processor, five different Test Pattern Generators (TPGs), and accordingly, five different types of tests are implemented. TPGs are based on the Linear Feedback Shift Register (LFSR) structure with embedded additional logic. The purpose of this additional logic is to enable non-random test vector generation that opens the LS pipelines for efficient testing. For example, we cannot test any block of the receiver beyond Rx_1 if we don't provide the synchronization sequence. For this case, the additional logic of TPG will first generate the preamble sequence and then random data. The TPGs are positioned between the asynchronous wrappers and support the handshake protocol for asynchronous communication. In order to observe possible faults, eleven different Test Data Extractors (TDEs) are implemented in the system. TDEs are inserted at the most important positions between the GALS blocks, where inter-block communication can be observed. They are triggered with the handshake signals of the respective GALS blocks. TDEs are constructed in such way that they record the changes on the data lines only when a valid control token is present. Consequently with this BIST approach we can tolerate timing nondeterminism in the LS behavior. With the proposed test architecture and position of TPGs and TDEs we can verify the functionality of both asynchronous GALS control flow and the locally synchronous block operation.

A central BIST controller (CBC), specially designed for this GALS implementation, performs control of the complete test procedure. When the particular BIST is finished, the state of signal *Test_OK* should indicate the test success. Activation of all BIST I/O pins is completely timing deterministic and synchronized since the CBC is driven from an external clock. Hence, a hardware tester can be used for automatic testing.

With the proposed test structure we have the possibility to run hierarchical tests. We have implemented a global test of the complete transceiver structure. This test is initiated from TPG0 in Fig. 4. TPG0 has an additional control mechanism that performs initialization of the transmitter and sets the number of transmitted data words. After that, random data is generated 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. During the test, the complete datapath is in operation and all TDE blocks are activated. Hence, this test will yield global information about the 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.

Additionally, the 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. For example, there is a test that mainly focuses on the transmitter operation. One test is focussed on the complete receiver. There is a test that takes a closer look at the receiver feedback-loop and finally a test of the very complex operation of block Rx_3 can be carried out. A suitable combination of different tests gives us detailed information about the GALS processor operation. A similar test strategy was implemented in the synchronous baseband processor. However, the implemented BIST structure there was simpler and provided less test coverage.

Due to lack of tool support, it is impossible for us to calculate the achieved test coverage 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 synchronizer (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 with this BIST. However, the estimated test coverage that we can reach is significantly higher than with a pure functional test (we can run hierarchical tests and start the test at different internal

estimation. The overall estimated dynamic power consumption is 324.6 mW. In the simulation example, the baseband processor receives one frame and transmits one frame. The synchronous logic blocks are using 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 wrappers with 2.9%. From this analysis it is possible to conclude that the power consumption of the asynchronous wrappers including pausable clock generators is comparatively low. However, the power consumption of the async-sync interfaces exceeds the usual limits for this type of circuitry. We used complex FIFO structures in those interfaces to increase the robustness of the system. However, these structures could be optimized which would reduce their power consumption significantly. Additionally, simple local clock trees in the BIST logic are not disabled during ‘normal’ operation.

Table 1 Area and power in GALS processor

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

Our GALS baseband processor was fabricated and the die photo of the produced chip is given in Fig. 6. The total number of pins is 120 and the silicon area including pads

is 45.1 mm². For comparison, the equivalent synchronous baseband processor occupies around 34 mm². The significant increase in silicon area for the GALS chip is mainly due to a separate FFT/IFFT implementation.

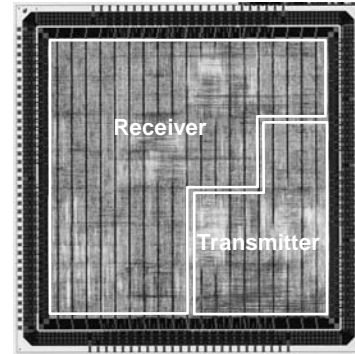


Fig. 6 Die photo of the fabricated GALS chip

7. Experimental results

The fabricated GALS baseband chip was thoroughly tested. For this purpose an Agilent 93000 hardware tester was used. After BIST and functional test, some additional measurements were performed. Two issues were the focus of our interest: measurements of power and supply voltage variations (to gain information on spectral noise and EMI).

Performing the power measurements was relatively easy, because the tester supports this feature. This was done on the basis of a realistic application scenario. The test represents the transmission of a 100 Byte frame followed by the reception of a 100 Byte frame at a transmission rate of 54 Mbps. For comparison, the same test was performed on the pure synchronous baseband chip. Statistical analysis was done on the basis of the results from the chips that have passed all BIST and functional tests. The dynamic power dissipated in the pure synchronous baseband processor was 332 mW, and for the GALS baseband processor slightly lower, at 328 mW. Static power dissipation in the GALS baseband processor is slightly higher due to the larger chip area.

According to the after-layout power estimation, the expected 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 power estimation of the synchronous baseband processor was overly pessimistic. In general, the GALS- and the pure synchronous baseband processor deploy similar power saving strategies. For the synchronous implementation deactivation of blocks is realized by clock-gating whereas for the GALS version it is embedded in the operation of the asynchronous wrappers. 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 leads to lower power consumption. However, the considerable power consumption in the async-sync interfaces diminished those advantages, so the difference in the dynamic power consumption is very small.

The potential for lower power consumption is greater if GALS is used in a more aggressive way. For example, asynchronous-to-synchronous interfaces could be realized in a more optimal manner. In our design they have a significant overhead due to robustness. Additionally, the most complex Rx_3 block can operate with much lower sample rate than the currently used 80 Msps. This conservative sample rate was chosen in order to re-use the design of the locally synchronous blocks.

Performing the EMI measurements was more complicated. Initially, we intended to directly measure the supply current profile via a small shunt resistor. However, this was not possible due to the 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 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.

Firstly, our measurement showed that instantaneous supply voltage peaks are reduced from 140 mV (synchronous design) from cycle to cycle to the less than 100 mV (GALS). After that, the spectrum of the power supply voltage was calculated. The result is shown in Fig. 7 in the measured spectrum of 500MHz. We can identify strong spectral components at frequencies of 20, 80, 160 and 240 MHz for the pure synchronous chip. 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 product of the actual settings of our stoppable clock generators in the asynchronous wrappers. However, from Fig. 7 we conclude that the GALS version does indeed show a smoother spectrum of the supply voltage when compared to the pure synchronous circuit. This reduction of supply noise will potentially make the GALS circuit a better candidate for application in mixed-signal designs. However, additional noise suppression is possible with some clock phasing effort within every particular locally synchronous block.

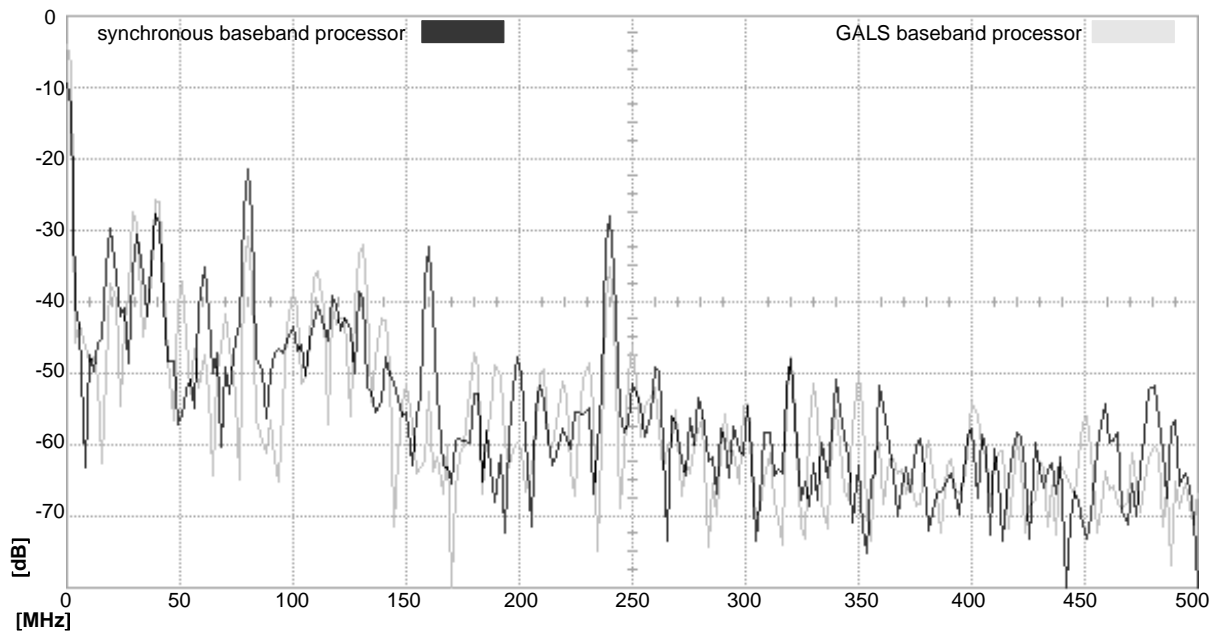


Fig. 7 Spectral profile of the supply voltage variations

8. Conclusions and further work

The GALS design process of a complex WLAN baseband processor is presented in this paper. To our knowledge, this is one of the most complex GALS implementation on silicon reported so far. The challenging process of power-optimal partitioning, based on our novel request-driven GALS technique, is described. The structure of our GALS baseband processor is complemented with several additional blocks necessary for efficient operation of the system. Furthermore, 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. The application of our GALS technique relaxed global timing, simplified clock tree-generation and reduced clock-skew. Furthermore, the results of measurements show that the GALSification leads to slightly lower dynamic power consumption and a considerable reduction of supply noise.

The request-driven GALS technique is applicable for systems with complex data paths based on point-to-point communication and bursty data transfer. The proposed GALS concept and corresponding design-flow are based on 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 the possible performance degradation. The proposed concept is also not directed toward low-performance systems that can operate correctly with the standard design support. Additionally, the concept is restricted for the applications with irregular and sporadic data-transfer between the blocks.

Further work is under way in several areas. In the area of formal analysis, we want to calculate the full state space. This can be achieved by further abstraction of the model. We will also investigate the use of symbolic model checking and wrapper decomposition. Furthermore, we are aiming at modelling the complete control flow in the GALS baseband processor using Petri nets.

For wider acceptance of the GALS methodology a number of improvements are possible. Deployment of many ring oscillators in the asynchronous wrappers is not practical, because of the need for complicated calibration and their additional power dissipation. Therefore, we plan to extend our asynchronous IP-library by introducing GALS wrappers for mesochronous operation. Finally, the used design flow is not fully automated. Additional work is under way to establish a complete user-friendly design framework that will allow easy integration of complex GALS blocks.

Acknowledgements. We thank Mark Greenstreet for his guidance during the paper revision process. We also thank Alexandra Julius for her support in verifying the function of the asynchronous wrapper.

9. References

- [1] J. Muttersbach, *Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems*, Doctor of Technical Sciences Dissertation, ETH Zurich, Switzerland, 2001.
- [2] S. Moore et al., Self Calibrating Clocks for Globally Asynchronous Locally Synchronous System, *Proc. ICCD*, Sep 2000.
- [3] D. Bormann, P. Cheung, Asynchronous Wrapper for Heterogeneous Systems, *Proc. ICCD*, Oct 1997.
- [4] M. Krstić, E. Grass, New GALS Technique for Datapath Architectures, *Proc. PATMOS*, pp. 161-170, Sep 2003.
- [5] M. Krstić, E. Grass, GALSification of IEEE 802.11a Baseband Processor, *Proc. PATMOS*, pp. 258-267, Santorini, Greece, Sep. 2004.
- [6] M. Krstić, et al., Baseband Processor for IEEE 802.11a standard with embedded BIST, *Facta Universitatis, Series: Electronics and Energetics*, Nis, Vol. 17, Aug 2004, pp. 231-239.
- [7] M. Krstić, et al., Optimized Low-Power Synchronizer Design for the IEEE 802.11a, *Proc. ICASSP*, Vol II, pp. 333-336, Hong Kong, 2003.
- [8] H. Genrich, R. Shapiro. Formal Verification of an Arbiter Cascade, In *Jensen, K., editor, Proc. ICATPN, Sheffield, UK, Springer LNCS, Vol. 616*, June 1992.
- [9] K. Schmidt, Lola -- a low level analyser. In *Nielsen, M. and Simpson, D., editors, Proc. ICATPN, LNCS 1825*, pp. 465 ff. Springer-Verlag, 2000.
- [10] 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.
- [11] E. Grass et al., 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, Dec 2001.
- [12] J. Seizovic, Pipeline Synchronization, *Proc. ASYNC*, pp 87-96, Nov 1994.
- [13] K. van Berkel, F. de Beest, A. Peeters, Adding Synchronous and LSSD Modes to Asynchronous Circuits, *Proc. ASYNC*, pp. 161-170, Apr 2002.
- [14] F. Gürkaynak et al., A Functional Test Methodology for Globally-Asynchronous Locally-Synchronous Systems, *Proc. ASYNC*, pp. 181-189, Apr 2002.
- [15] M. Heath, I. Harris, A Deterministic Globally Asynchronous Locally Synchronous Microprocessor Architecture, *Proc. MTV*, Austin, 2003
- [16] Ø. Damhaug, T. Njølstad, Arbitration And Meta-Stability Management In Globally Asynchronous Locally Synchronous Circuits, *Proc. NORSIG*, 2002, Norway
- [17] K. Yun, D. Dill, Automatic synthesis of extended burst-mode circuits: Part I and II, *IEEE Transactions on Computer-Aided Design*, 18(2), pp. 101-132, Feb 1999.