Hazard Detection in a GALS Wrapper: a Case study

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

Wolfgang Reisig
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
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 three techniques to reduce the model's state space are presented. By use of these techniques we detected several potential hazards 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 tests.

Still, an asynchronous wrapper must be tested, 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 a STG is synthesized. In the end the model is transformed into a hazard-free representation (see [2]). In a different approach, the authors of [5] 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 several hazards 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 [7]. To verify the given GALS wrapper for potential hazards, we firstly tried to transform our wrapper into a STG by use of tool support, but we failed. To generate the STG by hand also failed, because the wrapper has too many signals. As a result we decided for a different approach. For each gate type of the given wrapper a *Petri net pattern* is built. 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 some basic reduction techniques we suggest a further reduction technique: Upon investigating potential hazards at one gate, the behavior of all other gates may be *simulated*. We suggest *abstract patterns* for this purpose. This allowed us to detect further potential hazards. Based on the potential hazards we detected, the wrapper 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. GALS Wrapper

In [6] we suggested a novel request-driven GALS technique. There proposed asynchronous wrappers are deployed in the complex GALS chip as we presented in [7]. This chip performs the baseband processing compliant to the wireless LAN standard IEEE 802.11a. The baseband processor has 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 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 module is shown in Fig. 1. 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 are avoided. However, when the input burst is received

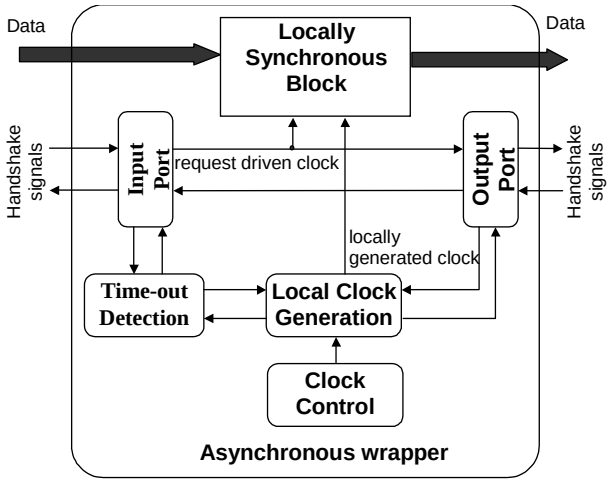

Figure 1. Block diagram of the proposed asynchronous wrapper

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 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 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.

The asynchronous wrapper consists of the *input port*, the *output port*, *local clock generator*, *time-out generator*, and a *clock control circuit*, as can be seen from Fig. 1. Additionally, input data are buffered in a transparent latch. This

is needed to prevent metastability at the input of the locally synchronous block.

The role of the input port is to perform the input and internal handshake and to grant safe input transfer of the data. Additionally, the input port resets time-out and clock control circuitry after every handshake. This port mainly consists of an input controller and few supporting gates. The input 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 output controller and few additional gates.

The time-out generation unit is implemented with a small number of hardware components. Generally it consists of one counter that counts the number of negative edges of the local clock. This counter is designed as a standard synchronous counter. When reaching its final value eventually it 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.

A local clock generator (LCG) triggers the time-out measurement. Additionally, when time-out is reached it generates clocks for a LS block. A LCG could be stretched from both input and output ports. The local clock generator is implemented as a ring oscillator and the structure of the LCG is described in [9]. Generally, a LCG consists of delay line, C-element, arbitration section and one NOR-gate for enable/disable function. A delayline is designed in such way that the tunability of the clock generator is granted. 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, the local clock generation is disabled.

In order to grant safe data-flow in 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 and 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 to represent the interplay of gates, signals, signal levels and signal edges.

We have chosen *elementary Petri nets*, i.e. nets where each place can carry at most one token. Readers not familiar with this notion are referred to [10], for instance. Places will be used to represent two different aspects of gates: Firstly each signal s is assigned a place p. One may expect a token on p to represent a value of s, e.g. the value *high*, and correspondingly no token to represent *low*. We will however interpret places and tokens fundamentally different: 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 behavior 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*. Details will become obvious in the forthcoming sections.

The idea of edge places has been suggested by Gomm in [4] already, and employed to some state independent properties of gates, in particular of Mutex and Muller C-elements. Level places have been used by Genrich and Shapiro, to model and verify an arbiter cascade [3]. We suggest to combine both ideas. This will allow to characterize hazards.

The wrapper is composed of quite a number of instances of gates. Each gate is a logic gate, a flip-flop, a counter, a Mutex or a Muller C-element. 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. The following two sections describe two such patterns in detail.

3.2. The pattern for the 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). 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 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

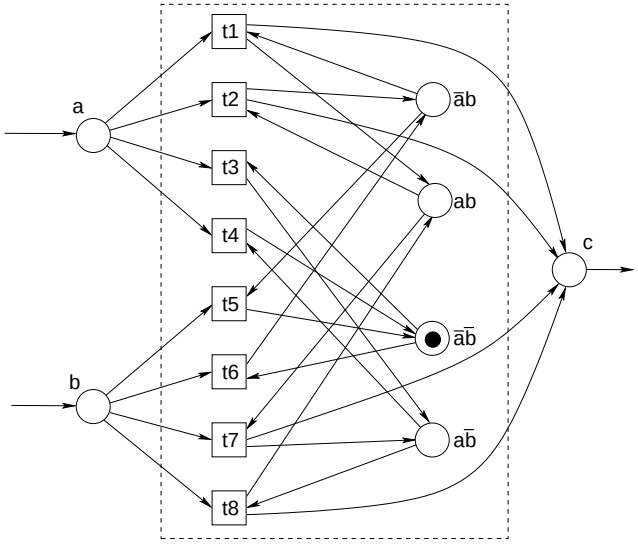

Figure 2. AND pattern

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.

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 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 both a and b are 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 $a\bar{b}$. Both states imply that the level of c is low. Hence, there is no arc from $t4$ to c.

The pattern of Fig. 2 is intended to receive signal edges on places a and b, and to potentially produce edges on place c. Therefore, a and b are *input edge places*, and c is an *output edge place* of the AND pattern. In technical terms an input edge place p has arcs from p to some transition of the pattern; an output edge place, q, has arcs from transitions of the pattern to q.

3.3. The pattern for the Mutex gate

Next we present the pattern of the Mutex gate. 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.

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 occur-

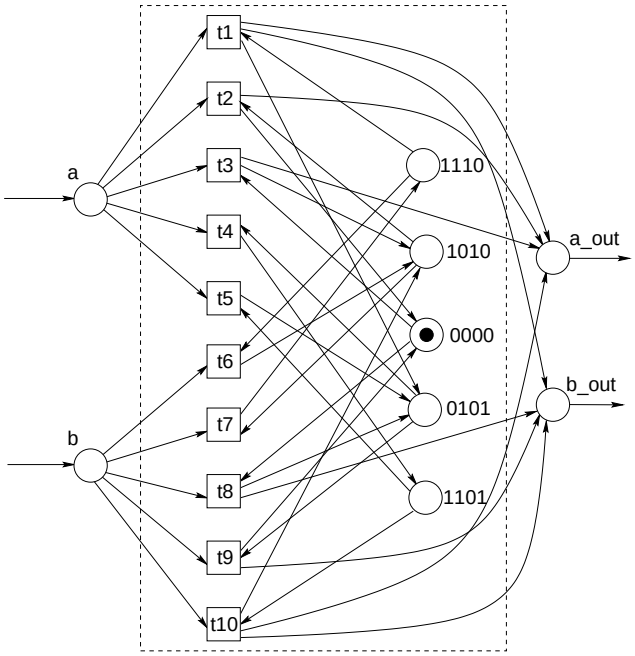

Figure 3. Mutex pattern

rence 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.

Fig. 3 shows the pattern of the Mutex gate. It 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 behavior of the Mutex pattern can be studied by considering the 10 transitions one by one. We leave this as an exercise to the reader.

3.4. Patterns for the other element types

In addition to the AND gate and the Mutex gate, the remaining gate types, i.e. logic gates, flip-flop, counter and Muller C-element, are likewise assigned patterns of elementary Petri 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 in case 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. For example, the AND gate has this kind of local states, whereas the Mutex gate has no such structure. 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. For example, in the AND gate it is possible that the edges of signal a and b occur in a very short timing interval.

Having this in mind we can define the following criteria of a potential hazard in a pattern:

Definition 1 (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*.

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.

We characterized hazardous states of gates by help of markings M with two tokens on an output edge place. This is intuitively reasonable, because this denotes a signal without sufficient delay between two edges. Instead of characterizing hazards by such markings M, one characterize them by the markings M' that may *cause* M. For example, the marking M' of the AND pattern with a token on a, b and $\bar{a}b$ with $t1$ followed by $t7$ leads to a marking with two tokens on c. This likewise applies to M'' with tokens on a, b and $a\bar{b}$. Hence, a hazard can be identified in a pattern by markings $M*$ that *may* cause two tokens on an output edge place. This in mind, we now turn to markings with two tokens on a output edge place, caused by "normal" behavior of gates. For each gate, those markings $M*$ can uniquely be identified.

3.6. Reducing the state space of reachable markings

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 is reachable in the Petri net that would *not* correspond to reachable states of the wrapper. Those states are not reachable due to relations among the gate delays.

As an example, assume two gates A and B, with a signal s obtaining edges from A that are consumed by B. Fig. 4 outlines the corresponding part of the wrapper model. Gate A is assumed to be hazard-free. Furthermore, assume each switching of B takes less time than any switching of A.

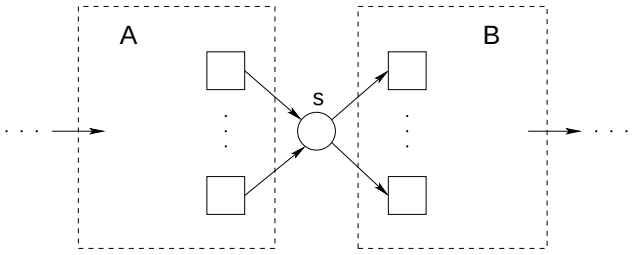

Figure 4. gates A and B, linked by signal s

Consequently, each edge rising at s is consumed by B, before the next edge appearing at s. In terms of the model, this would mean that no reachable marking has two tokens on s. But the model does not respect delay times so far. Consequently, a reachable marking M is conceivable with two

tokens on s. Generally speaking, 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 behavior, 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. As a consequence, the number of

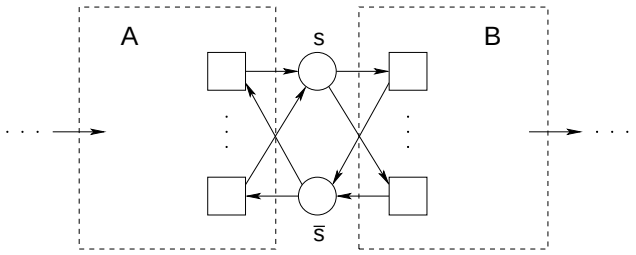

Figure 5. gates A and B, linked by signal s and its complement place

places of the wrapper model increases from 288 to 363.

The resulting net's reachable markings still represent all reachable states of the wrapper. Hazards are just no longer represented by two tokens on a place.

3.7. Verification

As we showed in section 3.6, the detection of a potential hazard of the wrapper is reduced to the reachabilty 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 [13]. Its features include powerful reduction techniques such as symmetry detection [12], partial order reduction [11] and the sweep-line method [14]. 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 [8] 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. Abstraction of the model

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 three such techniques in this section.

4.1. Integrating gates

The state space of a model of the GALS wrapper can be reduced by integration of several instances of its logic gates. An almost trivial integration step is the replacement of two instances of a gate into one instance. Fig. 6 shows a part of the wrapper's output port. Its model can apparently

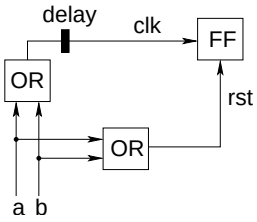

Figure 6. part of the output port

be replaced by the model of Fig. 7. Fig. 8 shows a more involved example, composed from two AND gates and one

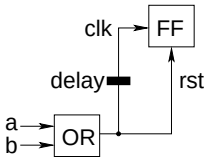

Figure 7. integrated OR gates

OR gate. Though we did not show the internal structure of OR gates, it should be fairly obvious that an OR gate has

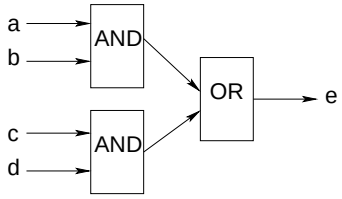

Figure 8. sequential logic gates

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.

The idea is to integrate the three gates of Fig. 8 into one pattern, as outlined in Fig. 9. We refrain from detailing the pattern of Fig. 9. It suffices to know that each occurrence

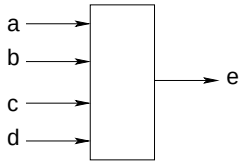

Figure 9. integrated gates

of an edge of signal a, b, c or d fires exactly one transition.

The circuit of Fig. 8 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 have to change. Secondly, the level of signal e has to change. The pattern in Fig. 9 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. 8 minimizes the number of states 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. LoLA reduced the wrapper model from 243 to 201 places.

The full state space of the reduced model is still too large for being handled by LoLA or by SMV. So, it was inevitable to search for additional reduction techniques.

4.2. Causalities

The correctness of a wrapper depends frequently on assumptions about the delays of its gates. We did not model this aspect so far. As a consequence, the model may yield states – and hence potential hazards – that in reality are known not to occur. We suggest not to model assumptions on gate delays, but only their effect to the gate's behavior, in particular the effect to the order of signal edges and consequently the order of gate switching operations.

As an example, consider the flip-flop in the context as in Fig. 7. The delay gate guarantees that the flip-flop processes the reset edge (rst) before the clock edge (clk).

The flip-flop pattern can be extended by additional control places, guaranteeing this behavior. We refrain from going into details here.

4.3. The Simplification Technique

In section 4.1 and 4.2 we presented two techniques that firstly reduce the state space of the model and secondly preserve 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 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

subcircuit	**states**
input port	81,352
time-out generator	2,603
output port	201
clock control	26
local clock generator	17

Table 1. state space of the wrapper's subcircuits

gate G, only the level information of G are needed. From G's environment we only have to preserve the level change of the signals. Thus, we can built simplified, more precisely, abstract patterns for each subcircuit. 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 level changes of the signals. Finally, the wrapper is composed by plugging four abstract subcircuits 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. In the following we show an example of an abstract pattern and further how we proved its correctness concerning its concrete pattern.

The wrapper includes a subcircuit, generating the local clock (LCG), as shown in Fig. 10. As initial state of the LCG assume the *STOPI* signal high, and all other signals low. The clock is now triggered by *STOPI* falling low. Due to the functionality of the involved gates, NOR switches,

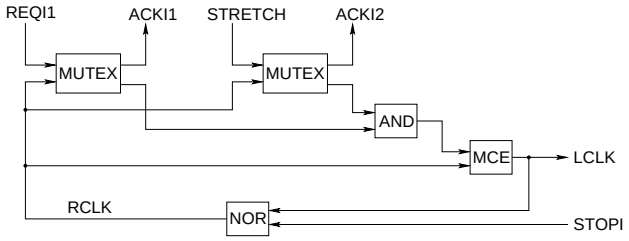

Figure 10. Simplified block structure of the LCG

causing *RCLK* to rise. Then both Mutex gates switch, followed by AND and C-element (MCE). This completes a cycle, now starting its second round by switching the NOR gate now with the effect of lowering the *RCLK* signal.

This circuit is apt to stretching the clock phase. This is achieved by rising edges at *REQI1* 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.

The simplified version of the Mutex gate reduces the set of places from 48 to 44 and the set of transitions from 52 to 40. Replacing in the LCG of Fig. 10 one of the Mutex patterns of Fig. 3 with its abstract counterpart of Fig. 11, reduces the state space from 2440 states to 1470 states (without using any reduction technique). The two Mutex

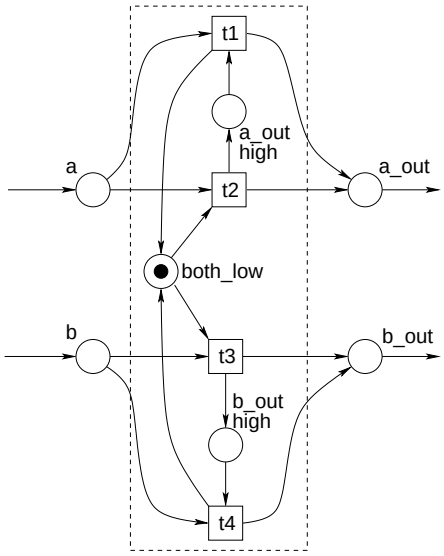

Figure 11. abstract Mutex pattern

gates of Fig. 10 can adequately be detailed by the pattern of Fig. 3.

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. Hence the pattern of Fig. 3 can be replaced by the simpler pattern of Fig. 11 without affecting the Mutex gate's behavior in the LCG. The inscriptions of the internal states of the pattern of Fig. 11 refers to the level of the output signals a_out and b_out.

Building abstract patterns it was 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. 11 and 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 at least one transition in the concrete pattern which has no counterpart in the abstract pattern, it can be expressed by an internal transition, denoted as a so called τ-step. Hence, the relation is called a *weak simulation*.

Table 2 visualizes the relation of the concrete and abstract Mutex pattern, which is a weak simulation. Note, the two transitions in the concrete Mutex ($t5$, $t6$) that could not be enabled are not considered. The table demonstrates that

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

the concrete Mutex model is weakly simulated by the abstract Mutex model (because of the τ-steps). Thus, Fig. 11 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 30 and the set of transitions from 52 to 15. In contrast to 2440 states in the concrete LCG the abstract LCG reduces the state space to 92 states which is a reduction of more than 96%. By use of partial order reduction and the sweep-line method the number of states is reduced from 17 (see Table 1) to 6.

In addition to the LCG we also simplified output port, time-out generator and clock control to check the input port for potential hazards. With it we simplified the wrapper's model for a better analysis with LoLA. The respective wrapper is reduced to 143 places and 177 transitions and LoLA reduced the number of places to 89. The full state space could not be calculated by LoLA, but by SMV. Nevertheless, instead of SMV we used LoLA to search for markings

of possible hazards, because most of the markings were calculated after short time and there was no need to calculate the full state space. Next, we will give more detailed information about our synthesis 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. The techniques presented in section 4.1 and 4.2 preserve all potential hazards in the model. In contrast, the use of abstract patterns as presented in section 4.3 only preserves all potential hazard 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 more than one wrapper model, each model with a smaller state space. In the following we present a method to detect hazards for a given model. Our synthesis is very pragmatical. Nevertheless, 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 could have potential hazards in the input port and output port under very specific circumstances. Finally, we observed potentially hazardous behavior of 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.

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 C-elements are no candidates; they are hazard-free due to their distinguished structure.

In the second step all candidate structures S are considered: S has a set of hazard markings 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 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 of the wrapper.

In a next step we tried to simulate those sequences of signal edges, 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. 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.

For the reason of state space explosion we often combined partial order reduction and the sweep-line method in LoLA. Due to the operation of the sweep-line method a witness could not be constructed by LoLA, but the final marking. The respective hazard marking detected for is only a subset of this final marking. So we had to enhance our hazard marking with information of the final marking. Then, we started another run by only using partial order reduction to construct a witness. Sometimes we had to iterate that procedure.

6. Conclusion

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 for 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. For this purpose, 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 three techniques to reduce the model's state space. These techniques allowed us to detect several potential hazards which could all be fixed in an improved version of the wrapper.

From this case study, we have learned that the use of patterns in different levels of abstraction is very useful and helps reducing the model and also the state space. LoLA proved useful in finding witnesses quickly, even without being capable of storing all of the reduced state space. This is due to its powerful reduction techniques, particularly, sweep-line method and partial order reduction. Using these two techniques together instead of using only partial order reduction, state space could be reduced by factor 2 – 10. Especially, 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 replace an abstract pattern of a gate by its corresponding concrete pattern instead of replacing patterns of subcircuits as is done in the paper. In addition we will focus our work on bringing more causalities into the model in

order to simplify hazard detection.

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] H. Genrich and R. Shapiro. Formal Verification of an Arbiter Cascade. In K. Jensen, editor, *Proc. 13th Int. Conf. Application and Theory of Petri Nets*, volume 616 of *LNCS*, pages 205–223. Springer Verlag, June 1992.

[4] D. Gomm. *Modellierung und Analyse verzögerungsunabhängiger Schaltungen mit Petrinetzen*. Dieter Bertz Verlag, 1996.

[5] 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.

[6] 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.

[7] 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.

[8] K. McMillan. *Symbolic model checking: an approach to the state explosion problem*. Kluwer Academic, 1993.

[9] J. Muttersbach. *Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems*. PhD thesis, ETH Zurich Switzerland, 2001.

[10] W. Reisig. *Petri Nets.* Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, EATCS Monographs on Theoretical Computer Science edition, 1985.

[11] 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.

[12] K. Schmidt. How to calculate symmetries of petri nets. *Acta Informatica*, (36):545–590, 2000.

[13] 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.

[14] 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.