Microprocessors and Microsystems 39 (2015) 157–166

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

# Transaction-based online debug for NoC-based multiprocessor SoCs ☆

Mehdi Dehbashi [c,*], Görschwin Fey [a,b]

[a] Institute of Computer Science, University of Bremen, Bremen, Germany
[b] Institute of Space Systems, German Aerospace Center (DLR), Bremen, Germany
[c] Infineon Technologies AG, Munich, Germany

ARTICLE INFO

*Article history:*
Available online 11 March 2015

*Keywords:*
Transaction-based online debug
System-on-Chip (SoC)
Network-on-Chip (NoC)

ABSTRACT

As complexity and size of Systems-on-Chip (SoC) grow, debugging becomes a bottleneck for designing IC products. In this paper, we present an approach for online debug of NoC-based multiprocessor SoCs. Our approach utilizes monitors and filters implemented in hardware. Monitors and filters observe and filter transactions at run-time. They are connected to a Debug Unit (DU). Transaction-based programmable Finite State Machines (FSMs) in the DU check assertions online to validate the correct relation of transactions at run-time. The experimental results show efficiency and performance of our approach.



## 1. Introduction

Modern high-performance Systems-on-Chip (SoC) include many IP cores such as processors and memories. Network-on-Chips (NoC) have been proposed as a scalable interconnect solution to integrate large multiprocessor SoCs [2] [3]. Having a large SoC with complex communication among its cores, achieving complete verification coverage at pre-silicon stage is almost impossible. Therefore in addition to electrical bugs, some design bugs may also appear in the final prototype of an SoC.

The idea of transaction-based communication-centric debug is introduced in [4] to debug complex SoCs which interact through concurrent interconnects such as NoC. The transactions are observed using monitors [5] and the debug control unit can control the execution of the SoC (stopping, single stepping, etc.). In [6], transactions are stored at run-time in a trace buffer using on-chip circuits. After an SoC run, the content of the trace buffer is read and analyzed offline with software. The analysis software tries to find certain patterns [7] in the extracted transactions that are defined by their *Transaction Debug Pattern Specification Language* (TDPSL). Because of limited size of a trace buffer, getting an execution trace of the transactions related to the time of bug activation is a challenging problem. To overcome this problem, the content of the trace buffer is utilized to backtrace the transactions along their execution paths [8]. The backtracing is performed in transaction-level states using *Bounded Model Checking* (BMC). However, backtracing needs formal pre-image computations which can blow up for large and complex designs [9]. To address this problem, we need to have online detection to stop the SoC close to the time of bug activation at the transaction level.

In this paper, we present a transaction-based debug infrastructure which can be used not only for online debug and online system recovery but also for interactive debug in which an external debug platform programs the FSMs and the filters according to the considered assertions at each round of debugging. Our hardware infrastructure contains monitors, filters, and a debug network including *Debug Units* (DU). Filters and DUs are programmed according to the transaction-based assertions defined by TDPSL. Transactions are monitored only at master interconnects. Slaves send information to masters. This redundant information is used to observe the elements of transactions online. No modification of the internal components of the NoC is required. At run-time the programmable FSMs in the DUs investigate the assertions online and detect an error. Upon detection of an error, the DU recovers the SoC by informing the masters which have participated in the observed error. Then, the corresponding masters start the recovery process at run-time. Also we identify the requirements which a debug infrastructure has to fulfill in order to perform transaction-based online debug.

The main contributions of this paper are as follows:

– Introducing a debugging infrastructure to transaction-based online debug of NoC-based SoCs without modifying the internal components of the corresponding NoC (non-intrusive to the NoC).

☆ Mehdi Dehbashi did this work as part of his PhD in the University of Bremen. This work has been supported in part by the University of Bremen's Graduate School SyDe, funded by the German Excellence Initiative and in part by the German Research Foundation (DFG, grant no. FE 797/6-1). This paper is an extended version of [1].

* Corresponding author.
*E-mail address:* fey@informatik.uni-bremen.de (G. Fey).

http://dx.doi.org/10.1016/j.micpro.2015.03.003
0141-9331/

– Analyzing and finding transaction-based debug patterns at-speed using debug units including programmable filters and FSMs.

– Presenting an ordering mechanism in the routers of the debug network to order the transactions online.

– Online system recovery without stopping and interrupting the NoC.

The experimental results show the efficiency of our approach using different assertion patterns defined by TDPSL such as race, deadlock, and livelock. An NoC-based SoC using a mesh network is setup in the Nirgam NoC simulator [10] to evaluate our approach. Also we show the effectiveness of the proposed online recovery in the experimental results.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Section 3 introduces preliminary information on transactions and TDPSL. Our debug method including hardware and software parts is explained in Section 4. The debug patterns and their corresponding FSMs are explained in Section 5. This section also presents experimental results on an NoC-based SoC. The last section concludes the work.

## 2. Related work

Previous work also considered infrastructures for SoC debug. The existing debug infrastructures for complex SoCs are reviewed in [11]. These infrastructures support debugging such that the internal nodes become observable and controllable from the outside. The work in [12] presents a Design-for-Debug (DfD) technique for NoC-based SoCs. The technique enables data transfer between a debugger and a *Core-Under-Debug* (CUD) through the available NoC to facilitate debugging. A debug platform to support concurrent debug access to the CUDs and the NoC in a unified architecture is proposed in [13,14]. This platform is realized by introducing core-level debug probes in between the CUDs and their network interfaces and a system-level debug agent. The work in [15] proposes a ring-based NoC architecture to debug SoCs. The NoC is used to send back the information observed by monitors to the debugger. A *Non-Uniform Debugging Architecture* (NUDA) is proposed in [16] to debug many-core systems. A NUDA node in each cluster has three main parts: nanoprocessor, memory and communication. The NUDAs are distributed across a set of hierarchical clusters and are connected to each other through a ring interconnection. Then the address space is monitored using non-uniform protocols for race detection. Monitoring the address space without abstraction consumes a large storage and increases the latency of the error detection.

NoC test and diagnosis is the main focus in most of the previous work. Packet address driven test configurations are utilized in [17] to test and to diagnose regular mesh-like NoCs using a functional fault model. Then, link faults are diagnosed using test results and a diagnosis tree. The system test is modeled at the transaction level in [18] in order to facilitate test design space exploration, as well as the validation of test strategies and schedules. Interconnect faults in Torus NoCs are detected and diagnosed using BIST structures in [19]. Afterwards, the NoC is repaired by activating alternative paths for faulty links. In [20] an NoC with a faulty router or a broken link is repaired using spare routers. The inherent structural redundancy of the NoC architecture is exploited in a cooperative way to detect the faults using BIST [21]. Also diagnosis units in switches are utilized to localize a fault. In the diagnosis unit there are different comparators to compare data from all the possible pairs of switch input ports. A comprehensive defect diagnosis for NoCs is proposed in [22]. The approach uses an end-to-end error symptom collection mechanism [23] to localize datapath faults and a distributed counting and timeout-based technique to localize faulty control components [22]. The work in [24] diagnoses the NoC switch faults using hardware redundancy in each switch and a high level fault model. These approaches focus only on electrical bugs in the components of an NoC. However we consider design bugs which influence communications in an NoC-based SoC.

The work in [4] proposes a communication-centric debug approach. The approach focuses on the communication and the synchronization between the IP cores. Their approach uses not only monitors on the IP interconnects but also monitors on the internal components of an NoC such as routers. Debug data is read-out using scan chains and *Test Access Ports* (TAP). In our work, we do not transfer the debug data out of the SoC. The debug data is analyzed online using debug units. Also we monitor only the master interconnects without modifying the internal components of an NoC. The work in [6] uses trace buffers to store the transactions at run-time. The content of the trace buffer is analyzed offline in order to form transactions and to find debug patterns. Their approach monitors the bus to store the events in the trace buffer. However, we present an approach to debug NoC-based SoCs. We form the transactions online using distributed monitors and debug units. Also the debug patterns are found at run-time.

## 3. Preliminaries

### 3.1. Transaction

In this section we shortly explain the transaction elements from [25,6]. Each transaction includes a request and a response. Masters request and slaves respond. Each transaction has four basic elements: *Start of Request (SoRq)*, *End of Request (EoRq)*, *Start of Response (SoRp)*, and *End of Response (EoRp)*. In TLM, SoRq corresponds to putting the request in the channel by the master. EoRq is getting the request by the slave. SoRp corresponds to putting the response in the channel by the slave. EoRp is getting the response from the channel by the master. Also there are two additional elements which are called: *Request Error* (ErrRq) and *Response Error* (ErrRp). These elements handle error conditions and correspond to any kind of error that causes a request or a response to fail.

### 3.2. Transaction Debug Pattern Specification Language (TDPSL)

TPDSL has three layers: Boolean layer, temporal layer, and verification layer [6]. The Boolean layer includes *trans_exp* which represents *the basic elements of transactions*. The *trans_exp* format is as follows:

$$trans\_type\ (master, slave, type, address, tag)$$

Field *trans_type* can be any transaction element mentioned in Section 3.1 as well as the *Start of Transaction* (SoTr) and the *End of Transaction* (EoTr) which are similar to SoRq and EoRp respectively. Fields *master* and *slave* specify the ID of master and slave. Field *type* can be *Rd* or *Wr*. Field *address* indicates the slave address symbolically as SAME, SEQ, and OTHER. Field *tag* indicates the transaction number and is only used for buses that allow non-blocking requests and out-of-order responses [6]. In our paper, we show a transaction without considering the field *tag*.

The motivation to use symbols for the address field is to abstract and to compress the address bits. In this case, only the compact address information is stored or sent via network for debugging. The symbols can be defined with respect to the application and the granularity of debugging. SAME specifies that in the current transaction, slave address is same as the address in the previous transaction for this slave. SEQ specifies that in the current