

Hazard Detection in a GALS Wrapper: a Case Study

Christian Stahl, Wolfgang Reisig
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
stahl,reisig@informatik.hu-berlin.de

Miloš Krstić
IHP Microelectronics
Im Technologiepark 25
15236 Frankfurt (Oder), Germany
krstic@ihp-microelectronics.com

Abstract

An asynchronous wrapper of a fabricated GALS system is analyzed for hazards. For this purpose a Petri net based modelling approach of this GALS wrapper is presented. In our model 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. In order to alleviate state space explosion two techniques to reduce the model's state space are presented. By use of these techniques we detected several potential hazards and a deadlock in the wrapper.

1. Introduction

Globally Asynchronous Locally Synchronous (GALS) systems is an approach for the design of circuits that has been suggested by Chapiro in 1984 [1]. The idea of GALS is to combine the advantages of synchronous and asynchronous design methodologies while avoiding their disadvantages. Usually, a GALS system is defined as a set of *locally synchronous blocks* communicating with each other via *asynchronous wrappers*. Hence, the asynchronous part of a GALS system is limited to its wrappers. A wrapper is a not too large circuit. This reduces conventional problems of asynchronous system design, in particular problems of exhaustive analysis.

Still, an asynchronous wrapper must be analyzed, too. The main challenge in designing a wrapper is to avoid *hazards*. A hazard is an effect where the level of the output signal of a gate changes to an undefined value, due to "too dense" edges on the signal.

Unfortunately, potential hazards are hard to detect and hence, require a complete, formal model of the wrapper. The problem of hazard detection in asynchronous systems is not new. One well known approach uses *Signal Transition Graphs* (STGs), a special Petri net class. By help of the tool *Petrify*, a circuit modelled as an STG is synthesized. In

the end, the model is transformed into a hazard-free representation (see [2]). There are two other approaches that use Petri nets (see [12], for instance): In the *level-based modelling*, the Petri net is built from level-based components such as AND and OR gates. Each signal s is associated with two places, representing its two logical states. In the *event-based modelling*, in contrast, a token in the Petri net represents an edge of a circuit signal s . The firing of a transition models the switching of a logical level of s , but it abstracts from direction of the switching. In a different approach, the authors of [3] successfully verified a GALS system by use of a verification framework, called *process spaces*. A process considers the set of all possible executions, but without referring to their structural details. Hazards then are calculated by a tool called *FIREMAPS*. The authors detected a hazard and other pitfalls in the analyzed system.

Only few GALS systems have been synthesized and fabricated so far. As a case study, in this paper we analyze the asynchronous wrapper of a GALS system that performs a baseband processor [5] for potential hazards. In our approach we are able to preserve the principle structure of the circuit. This structure can be exploited in automatic verification tools. For each gate type of the given wrapper a *Petri net pattern* is built. Our modelling approach is a combination of event-based and level-based modelling. In our model a signal s is represented by a place p with a token on p if an edge occurs at s . We furthermore save the internal states of a gate, i.e. the levels of all signals in the pattern. This way, a pattern preserves all information needed to detect hazards. The wrapper is then just a plugged composition of several instances of those 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. The resulting net is far too large for model checking. But it can decisively be reduced while preserving potential hazards. Besides a basic reduction technique we suggest a further reduction technique: Upon investigating potential hazards at one gate, the behaviour of all other gates may be *simulated*. We suggest a *stepwise, pattern-based abstraction* for this

purpose. This allowed us to detect further potential hazards and a deadlock. Based on our verification results, the wrapper from [5] has significantly been improved.

The remaining structure of the paper is as follows: Section 2 introduces the GALS wrapper. Our modelling approach is presented in Section 3. Section 4 deals with the verification of the model and in particular with state space reduction. In Section 5 we explain our method for hazard detection.

2. The GALS Wrapper

In [4] we suggested a novel request-driven GALS technique. There proposed asynchronous wrappers are deployed in the complex GALS chip as we presented in [5]. This chip performs the baseband processing compliant to the wireless LAN standard IEEE 802.11a. The baseband processor has a datapath architecture with point-to-point communication. The communication between processor blocks is very intensive (w.r.t. the clock cycle of the local clock) but bursty, with longer periods of inactivity. Three important aspects motivated us to apply the novel request-driven GALS technique in the wireless communication environment. 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 goal to avoid unnecessary transitions and the associated waste of energy during the data transfer between GALS blocks. The fabricated GALS baseband is tested and shows superior dynamic power and noise characteristics in comparison with the respective synchronous version of the same processor.

The principle architecture of the used asynchronous wrapper around a locally synchronous (LS) module is shown in Fig. 1. The asynchronous wrapper consists of five components: the *input port*, the *output port*, a *local clock generator*, a *time-out generator*, and a *clock control circuit*. Additionally, input data are buffered in a transparent latch (not shown in Fig. 1). This is needed to prevent metastability at the input of the locally synchronous block. Conceptually, the locally synchronous circuit can be 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.

The proposed wrapper implements the following scenario: When a data burst is being received, the respective GALS block operates in a request-driven mode, i.e. it is possible to synchronize the local clock generators with the request input signal. In this way unnecessary transitions

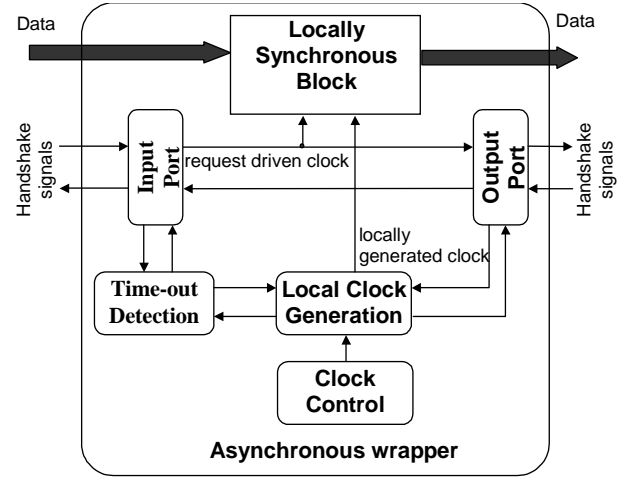


Figure 1. Block diagram of the proposed asynchronous wrapper.

are avoided. However, when the 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 operations 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, but data that has to be processed is still stored in the internal pipeline stages.

When there is no incoming request signal for a certain period of time (defined as a $T_{time-out}$), 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 port. This way we avoid possible energy waste.

More complex and demanding is the scenario when after 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 the clock generation from the local ring oscillator to the input request line. To deal with this situation it is necessary to implement additional circuitry to prevent metastability and hazards in the system.

Let us have a more detailed look at each of the wrapper's components.

The role of the input port is to perform the input and the internal handshake and to grant safe input transfer of the data. Additionally, the input port resets the time-out and the clock control circuitry after every handshake. This port mainly consists of a controller and few supporting gates. The controller must guarantee safe data transfer and it is implemented as an Asynchronous Finite State Machine (AFSM) working in burst mode.

The role of the output port is to safely perform the output handshake of the GALS block. Subsequently until the handshake is finished, appearance of new clock cycle will be disabled. When there is no output data to be transferred, the output port passively acknowledges any internal request. It consists of an AFSM controller and few additional gates.

The time-out generation unit is implemented with a small number of hardware components. Generally it consists of a counter for negative edges of the local clock. This counter is designed as a standard synchronous counter. When reaching its final value it eventually generates a time-out signal. On the other hand, the counter's reset signal is activated once during every input port handshake. Therefore, time-out signals can be generated only when the input handshake channel has been inactive sufficiently long.

The local clock generator (LCG) triggers the time-out measurement. Additionally, when the time-out is reached it generates clocks for a LS block. A LCG can be stretched from both input and output port. The LCG is implemented as a ring oscillator and the structure of the LCG is described in [7]. Generally, a LCG consists of a delay line, a C-element, an arbitration section and one NOR-gate for enable/disable function. A delayline is designed in such a way that the tunability of the clock generator is asserted. Tunability is a very important property of the proposed LCG in order to calibrate the clock frequency and to avoid the effect of changes of processes, temperature, or voltage.

To increase power-saving capabilities, a clock control block is designed. The role of this block is to count the number of locally generated clock cycles. When the LS pipeline is empty, it disables the local clock generation.

In order to guarantee safe data-flow in the GALS system we must be certain that the wrapper behaves hazard- and glitch-free. However, this is not possible without complete formal analysis of the asynchronous wrapper. Details of this approach will be given in following sections. The analysis results helped us to improve the reliability of the asynchronous wrapper.

3. A Petri Net Based Modelling Approach for Analyzing Hazards

3.1. Petri Net Models of Wrapper Gates

Proof of the asynchronous wrapper's correctness requires a *formal model* of the wrapper. To this end, a modelling technique is required that is, in particular, capable of representing the interplay of gates, signals, signal levels, and signal edges.

We have chosen *Place/Transition nets*, i.e. nets where each place can carry any number of tokens. Readers not familiar with this notion are referred to [8], for instance. Places will be used to represent two different aspects of gates:

Firstly, each signal s is assigned a place p . A token on p represents an *edge* on s . The token does not show whether the edge is rising or falling. Places of this kind will be denoted as *edge places*.

The second aspect to be represented by places, are the actual *levels* of signals. Receiving a signal edge, a gate may react in different ways, depending on the actual level of a fixed set of signals. To capture this behaviour properly, the Petri net model of a gate has a number of places, representing potential combinations of the level of some signals. Those places will be denoted as *level places*.

In fact, the idea of level and edge places is not new (see [12]). Level-based modelling has been used for representing circuits built from level-based components, e.g. AND gates and OR gates. The technique of event-based modelling has been applied for modelling event-based components, for instance XOR gates and Muller C-elements (MCEs).

Our considered wrapper is composed of quite a number of instances of gates. Each gate is either a logic gate, a flip-flop, a counter, a Mutex, or a MCE. Thus, we have both level-based and event-based gates. In contrast, to model a flip-flop or a counter the logical level *and* the edge switching of each signal is needed. For a unique structure of our patterns, we suggest to combine both ideas in all types of gates, i.e. an AND gate pattern, for instance, has both level and edge places. As we will see later on, this unique structure simplifies the composition of patterns as well as their abstraction.

In this manner, we constructed a Petri net pattern for each gate. The wrapper's Petri net model is then just composed from instances of the corresponding patterns. In general, a pattern is depicted as a dashed box. Inside the box, the structure of the corresponding gate is modelled. Outside the box, the nodes of the interface are depicted. The interface is represented by the pattern's edge places. Its level places, in contrast, are part of the pattern's structure. The following two sections describe two such patterns in detail.

3.2. The Pattern for an AND Gate

We start with a representation of the AND gate. An AND gate consists of three signals, a , b , and c , such that $c = a \wedge b$. The AND gate updates (i.e. re-computes the level of) c whenever a or b have been updated (i.e. have been given new levels from outside the gate).

The AND gate is represented by the help of four *internal states*, representing the combinations of the levels of the signals a and b , i.e. the four logic gate levels. The actual state may change due to the change of the actual level of one of the signals a and b by the occurrence of an edge at one of the two signals. Obviously we do not need to save the level of output signal c , because it can be calculated by the given levels of input signals a and b .

Fig. 2 shows our Petri net model of the AND gate: Its internal states are represented by the level places ab , $\bar{a}b$, $a\bar{b}$ and $\bar{a}\bar{b}$. The place ab represents the state where both levels of signals a and b are high. If the place $\bar{a}b$ is taken, the level of a is low and the level of b is high. $a\bar{b}$ and $\bar{a}\bar{b}$ are now obvious. In every reachable marking, exactly one of the four level places carries a token.

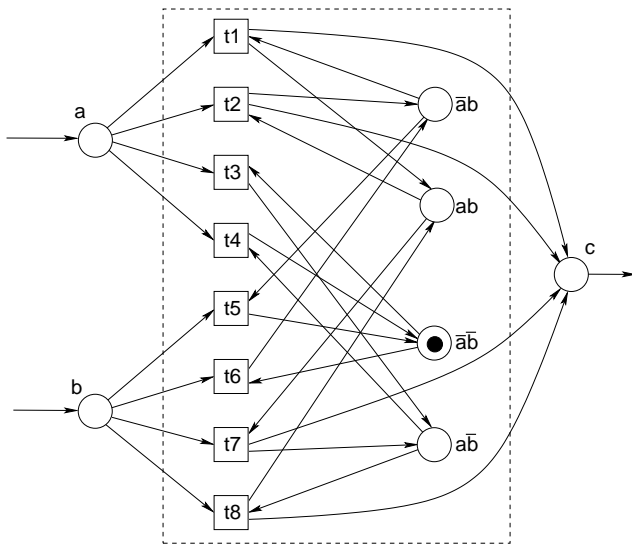


Figure 2. The AND pattern.

Fig. 2 exhibits three more places, a , b , and c , the edge places. Place a represents the potential (enforced) edge change at the signal a : A token on place a represents an edge which indicates that the actual level of signal a at the AND gate must change. A token on place a activates one of the four upper transitions. For example, a token on the place $\bar{a}b$ together with a token on a , activates the upmost transition, $t1$. In this situation the level of a is low and the level of b is high, hence, the level of c is low. The token on place a indicates that the level of signal a is due to swap, concretely, to change from low to high. As the levels of a

and b are now both high, the level of c must turn high, too. Thus, occurrence of $t1$ in Fig. 2 yields a token on c , indicating a change of the level of c from low to high, i.e. a rising edge. As a further example transition $t4$ transforms the state $\bar{a}\bar{b}$ into the state $a\bar{b}$. Occurrence of $t4$ implies that the level of c remains low. Hence, there is no arc from $t4$ to c .

An edge place is either *input edge place* or *output edge place*. In Fig. 2 places a and b are input edge places and place c is an output edge place.

3.3. The Pattern for a Mutex Gate

Next we present the pattern of the Mutex gate shown in Fig. 3. A Mutex gate consists of four signals a , b , a_out , and b_out . For each input signal a and b , respectively, there exist one output signal a_out and b_out , respectively. When both input signals are low, both output signals are low, too. When now one of the input signal's level changes to high, the level of the corresponding output signal changes to high, too and the level of the other output signal cannot change to high as long as the first output signal is high. If the level of both input signals change to high simultaneously, the circuit tosses a coin to decide which of the output signals changes to high. In other words, Mutex guarantees that the levels of both signals a_out and b_out are never high at the same time.

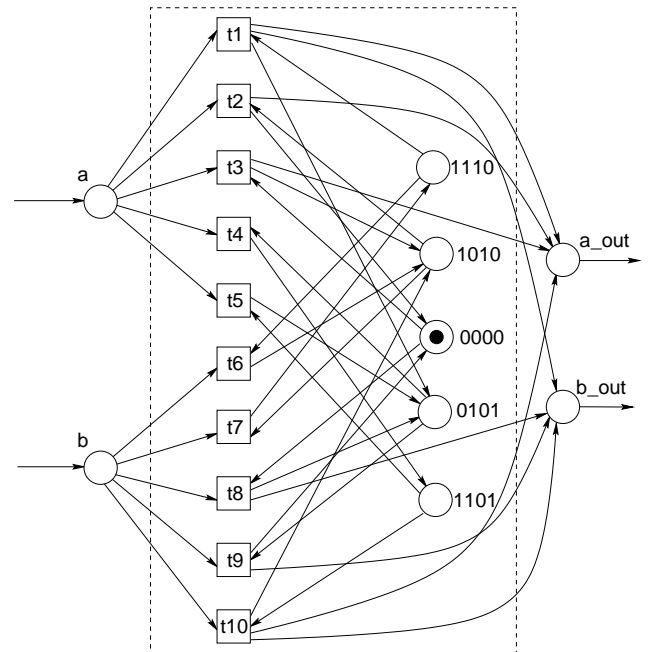


Figure 3. The Mutex pattern.

The Mutex gate is represented by the help of five internal states, representing the possible combinations of the levels of the signals a , b , a_out , and b_out . As already explained

in Section 3.2, the actual state may change due to the change of the actual level of one of the signals a or b by the occurrence of an edge of one of the two signals. In contrast to the AND pattern, it is not sufficient to save only the level of the two input signals. We must also save the levels of both output signals in order to distinguish between the two possibilities if the level of both input signals is high, because in this scenario we have to block the level change of one of the output signals.

The pattern of the Mutex gate shown in Fig. 3 is similar to the AND pattern presented in Fig. 2. Its internal states are represented by the places 1110, 1010, 0000, 0101, and 1101, where the numbers 0 and 1 visualize the level of a , b , a_out , and b_out , respectively.

Proper behaviour of the Mutex pattern can be studied by considering the 10 transitions one by one. This is left out.

3.4. Patterns for the Other Element Types

The remaining gate types, i.e. logic gates, flip-flop, counter, and MCE, are likewise assigned patterns of Place/Transition nets, following the principles already applied in Section 3.2 and 3.3: Each gate has its set of input and output signals, with each signal s represented as a Petri net place, p . A token on p represents an edge at signal s , where the gate is enforced to change the level of signal s . Hence, the gate's Petri net pattern is expected to contain an enabled transition, consuming the token. The occurrence of that transition changes the internal state of the pattern represented as a token on a level place and it may also produce new tokens on places representing the gate's output signals.

3.5. Potential Hazards

A hazard occurs in a gate if two preconditions match: Firstly, *structural conditions*: a gate's structure allows for local states that in case of "unsound" input signals allow for a confusing result. Secondly, *temporal conditions*: it is possible to change the level of more than one signal in a short timing interval, i.e. edges of at least two different signals occur at nearly the same time.

As an example consider the AND gate of Fig. 2 in state $\bar{a}b$. This gate meets the structural condition, because if a falling edge occurs at a and afterwards a rising edge at b , it causes no edge at c . In contrast, if both edges occur the other way round, c changes first to high and then to low. The AND gate also meets the temporal condition, because it is possible that the edges of signal a and b occur in a very short timing interval. As a further example, look at the Mutex gate. This gate meets the temporal condition, too. The Mutex gate, in contrast, never meets the structural condition, because it has a different structure than the AND gate.

Having this in mind, we can define the following criteria of a potential hazard in a pattern: Let G be a pattern and let M be a marking of G where each input edge place and no output edge place carries a token. Let M' be reachable from M with an output edge place carrying more than one token. Then the pattern G is *hazard prone*. Thus, a *hazard* is characterized by help of markings M with more than one token on an output edge place.

Accordingly in our patterns, detection of a hazard is reduced to the reachability of a marking.

As an example for the AND pattern of Fig. 2, assume tokens on a , b , and $\bar{a}b$. Then occurrence of transitions $t1$ and $t7$ yields two tokens on c .

3.6. Composition of Patterns

The GALS wrapper can be modelled by composing a number of instances of the 7 gate patterns. This results in a Petri net with about 288 places and 526 transitions. Each reachable state of the wrapper is represented as a reachable marking of this Petri net. Vice versa, a number of markings reachable in the Petri net does *not* correspond to reachable states of the wrapper. Those states are not reachable due to the gate delays.

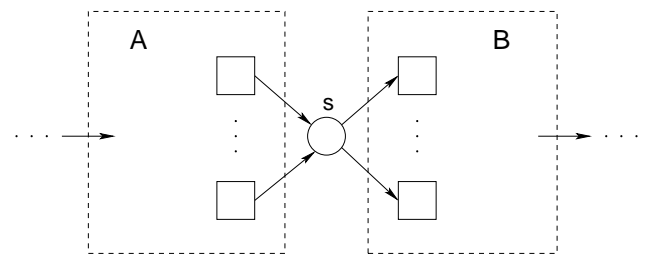


Figure 4. Gates A and B, linked by signal s .

As an example, assume two gates A and B , with a signal s obtaining edges from A that are consumed by B as depicted in Fig. 4. Gate A is assumed to be hazard-free. Furthermore, assume each switching of B takes less time than any switching of A . Consequently, each edge rising at s is consumed by B before the next edge appears at s . In terms of the model, this would mean that no reachable marking has two tokens on s . But our model does not respect delay times so far. Consequently, a marking M with two tokens on s is reachable and M is no hazard marking. So the set of reachable markings of the Petri net model could drastically be reduced if we would respect delay times of the gates. This in fact would be feasible by help of *timed Petri nets*. We applied an entirely different approach, however. It is based on the observation that the designer of the wrapper exploits delay times not in order to guarantee any real-time behaviour, but just to avoid "dangerously close" edges on one signal. As we want the model only to detect hazards,

we may abstract away any real-time aspects, and just model the assumption that delay times guarantee absence of “close edges” on a signal. This is easily achieved in the Petri net model by help of the well-known concept of *complement places*. We extend the model by the complement places, \bar{s} , for each edge place, s . Fig. 5 outlines this construct for Fig. 4. As a consequence, the number of places of the wrapper model increased from 288 to 363.

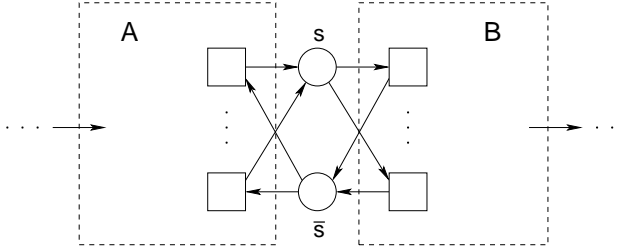


Figure 5. Gates A and B, linked by signal s and its complement place.

The resulting net’s reachable markings still represent all reachable states of the wrapper. Hazards are no longer represented by two tokens on an output edge place, but by markings M^* where two pre-transitions of an output edge place carry all necessary tokens to fire, except for the introduced complement place. The complement place, of course, allow only one of them to fire.

As an example for the AND pattern in Fig. 2 the marking with a token on a , b , $\bar{a}\bar{b}$, and \bar{c} and the marking with a token on a , b , $\bar{a}b$, and \bar{c} where \bar{c} is the complement place of c are the hazard markings.

3.7. Verification

As shown in Section 3.6, the detection of a potential hazard of the wrapper is reduced to the reachability problem of some distinguished markings in the Petri net model. Reachability is a typical problem to be tackled by a model checker. We employed *LoLA*, an explicit model checker [10]. Its features include powerful reduction techniques such as partial order reduction [9] and the sweep-line method [11]. *LoLA* reduces the wrapper model from 363 places to 243 places. Still, the space of potentially reachable states has the magnitude of about 2^{243} elements, far too many to be tackled by conventional PC technology.

LoLA provides techniques to solicit short paths to given reachable markings. Those techniques have successfully been applied to detect some potential hazards.

Exhaustive search of the state space, however, has not been feasible with *LoLA*.

As an alternative, we tried the *SMV* model checker [6] that is very popular for hardware verification. But we failed

again, due to the state space explosion. This remained also after *SMV*-tuned simplification of the model.

4. State Space Reduction Techniques

As the model’s state space exceeds any model checker’s capacity, further reductions of the model were useful, provided they preserve hazards. We suggest two such techniques in this section.

4.1. Hazard-Preserving Abstraction Technique

The state space of a model of the GALS wrapper can be reduced by *integration* of several instances of its logic gates. Fig. 6 shows an example, composed from two AND gates and one OR gate which is part of the wrapper’s input port. Though we did not show the internal structure of OR gates, it should be fairly obvious that an OR gate has two input signals and one output signal, and that an edge will occur at signal e whenever one of the AND gates has switched and e ’s level is changed.

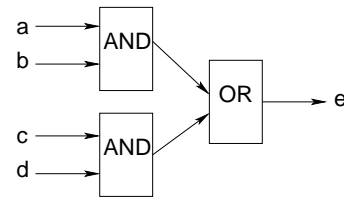


Figure 6. Sequential logic gates.

The idea is to integrate the three gates of Fig. 6 into one pattern, as outlined in Fig. 7. We refrain from detailing the pattern of Fig. 7. It suffices to know that each occurrence of an edge of signal a , b , c , or d causes the firing of exactly one transition.

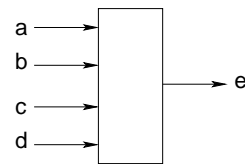


Figure 7. Integrated gates.

The circuit in Fig. 7 preserves all hazards of the circuit in Fig. 6 including hazards at the outputs of both AND gates, because it consists of all level and edge places of signals a , b , c , and d . Thus, the hazard markings can still be calculated.

The circuit of Fig. 6 includes $3 \cdot 8 = 24$ transitions and $6 + 1 + 3 \cdot 4 = 19$ places (6 input, 1 output and 12 level

places). To change the level of signal e , at least two transitions have to occur: Firstly, the level of one of both AND gates has to change. Secondly, the level of signal e has to change. The pattern in Fig. 7 consists of $4 \cdot 16 = 64$ transitions and $4 + 1 + 2^4 = 21$ places (4 input, 1 output and 16 level places). This does not seem to be much of an improvement at first glance, because the number of places and transitions increased. But in the context of tool based analysis, the bottleneck is not the number of places or transitions, but the number of *states*, more precisely, the number of *intermediate states*. And the latter is drastically reduced as each change of signal e 's level implies that at least one transition fires.

Integrating the three gates of Fig. 6 reduces the number of states in the input port from 4,621,595 to 2,133,526 by use of the model checker LoLA. In this run LoLA combined partial order reduction and the sweep-line method.

We applied the above technique to seven components of the wrapper. This reduced the entire wrapper model from 363 places and 526 transitions to 278 places and 493 transitions.

The full state space of the reduced model is still too large for being handled by LoLA or by SMV.

4.2. Non-Hazard-Preserving Abstraction Technique

In the last section we presented a technique that firstly reduces the state space of the model and secondly preserves all potential hazards. Unfortunately, the state space was too large for being handled by LoLA or by SMV, so, it was inevitable to search for additional reduction techniques.

As mentioned in Section 2, the wrapper consists of five subcircuits. Table 1 shows the number of places and transitions as well as the state space for each subcircuit calculated by LoLA using partial order reduction and the sweep-line method. If we searched for a hazard in a specific gate G , only the level information of G is needed. From G 's environment we only have to preserve the interface and the signal flow, more precisely, the level change of the signals. Thus, we can build a simplified, more precisely, an abstract environment for gate G which only preserves the interface and the level changes. Finally, the wrapper is composed by plugging four abstract subcircuits (the environment) and one concrete subcircuit, whereas the latter embeds gate G which has to be checked for hazards. By flexible change of abstract and concrete patterns of the five subcircuits we can search for every potential hazard. We only need five wrapper models instead of one.

Before the following three paragraphs present our step-wise abstraction method we first of all have a more detailed look at one of the wrappers subcircuits, the LCG, generating the local clock, as shown in Fig. 8. As initial state of the

subcircuit	places	transitions	states
input port	85	147	81,352
time-out generator	67	96	2,603
output port	71	106	201
local clock generator	38	92	186
clock control	24	29	26

Table 1. State space of the wrapper's subcircuits.

LCG, assume the *STOP1* signal high, and all other signals low. The clock is now triggered by *STOP1* falling low. Due to the functionality of the involved gates, NOR raises *RCLK* to high. Then both Mutex gates switch, followed by AND and MCE. This completes a cycle, now starting its second round by switching the NOR gate now with the effect of lowering the *RCLK* signal.

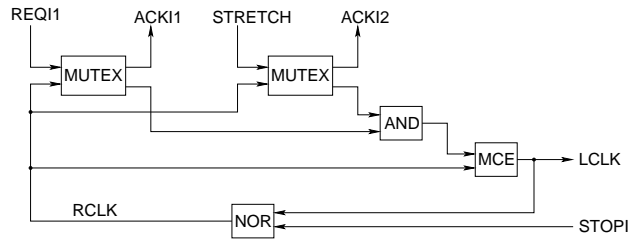


Figure 8. Simplified block structure of the LCG.

This circuit is apt to stretching the clock phase. This is achieved by rising edges at *REQ11* or *STRETCH*. The effect of those edges is to block (i.e. to delay) the change of signal *RCLK*'s level from low to high.

Abstract Patterns In the first step of our simplification method we built abstract patterns for each subcircuit of the wrapper. An abstract subcircuit is the composition of abstract gate patterns plugged together. In such an abstract gate pattern we tried to avoid the use of level places whenever possible. As a consequence, potential hazards inside this subcircuit cannot be detected. This pattern only preserves the interface and the level changes of the signals. In the following we show an example of an abstract pattern and further how we proved its correctness concerning its concrete pattern.

A formal proof by help of the LoLA analysis tool revealed that the two transitions $t5$ and $t6$ of both instances of the Mutex pattern are never enabled in the LCG shown in Fig. 8. Hence, the pattern of Fig. 3 can be replaced by the abstract pattern of Fig. 9 without affecting the Mutex gate's behaviour. The inscriptions of the internal states of the pat-

tern of Fig. 9 refer to the level of the output signals a_out and b_out .

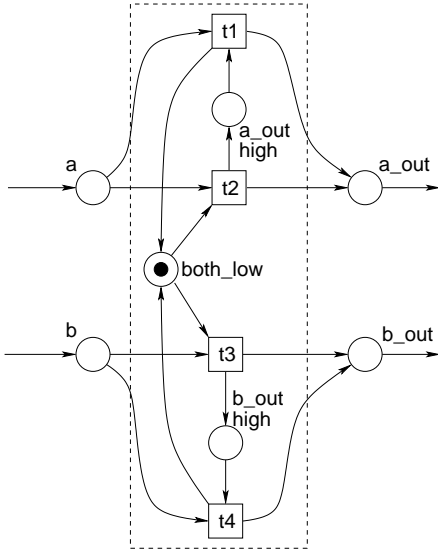


Figure 9. Abstract Mutex pattern.

Replacing both Mutex patterns of Fig. 3 with its abstract counterpart of Fig. 9, reduces the set of places from 48 to 44 and the set of transitions from 52 to 40. The state space is reduced from 2440 states to 1980 states (full state space). In this manner we also built abstract patterns for the other gate types.

When building abstract patterns it is necessary to prove whether the abstract pattern behaves like the concrete pattern. For that purpose the abstract pattern has to change the level of its output signals whenever the concrete pattern does. Informally spoken, for each transition that can occur in the concrete pattern in Fig. 3 we have to find one or more transition in the abstract pattern in Fig. 9. The latter pattern has to behave like the former. The transitions of both patterns then form a relation, known as a *simulation*. If there is a transition in the concrete pattern which has no counterpart in the abstract pattern, it can be expressed by an internal transition. This is a τ -step. Hence, the relation is a *weak simulation*.

Table 2 shows the relation of the concrete and abstract Mutex pattern. Note the two transitions in the concrete Mutex ($t5$, $t6$) that could not be enabled are not considered. The table demonstrates that the concrete Mutex model is weakly simulated by the abstract Mutex model (because of the τ -steps). Thus, Fig. 9 is a correct abstraction of Fig. 3 and it preserves the level change of all signals. In fact, we built an abstract LCG that weakly simulates the concrete pattern of the LCG. This abstract LCG reduces the set of places from 48 to 32 and the set of transitions from 52 to 16. In contrast to 2440 states in the concrete LCG the ab-

stract LCG reduces the state space to 112 states which is a reduction of more than 95%. By use of partial order reduction and the sweep-line method the number of states is reduced from 186 (see Table 1) to 6.

concrete Mutex	abstract Mutex
t1	t1, t3
t2	t1
t3	t2
t4	τ
t7	τ
t8	t3
t9	t4
t10	t4, t2

Table 2. Weak simulation between concrete and abstract Mutex pattern.

In addition to the LCG, we also simplified input port, output port, time-out generator, and clock control. To check for example the input port for potential hazards, the concrete pattern of the input port and the four abstract patterns of the remaining subcircuits are composed. The resulting wrapper is reduced to 143 places and 177 transitions. The full state space could not be calculated by LoLA, but by SMV. Consequently, we went on searching for stronger abstraction.

Abstraction of the Subcircuit's Internal Behaviour In the last step we built abstract patterns for each gate type. A subcircuit is a composition of abstract gate patterns. Now we build an abstract pattern for a complete subcircuit. Such an abstract pattern must preserve the interface and the flow of signals of its corresponding concrete pattern. More detailed, every signal which arrives at (resp. leaves) the concrete subcircuit must arrive at (resp. leave) the abstract subcircuit. Then there is no need to preserve the inner structure, i.e. the gate structure of the respective subcircuit. Thus, the subcircuit's inner structure can be compressed.

We draft this procedure exemplarily for the LCG in Fig. 8: The subcircuit is triggered by $STOPI$ falling low. Then rising and falling edges of $LCLK$ alternate until $STOPI$ rises. The clock is stretched, if at least one of the signals $STRETCH$ or $REQ11$ rises. This is the behaviour of the LCG which can be in fact modelled simpler if no gate patterns are used. The resulting pattern is not shown in this paper due to its size. In this pattern the number of places is reduced from 32 to 16 and the number of transitions from 16 to 12. The full state space of this pattern consists now of 80 states.

In addition to the LCG, we also applied this technique to all other wrapper's components. To prove the correctness

of the abstraction, we used the technique of weak simulation. Table 3 shows for every abstract subcircuit the state space calculated by LoLA using partial order reduction and the sweep-line method. Comparing these results to those of Table 1, we notice for every pattern except the clock control a strong reduction in the number of places, transitions and states. The values of the clock control are the same in both tables, because we did not find any simplification of its gate structure. This justifies the first abstraction step.

subcircuit	places	transitions	states
input port	71	80	4,644
time-out generator	20	16	11
output port	20	14	30
local clock generator	16	12	7
clock control	24	29	26

Table 3. State space of the wrapper’s abstract subcircuits.

Again, we built a wrapper to check for hazards in the input port. The resulting wrapper consists of 142 places and 133 transitions. The state space could be fully explored by LoLA using partial order reduction and the sweep-line method. It consists of 3,013,819 states.

Interface Abstraction For every subcircuit we built an abstract pattern by preserving its interface and constructing an inner structure which models the behaviour of the respective subcircuit independent of its gate structure. Hence, the environment is only the plugged composition of four abstract subcircuit patterns. The next step of abstraction is similar to the former one. The environment is no longer considered as a plugged composition of patterns, but as a single pattern. So we further abstract from the environment’s inner structure, i.e. its subcircuits, and only preserve its interface to the concrete pattern which embeds gate G which has to be checked for hazards. More detailed, the interfaces and the inner structures of its four subcircuits can be compressed while the environment’s signal flow and its interface must be preserved.

We only used this abstraction step to build a wrapper with only abstract subcircuits for composing multiple wrappers. The wrapper consists of 76 places and 46 transitions and the state space reduces to 385,000 states (full state space) and 25,600 states (reduced state space), respectively.

The technique presented in Section 4.1 preserves all potential hazards in the model. In contrast, the use of abstract patterns as presented in Section 4.2 only preserves all potential hazards in a concrete subcircuit. Nevertheless, all potential hazards of the whole wrapper can be detected by replacing a concrete subcircuit by its abstract counterpart and vice versa. As already mentioned, this implies five wrapper

models instead of one, but each model with a smaller state space.

In the next section, we will give more detailed information about our analysis procedure.

5. Hazard Detection

We presented a method to reduce the state space of a Petri net model of a GALS wrapper in the last section. In the following we present a very pragmatical method to detect hazards for a given model.

The hazard detection procedure works as follows: As a first step the set of “candidates structures” for hazards is identified. Candidates are the logic gates, flip-flops, and counters. Mutex and MCEs are no candidates; they are hazard-free due to their distinguished structure.

In the second step all candidate structures S are considered: Each S has a set of hazard markings similar as defined exemplarily for the AND gate in Section 3.6. The problem is to decide whether or not the markings are reachable. We solved this problem by help of the model checking tool LoLA: LoLA detects whether or not a hazard marking M of S is reachable. If reachable, LoLA constructs a *witness*, i.e. an occurrence sequence from the initial marking to M . Each witness sequence corresponds to a sequence of signal edges s of the wrapper.

In a next step we tried to simulate s with the help of simulating the wrapper’s VHDL code, in order to confirm whether any “dangerous” state transition can occur in reality. Additionally, we searched for auxiliary constraints of that potential hazard which have to be valid if the hazard reflects reality, e.g. “the controller of the input port must be in state 2”. With these constraints the hazard marking was enhanced and a new model checking run could be started to get a more precise witness. While doing this iteration, most of the witnesses could be excluded, because they violate known timing constraints, for instance. As the wrapper is built from the composition of four abstract and one concrete subcircuit pattern (as described in the 2nd paragraph in Section 4.2) at most 3 – 4 iterations were needed.

As a result we detected a number of hazards in our model of the GALS wrapper: We detected a hazard in the input port and some potential hazards in the time-out generator. Additionally, we noticed that in the local clock generation mode we can have potential hazards in the input port and output port under very specific circumstances. Finally, we observed potentially hazardous behaviour of the data-latch enable signal. Fortunately, in the usual application scenario for GALS blocks no reachable state generates any of the detected potential hazards. Nevertheless, in order to increase reliability of the asynchronous wrapper we have fixed all those potential hazards in the system. Without the formal verification this would not have been feasible.

We also used the model of the wrapper to check for deadlocks. As the our simplification technique described in Section 4.2 preserves the flow of signals, deadlocks in the model are preserved, too. Due to a nondeterministic choice in the controller of the input port a deadlock could occur. By knowing its cause the deadlock could be easily fixed.

6. Conclusions

In this paper, we suggested a pattern-based approach for modelling an asynchronous wrapper of a fabricated GALS system. We used Petri nets as modelling technique. The model preserves all information that is necessary to detect hazards. Due to the design of our patterns the question whether a potential hazard can occur in a gate is reduced to the reachability of a particular marking. In fact, the latter is a well known model checking problem. To solve this problem, we used two different model checkers – LoLA, an explicit model checking tool and the symbolic model checker SMV. In order to alleviate state space explosion we presented two techniques to reduce the model’s state space. These techniques allowed us to detect several potential hazards and a deadlock which could all be fixed in an improved version of the wrapper.

From this case study, we have learned that the use of patterns on different levels of abstraction is very useful and helps reducing the model and also the state space. Our pattern-based modelling approach furthermore simplifies the construction of the wrapper’s model, because it reflects the wrapper’s gate-level and not for example its behaviour. As analysis and improvement of the wrapper formed an iterative process, it was important to easily exchange a subcircuit’s pattern by the pattern of the improved subcircuit. On the one hand the unique construction of the patterns causes a higher number of places in the model. For example, our AND pattern consists of ten places whereas an AND pattern in the level-based approach only has two places for each signal, i.e. six places. On the other hand by the use of pattern-based abstraction the model’s size is reduced successfully. LoLA and SMV proved useful in finding witnesses quickly. LoLA finds witnesses even without being capable of storing all of the reduced state space. This is due to its powerful reduction techniques, in particular the sweep-line method and partial order reduction. Using these two techniques together instead of using only partial order reduction, the state space could be reduced a factor of 2 – 10. In particular, the recently proposed sweep-line method performed very well. Our results furthermore show that explicit model checking techniques become quite powerful and a stronger competition to symbolic model checkers in the domain of hardware verification.

Further work includes ongoing analysis of the improved wrapper. We will furthermore try to find a better abstraction

of the patterns to minimize the state space. It seems to be possible to build wrappers that consist of only one concrete gate G which has to be checked for hazards instead of a concrete subcircuit.

Acknowledgements

This work is part of the project “GALS-Design” supported by Deutsche Forschungsgemeinschaft (German Research Council). The authors thank Jan Bretschneider, Alexandra Julius and Karsten Schmidt for their suggestions and help during the verification attempt.

References

- [1] D. M. Chapiro. *Globally-asynchronous locally-synchronous systems*. PhD thesis, Stanford University, 1984.
- [2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer Verlag, 2002.
- [3] X. Kong, R. Negulescu, and L. W. Ying. Refinement-based formal verification of asynchronous wrappers for independently clocked domains in systems on chip. In *Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 370–385. Springer-Verlag, 2001.
- [4] M. Krstić and E. Grass. New GALS Technique for Datapath Architectures. In J. Juan-Chico and E. Macii, editors, *PATMOS*, volume 2799 of *LNCS*, pages 161–170. Springer, September 2003.
- [5] M. Krstić and E. Grass. GALSification of IEEE 802.11a Baseband Processor. In E. Macii, O. G. Koufopavlou, and V. Paliouras, editors, *PATMOS*, volume 3254 of *LNCS*, pages 258–267. Springer, September 2004.
- [6] K. McMillan. *Symbolic model checking: an approach to the state explosion problem*. Kluwer Academic, 1993.
- [7] J. Mutersbach. *Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems*. PhD thesis, ETH Zurich Switzerland, 2001.
- [8] W. Reisig. *Petri Nets*. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, EATCS Monographs on Theoretical Computer Science edition, 1985.
- [9] K. Schmidt. Stubborn set for standard properties. In *Proc. 20th Int. Conf. Application and Theory of Petri nets*, volume 1639 of *LNCS*, pages 46–65. Springer-Verlag, 1999.
- [10] 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, page 465 ff. Springer-Verlag, 2000.
- [11] K. Schmidt. Automated generation of a progress measure for the sweep-line method. In *Proc. 10th Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2988 of *LNCS*, pages 192–204. Springer-Verlag, 2004.
- [12] A. Yakovlev and A. M. Koelmans. Petri Nets and Digital Hardware Design. *LNCS: Lectures on Petri Nets II: Applications*, 1492, 1998.