Ultra Low Cost Asynchronous Handshake Checker

Steffen Zeidler Marcus Ehrig Miloš Krstić
Michael Augustin Christoph Wolf Rolf Kraemer

IHP – Innovations for High Performance Microelectronics
Im Technologiepark 25, 15236 Frankfurt (Oder), Germany

Email: {zeidler|ehrig|krstic|augustin|cwolf|kraemer}@ihp-microelectronics.com

***Abstract*—This paper presents a new on-line checking scheme for asynchronous handshake protocols. The proposed scheme requires very small chip area while maintaining high coverage for all considered faults which are briefly exposed. In addition to simple pass-fail information the checker provides off-line diagnosis capabilities in order to further analyze the cause of a fault and the time of its occurrence. In order to verify its functionality the checker was proven by performing analogue simulations. In addition the area overhead and the power consumption was determined and compared with existing implementations.**

I. Introduction

Fast enhancements in semiconductor technologies result in the design of complex systems integrated within one single chip that becomes a System-on-Chip (SoC). But the growing complexity of SoCs also leads to problems circuit designers have to solve, e.g. synchronous clock distribution. Considering this, asynchronous circuits are becoming an interesting solution for these design issues due to their beneficial properties:

- less power dissipation
- independence of process variability
- performance aspects
- modularity
- lower EMI

A heterogenous form, globally-asynchronous locally-synchronous (GALS) circuits, offer similar capabilities and combine both synchronous and asynchronous design styles (cf. [1]). Here, synchronous blocks (SBs) process data with individual clock domains and communicate via asynchronous channels. The advantage is that the SBs benefit from the full synchronous design flow while using a globally asynchronous methodology.

Unfortunately, industrial companies avoid asynchronous circuits as well as GALS systems in commercial designs due to the lack of established commercial EDA tools and the widespread assumption that these systems are difficult to test. This is based on several issues, e.g. most test automatons support mainly synchronous circuits. For changing this assumptions, many investigations were spend into testing techniques for asynchronous circuits. H. Hulgaard et al. have given an overview about these technologies in [2]. One popular test solution is to integrate scan paths into the asynchronous design (cf. [3], [4], [5]), since this technique is established for synchronous systems and accepted by the industry. Unfortunately, introducing scan paths is a large overhead due to the integration of a clock tree that should be avoided within asynchronous designs. Furthermore, a scan test is very limited in performing functional and at-speed tests.

One solution for testing asynchronous and GALS systems may be the usage of on-line testing techniques that integrate additional logic to the circuit for checking (at least parts of) the design. This Design-for-Testability (DfT) technique is feasible to perform tests during normal operation of the circuit, thus, it is capable of detecting permanent as well as transient and dynamic faults (cf. [6]). This aspect of on-line testing techniques is really useful for circuits with asynchronous handshake channels, because dynamic or transient faults that affect the handshake signals of a channel are difficult or even impossible to test using off-line test techniques which, furthermore, may not operate at-speed. Those faults most likely are disastrous in asynchronous designs since they may inhibit the communication or lead to setup and hold time violations within registers of communicating modules.

In this paper an enhanced on-line checking scheme for asynchronous handshake protocols is proposed. Such a protocol checker for asynchronous handshake circuits was presented by D. Shang et al. in [7] first and further improved in [8]. The goal of the checker, proposed in this paper, is to further reduce its complexity, area requirement and power dissipation in order to integrate this DfT technique easier into designs with asynchronous handshake channels. Therefore, a completely new checker architecture is proposed that requires approximately one half the area of existing checker implementations.

The paper is organized as follows: The next section provides an overview about asynchronous handshake circuits. Section III gives an introduction into on-line testing and presents related work, especially D. Shang's checker solution. Section IV introduces the considered fault models that the checker shall be capable to detect. The checker itself is presented in section V. Section VI presents experimental results. Finally, the paper is concluded in section VII.

II. Asynchronous Handshake Circuits

The basic characteristic of asynchronous circuits is that they are not controlled by a global clock. Hence, a complex clock distribution tree is not necessary to control the memory elements. In general, these circuits are composed of asynchronous communicating modules as shown in Fig. 1. These asynchronous modules consist of asynchronous control

978-1-4244-4595-0/09/$25.00

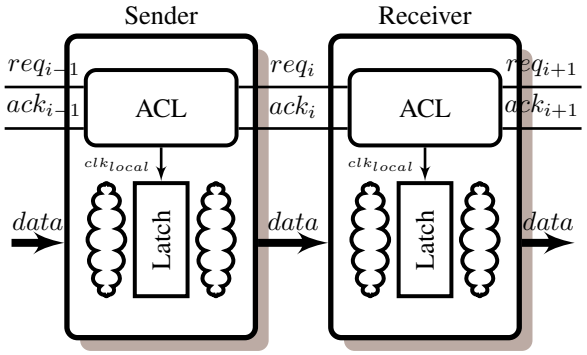


Fig. 1. Asynchronous handshake circuits

logic (ACL) which generates a local clock signal that triggers corresponding memory cells to store the current state. The communication between asynchronous modules is organized by a bidirectional handshaking protocol using a request (req) and an acknowledgment (ack) signal. A request is assigned (req^+), when data has to be transmitted. If the data was received then an acknowledgment fires (ack^+). Afterwards, the request is set low followed by the acknowledgment that is also set low.

A handshake protocol can be categorized in different ways. One major distinction regards the number of phases. Here, two major kinds of handshaking protocols can be distinguished, 2-phase and 4-phase as shown in Fig. 2. The 4-phase handshake protocol is level-based, i.e. the four different phases of the protocol depend on the levels of the handshake signals. In comparison to this, the 2-phase handshake protocol is transition-based, i.e. the phase of the handshake depends on signal transitions, rather than on the levels of the handshake signals.

Another distinction considers how data bits are transmitted. A handshake-protocol with explicit request and acknowledgement signals and one wire for each data bit is called single-rail protocol. Here, the handshake is organized using the protocol given in Fig. 2. The counterpart to this is the dual-rail handshake protocol. It uses two wires $(d_f d_t)$ for each data bit where a valid '1' is encoded with (01) and a valid '0' with (10). When both signals are low then the bit is not valid. This is referred to as the so called NULL-value. The last case, if both signals are high, corresponds to the forbidden state. Dual-rail handshake protocols have no explicit request signal. Instead, the request is encoded within the data lines and is

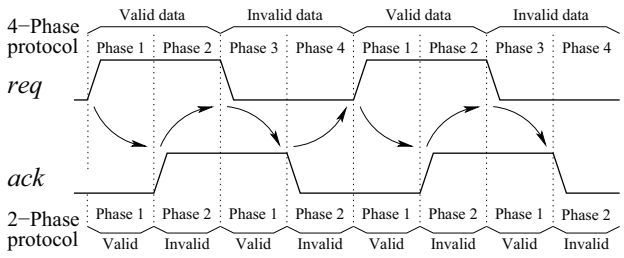


Fig. 2. Asynchronous handshake protocols

assigned when all data bits have a valid value.

The last distinction regards the initiator of the handshake. In a *push*-protocol the sender is the active part, i.e. that the request is transmitted from the sender to the receiver. The opposite is also possible and is called a *pull*-protocol. Here, the receiver initiates the transmission by sending a request that is acknowledged by the sender. A particular handshake protocol realization is a combination of the mentioned categories. J. Sparsø and S. Furber have given a comprehensive overview of the most important protocols in [9]. In this work we focus only on single-rail protocols, since the proposed on-line testing scheme is intended for those protocols. Other restrictions are not necessary, thus, the checker can be used for any protocol implementation that belongs to the single-rail protocol family regardless if it is a *push*- or a *pull*-protocol, 2-phase or 4-phase signaling.

III. Introduction into On-Line Testing and Related Work

The integration of on-line testing capabilities directly into the design-under-test (DUT) has several advantages:

- Reduction of required output pins for test purposes
- Internal parts can be tested.
- The DUT can be tested at-speed.
- Performing test even when the DUT is integrated into a working system.
- Detection of transient and dynamic faults

However, there are also some disadvantages. The additional hardware overhead, energy consumption and even a small delay are things that discourages the usage of on-line checking schemes. For this reason the check scheme should preferably be of low complexity. However, on-line checking becomes an interesting solution with the growing complexity of semiconductor circuits.

The term on-line checking/testing is often associated with concurrent error detection and self-checking techniques. In our opinion, the term self-checking in consideration of asynchronous circuits is more appropriate for the property of *speed-independent* (SI) and *self-timed* (ST) circuits to halt in the presence of a stuck-at-fault within the handshake logic. This well known property was for instance discussed by H. Hulgaard et al. in [2].

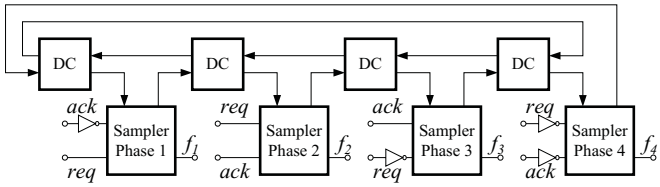
Mainly, on-line testing techniques utilize information about the output generated from the inputs of the DUT. This can be realized by the usage of codes, e.g. parity code or residue codes. M. Gössel et al. have given a comprehensive overview about on-line testing scheme in [10]. Another type of on-line fault detection can be utilized when the output of a DUT must fulfill a known property. Here, the checking unit acts as a watchdog that only monitors the output of the DUT independently of its inputs.

Based on the latter on-line testing scheme, D. Shang et al. have developed checking schemes to detect faults within handshake protocol of asynchronous circuits. Their checker implementation, proposed in [7] and [8], respectively, are used for on-line monitoring of asynchronous handshakes to detect

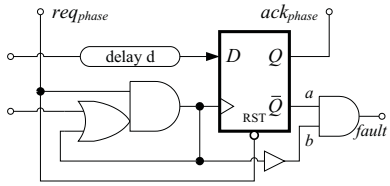
any faults especially transient and dynamic faults which cannot be detected using external testers. These faults most likely disturb the correct operation of the circuit.

In this paper we also present an on-line asynchronous handshake protocol checker (AHPC). As well as it was done by D. Shang et al. in [7], [8], the handshake signals of an asynchronous handshake protocol channel are observed. Their latest solution of an AHPC, which was presented in [8], is composed of four stage checking units, referred to as samplers, and four control units. These components are arranged in a cyclic structure which is given in Fig. 3(a). The purpose of a sampler is to check one of the four handshake protocol stages. As shown in Fig. 3(b), each sampler consists of a D-flip-flop (DFF), a delay element and some logical gates. The delay element is used to determine a minimum delay $d_{\min}$ between every signal transition. The control units are realized by David Cells [11] which are shown in Fig. 3(c). Their purpose is the activation of the samplers and, therefore, controlling the operation flow in an asynchronous manner. The major problem of this design is the multiple usage of completely identical sampling units at which especially the delay elements require much chip area. Although separate delay elements enable the definition of separate minimum delays for each protocol phase, this aspect is not intended by their AHPC solution. Thus, the multiple usage of exactly the same delay introduces an overhead which can be omitted. In addition, one has to consider that delay elements have to be adjusted for each technology to achieve the desired delay. The higher the integration level the higher is the relative overhead for a specific delay. Thus, the number of delay elements should be kept low.

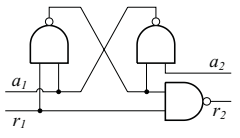
For this reason, our proposed AHPC solution avoids the multiple usage of sampling units in order to reduce its complexity. Here, the number of sampling units and the according number of delay elements for determining $d_{\min}$ is reduced to a minimum of one. Consequently, the resulting design is much simpler in comparison with the checker proposed in [8].



(a) Global structure



(b) Sampler



(c) David Cell

Fig. 3. AHPC proposed in [8]

However, this simplification sacrifies the flexibility to define a separate delay for each of the protocol phases. Nevertheless, in our opinion the definition of only one minimum delay for all protocol phases is sufficient.

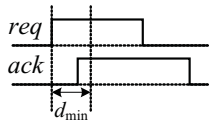
IV. Considered Fault Models

Asynchronous SI and ST handshake circuits tend to halt in the presence of stuck-at-faults (s-a-f). Hence, these faults can be easily detected using a deadlock detection scheme that can either be integrated within the DUT or realized by the external tester. Beside those faults other effects, e.g. glitches caused by hazards or single event effects, can negatively influence an asynchronous handshake. Such faults are hard to test using external off-line test methods, but can easily be observed using on-line test schemes, as it is shown in this paper. In addition, hazards are often the result of other internal faults within the communicating modules. Consequently, they can be considered as the first indication whether a circuit is faulty or not. Since all these faults affect the functional behavior of asynchronous handshake protocol channels it is enough to detect faults from the functional sight. These faults are the same as the ones defined in [8] and include the following:

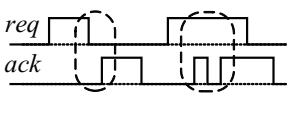
- Stuck-at-faults on the handshake signals
- Premature transitions of a handshake signals
- Order-violations within the protocol

In the presence of a s-a-f at the handshake signals the whole system will run into a deadlock. Unfortunately, our checker as well as the previous solutions are not able to indicate such a fault at the corresponding output. For the previous AHPC implementations D. Shang et al. suggested to use a deadlock detection scheme to detect such s-a-fs. In our solution, s-a-f at the handshake signals can be detected using off-line analysis of the state of the checker. This mechanism, which is presented later, also offers the possibility to further analyze the cause of the error and its time of occurrence.

The next category of faults that shall be detected by the AHPC are *premature transitions* (p-t) of the observed handshake signals, as they are shown in Fig. 4(a). A premature transition is a transition that occurs to close to its preceding one. In other words, the time between two consecutive transitions is smaller than a minimum permitted time $d_{\min}$. In asynchronous circuits these faults can lead to setup and hold time violations of registers which are controlled by the asynchronous handshake signals. The last category of considered faults are violations regarding the order of handshake signal transitions, referred to as *order-violation-faults* (o-v-f). An o-v-f happens when the monitored signal transitions occur



(a) Premature transition



(b) Order-violation-fault

Fig. 4. Fault models

in an order different from the one mentioned in section II. These faults can be the result of temporary glitches on the handshake signals resulting from hazards or faults within the communicating modules. Two o-v-f examples are given in Fig. 4(b).

V. Handshake Protocol Checker

Both checker implementations in [7] and [8], respectively, were separated into submodules of two different types. The checker in [7] consists of four samplers and five David Cells and has two operation modes: normal operation and the self-test mode. In normal operation the checker works according to its specification, hence, it detects the considered faults on handshake signals. The self-test mode is used to detect internal faults within the checker itself. When a fault is detected the checker deadlocks. This deadlock must be monitored by an additional control circuitry, the test-mode controller, that generates a fault indication signal. The improved version, proposed in [8], requires only four David Cells and the complexity of the samplers was also reduced. Furthermore, the self-testing mode was removed resulting in a simpler design.

However, both designs are relatively complex. For this reason, we implemented a new AHPC which has a reduced complexity. The structure of the design is shown in Fig. 5. It contains three simple submodules: a transition detection unit (TDU), a 2-bit feedback-shift-register (FSR) and a comparator. In addition, the checker comprises one delay element, one D-flip-flop (DFF) and some combinational gates.

The TDU works as its name suggests: It detects transitions on the handshake signals and generates a pulse used to clock the DFF and the FSR. Fig. 6 illustrates two implementations of the TDU. Both mainly consist of XOR-gates and one or two buffer cell(s), respectively. The first solution shown in Fig. 6(a) is cheaper regarding its area requirement due to the usage of only one buffer. Its first XOR-gate fires on each signal transition of both handshake signals and the output level depends on whether both signal are equal (logic 0) or different (logic 1). The buffer ensures that for its delay period d_b both inputs of the second XOR-gate are unequal causing a pulse at the output of the second XOR-gate for the period d_b. The problem here is that simultaneous transitions on both handshake signals can mask each other, i.e. it is possible that

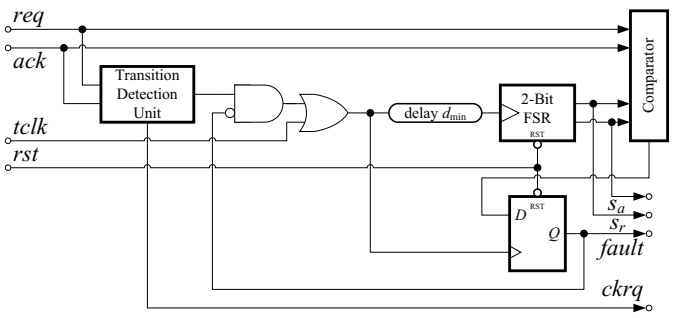
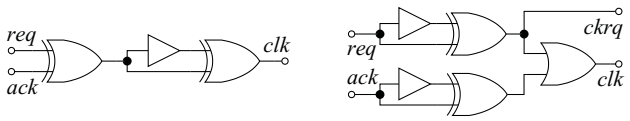


Fig. 5. New handshake protocol checker



(a) simple TDU (STDU) (b) enhanced TDU (ETDU)

Fig. 6. Transition detection units

no pulse is generated when both signals fire simultaneously. Fig. 7 shows such a faulty handshake sequence. The first and the second waveform represent the handshake signals. The third waveform of Fig. 7 illustrates the output of the first TDU solution of Fig. 6(a). The second implementation illustrated in Fig. 6(b) avoids this masking, as it can be seen in the fourth waveform of Fig. 7. Here, every transition of each handshake signal generates a separate pulse on the output of the corresponding XOR-gate. The usage of the OR-gate avoids the masking of two simultaneous pulses. Unfortunately, the second buffer increases the area overhead. However, this implementation offers the possibility to distinguish between transitions on the request and the acknowledge signal. Therefore, the enhanced TDU implementation has the additional output signal $ckrq$. This signal can be used to count the transitions of the request signal in order to detect s-a-fs which cannot be detected by the checker itself. Furthermore, this signal can be used to determine the number of requests on a handshake channel before a fault is detected in order to improve the diagnostic capabilities.

The FSR is used to generate the expected handshake signal values of the next rather than of the actual protocol phase. Therefore, it generates the sequence $10 \rightarrow 11 \rightarrow 01 \rightarrow 00$ corresponding to the protocol phases. These values are compared with the actual signal values within the comparator, which indicates the difference between the values by a logical 1 at its output. The output value of the comparator is sampled by the DFF. Both, the DFF and the FSR are clocked by the pulse generated by the TDU. The only difference is that the propagation of the clock signal to the FSR is delayed for the duration $d_{\min}$ using the delay element.

In the fault free case the checker works as follows: Without loss of generality, both signals are initially low, i.e. $(req, ack) = (0, 0)$. Because the FSR stores the state of the next handshake protocol phase, the stored value is

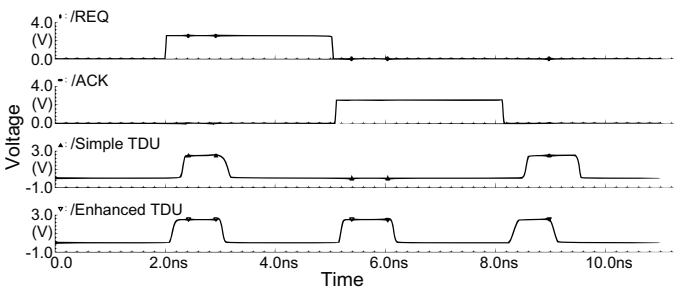


Fig. 7. Simulation of both TDU versions

$(s_r, s_a) = (1, 0)$. When the next transition of the handshake signals happens, i.e. req^+, then the values of the handshake signals change to $(req, ack) = (1, 0)$. This change causes the TDU to generate a clock pulse that triggers the DFF to sample the result of the comparison of (req, ack) and (s_r, s_a). When no fault has occurred then both values are equal and the DFF memorizes a logic 0. Subsequently, the pulse propagates through the delay element, thus, after the period of $d_{\min}$ the FSR changes its state (s_r, s_a) to $(1, 1)$ corresponding to the handshake signal values of the next protocol phase.

In order to guarantee that the checker works correctly, it must be ensured that the pulse generation and propagation takes a certain amount of time d_{pulse} that has to be greater than the delay of the comparator d_{comp}. Otherwise the DFF would sample the wrong comparison result or the setup and hold time of the DFF would be violated resulting in an unexpected behavior.rr

If no fault occurs then the progress continues and no error will be reported at the corresponding output of the checker. When a premature transition occurs, then the DFF is triggered to sample the comparison result of the handshake signal values and the values stored in the FSR before the state of the FSR has changed. Assume that the protocol is in its second phase, i.e. $req = 1$ and $ack = 0$. Now, ack^+ fires too close to req^+, i.e. $t(ack^+) - t(req^-) < d_{\min}$. Due to the delayed clock pulse which still propagates through the delay element the FSR has not changed its state according to the next protocol phase. Thus, the values stored within the FSR (s_r, s_a) and the also s of the handshake signals are not equal. In consequence, the sampled comparator output is a logic 1 that indicates an error. In case of an order violation it is obvious that such a fault is also detected using this scheme, because the values of the handshake signals differ from the expected values stored in the FSR. Thus, the DFF will also sample this difference and will indicate a fault at its output.

To ensure that this fault indication can be observed by the external tester, the checker shall halt in its state. Therefore, the inverted output of the DFF is fed back and coupled with the clock signal using the AND-gate between the TDU and the FSR. This gate works as a clock gate when a fault was detected and inherits the propagation of further pulses generated by the TDU. Thus, the circuit is caused to deadlock until the checker is set to its initial state using the *rst* signal. In order to detect faults within the checker itself the *tclk* signal can be used to clock the FSR and the DFF. A fault within the checker can be detected by applying four clock pulses. For instance, a fault that affects the delay element or the FSR will result in an output sequence of the FSR different from the expected one. Similarly, a fault that affects the comparator, the DFF or their interconnection will cause that after the first clock pulse the DFF does not change its value from logic 1 to logic 0 and after the third clock pulse form logic 0 to logic 1. To perform this kind of test it is necessary to inhibit all transitions on the handshake signals. In a GALS design this can be performed by stopping all local clock generators. In a pure asynchronous design this can simply be achieved by avoiding the application of a handshake.

To collect the status information of all checkers two mechanism can be distinguished, one for on-line monitoring of all checkers and one for off-line analysis with special emphasis of diagnosis. For off-line analysis the checker scheme is extended by a checker control unit (CCU) which provides a standard JTAG interface for external tester connection, as it is shown in Fig. 8. The CCU receives the signals *TRS*, *TEN*, *TMS*, *TDI* from the automated test environment/equipment (ATE) to control the state of the test and delivers the result of the test at *TDO*. As it can be seen in Fig. 8 the checkers are connected to the CCU via a bus. In order to determine which checker shall be connected to the CCU the *TDI* signal can be used to serially scan-in the binary encoded number of this checker. Using this number the CCU generates a bus enable signal for this checker.

Fig. 9 shows the structure of the control unit. In principle, the CCU mainly consists of combinational logic, a decoder and three scannable registers. Thereby, the configuration register is used to store the binary number representation of the checker that shall be connected to the CCU. The parallel-output port of the register is connected with the decoder which generates a signal to connect the corresponding checker to the bus. The request-transition-count register is a scannable counter, that can either be clocked by the connected checker to count the request transitions or by the external clock signal when the CCU is in scan mode. The last register is used to capture the results of the connected checker. In consequence the CCU provides the following operation modes:

Reset all (*TEN=TMS=RST=0*): All components including the CCU and all checkers are set to their initial states.

Reset configuration register (*TMS=1, RST=0*): Resets the configuration register only.

Reset request counter (*TEN=1, RST=0*): Resets the request counter only.

Normal observation mode (*TEN=TMS=0, RST=1*): Each checker is in observation mode. In addition, the result register of the CCU samples the output of the connected checker at each rising edge of the external clock *TCK*.

Checker testing mode (*TEN=0, TMS=RST=1*): In this mode *TCK* is connected to the *tclk* input of each checker. This mode is used to perform tests to verify the correct behavior of the checkers.

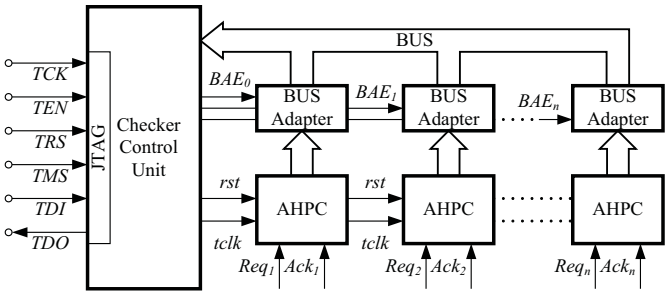


Fig. 8. Protocol check architecture

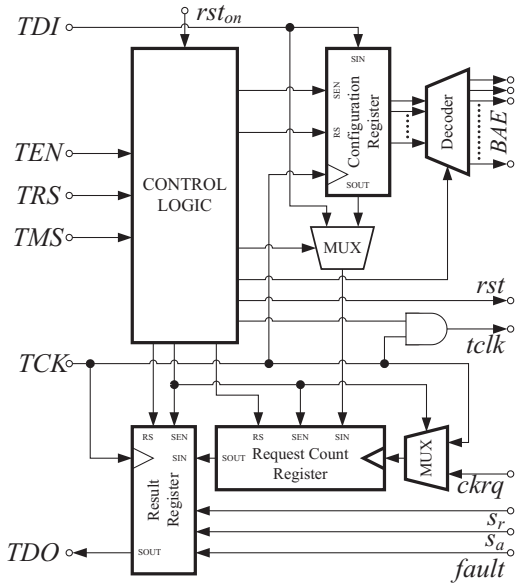


Fig. 9. Checker control unit

Result scan-out mode (*TEN=1, TMS=0, RST=1*): When the state of a checker was captured it can be scanned out during this mode. Three clock cycles are enough to scan-out the checker result. When also the request counter shall be scanned then more cycles are necessary.

Configuration scan-in mode (*TEN=TMS=RST=1*): During this mode the configuration register can be programmed in order to connect a specific checker to the CCU. Therefore, *TDI* can be used to sequentially apply the binary number representation of the checker.

Although, some s-a-fs cannot directly be indicated by the checker itself, it is possible to detect them using further off-line analysis of the request counter and the state of the FSR. A stuck-at fault at the request signal will cause the request count register to be at its initial state. Similarly, an ack-stuck-at-0 cannot directly be indicated by the checker, because the channel is in a deadlocked state where it waits for the firing of ack^+. In this case the state of the FSR (s_r, s_a) is $(1, 1)$ and the request count register has counted one request transition (req^+).

In normal operations mode the on-line mechanism can be used to determine whether a fault has occurred. Therefore, the fault indication signals of all checkers are combined to one signal via OR-gates, as shown in Fig. 10. Using this indication signal one can take measures to handle a fault during normal operation, e.g. all data transfers are repeated. After handling the fault, all checkers have to be set to their initial state in order to detect subsequent faults. For this reason the environment should provide a reset signal which is combined with the reset signal of the CCU to the rst input signal of the checker.

VI. Results

Both AHPC implementations, the one proposed here and the one provided in [8], were realized in IHP $0.13\,\mu\mathrm{m}$ technology

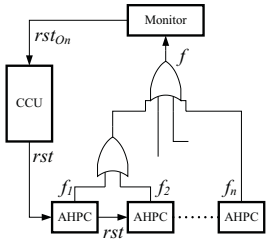


Fig. 10. On-line monitoring scheme

using the Synopsys Design Compiler for synthesis in order to estimate the area and power requirements. Our checker solution was realized using the simple and the improved implementation of the TDU. Table I shows the resulting area requirements for the considered checker implementations. The first column illustrates the area of both designs without any delay element for determining $d_{\min}$. It can be seen that the checker provided by D. Shang et al. has almost double the area requirement of the one provided here. The subsequent columns illustrate the overhead for the checker with delay elements realizing $d_{\min} \in \{1\,\mathrm{ns}, 2\,\mathrm{ns}, 5\,\mathrm{ns}\}$ where each delay element was implemented by consecutively connecting standard buffers.

The two last rows of the table list the area relation between Shang's checker and the two checker implementations proposed here. It is obvious to see that the differences between the checkers are rising with the growing length of the buffer chains. This is based on the fact that Shang's checker implementation requires four instances of the delay element. Hence, an increase of the minimum delay affects the area overhead by the factor of four.

Similarly, Table II shows the peak power and total power consumption of the considered checker implementations. These results were determined using Synopsys PrimeTime by applying a pattern sequence of 504 handshakes. With each run of the first 500 handshakes the time between req^+ and ack^+ was shortened by 20 fs such that from a specific handshake depending on $d_{\min}$ the checkers report p-ts. The last four handshakes were used to simulate the behavior of the checkers in case of o-vs. The results show that the power consumption of our solution is approx. 34-22% lower than the one of the previous implementation.

In order to demonstrate the correct operation of the checker

TABLE I
Area requirements

$d_{\min}$	$= 0\,\mathrm{ns}$	$\approx 1\,\mathrm{ns}$	$\approx 2\,\mathrm{ns}$	$\approx 5\,\mathrm{ns}$
Area in $[\mu\mathrm{m}^2]$				
AHPC ([8])	469.34	738.30	1007.26	1814.14
New AHPC[a]	195.00	268.96	336.19	537.92
New AHPC[b]	228.62	295.85	363.10	564.82
Area consumption in relation to [8]				
New AHPC[a]	41.55%	36.43%	33.38%	29.65%
New AHPC[b]	48.71%	40.07%	36.05%	31.13%

[a] Simple TDU [b] Enhanced TDU

TABLE II
POWER REQUIREMENTS

$d_{\min}$	$= 0\,\mathrm{ns}$	$\approx 1\,\mathrm{ns}$	$\approx 2\,\mathrm{ns}$	$\approx 5\,\mathrm{ns}$
Peek power in $[\mu\mathrm{W}]$				
AHPC ([8])	1059.00	1065.00	1169.00	1169.00
New AHPC[a]	816.00	768.60	768.00	768.60
New AHPC[b]	767.50	771.70	873.20	775.60
Total power in $[\mu\mathrm{W}]$				
AHPC ([8])	21.48	27.43	32.95	49.54
New AHPC[a]	14.12	19.96	25.00	35.96
New AHPC[b]	14.71	20.48	25.71	36.93
Power consumption in relation to [8]				
New AHPC[a]	65.74%	72.77%	75.87%	72.59%
New AHPC[b]	68.48%	74.66%	78.03%	74.55%

the design was verified by performing analogue simulations using CADENCE SPECTRE. The results are given in Fig. 11 which shows a pattern sequence that represents the different fault scenarios the checker is intended to detect. The sequence starts with a fault free handshake followed by a p-t of ack^- which is recognized by the checker, i.e. the checker sets the fault indication signal high. It can be seen that this indication signal stays high until the reset signal is applied for setting the checker to its initial state. In addition to this scenario, the figure shows other faults like o-vs and s-a-fs which are also detected by the checker. The last two waveforms shown in Fig. 11 represent the value of the FSR. When a fault occurs this value is also frozen so that the ATE can read out this information for performing fault diagnosis.

VII. CONCLUSIONS

An on-line checker is provided that has a drastically reduced complexity in consideration to previous implementations. This checker and the one provided in [8] were implemented using the IHP $0.13\,\mu\mathrm{m}$ CMOS technology to compare their area overhead and their power consumption. The results have shown that the new checker architecture has lowered the complexity of on-line checking handshake protocols to a fractional part of previous implementations while providing the same fault coverage for the considered fault models. Depending on the time $d_{\min}$ between signal transitions and the TDU implementation, the area requirement of our checker solution with respect to the considered minimum delays lies approx. between 30 and 49% of the chip area required for the design proposed in [8]. In consequence we submitted this checker design as a german patent.

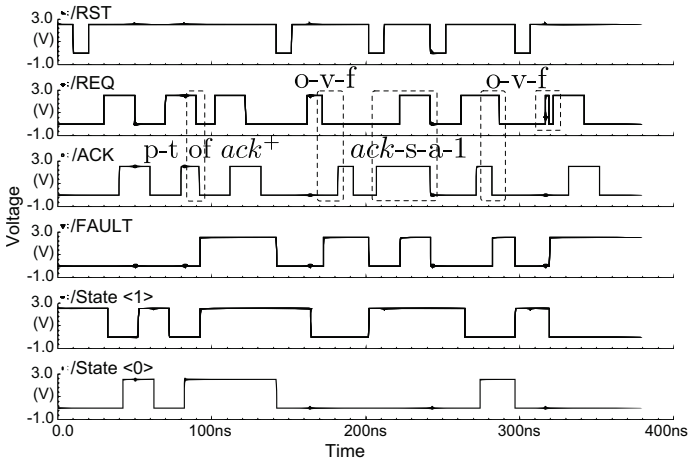


Fig. 11. Simulation of fault scenarios

REFERENCES

[1] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," Ph.D. dissertation, Stanford University, October 1984.

[2] H. Hulgaard, S. M. Burns, and G. Borriello, "Testing Asynchronous Circuits: A Survey," *Integration, the VLSI Journal*, vol. 19, no. 3, pp. 111–131, March 1994.

[3] F. T. Beest, A. Peeters, M. Verra, K. van Berkel, and H. Kerkhoff, "Automatic Scan Insertion and Test Generation for Asynchronous Circuits," in *Proceedings of the International Test Conference (ITC'02)*, 2002, pp. 804–813.

[4] K. van Berkel, A. Peeters, and F. te Beest, "Adding Synchronous and LSSD Modes to Asynchronous Circuits," in *Proceedings of the International Symposium on Asynchronous Circuits and Systems*, April 2002, pp. 146–155.

[5] A. Efthymiou, J. Bainbridge, and D. Edwards, "Adding Testability to an Asynchronous Interconnect for GALS SoC," in *Proceedings of the 13th Asian Test Symposium (ATS 2004)*, November 2004, pp. 20–23.

[6] M. Abramovici, M. A. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. IEEE Press, 1990.

[7] D. Shang, A. Bystrov, A. Yakovlev, and D. Koppad, "On-line Testing of Globally Asynchronous Circuits," in *Proceedings of the 11th IEEE International On-Line Testing Symposium (IOLTS'05)*, July 2005, pp. 135–140.

[8] D. Shang, A. Yakovlev, F. Burns, F. Xia, and A. Bystrov, "Low-cost Online Testing of Asynchronous Handshakes," in *Proceedings of the Eleventh IEEE European Test Symposium (ETS'06)*, May 2006, pp. 225–232.

[9] J. Sparsø, S. Furber, R. van Leuken, R. Nouta, and A. de Graaf, *Principles of Asynchronous Circuit Design: A Systems Perspective*, Kluwer Academic Publishers, Boston, 2001.

[10] M. Gössel, V. Ocheretny, E. Sogomonyan, and D. Marienfeld, *New Methods of Concurrent Checking*. Springer Verlag, 2008. [Online]. Available: http://www.springer.com/engineering/circuits+&+systems/book/978-1-4020-8419-5

[11] R. David, "Modular Design of Asynchronous Circuits Defined by Graphs," *IEEE Transactions on Computers*, vol. C-31, pp. 727–737, December 1982.