Transaction-based Online Debug for NoC-based Multiprocessor SoCs

- by Mehdi Dehbashi, Görschwin Fey

Presented By
Xiangfei KONG, Chenxi LOU

11/17/2015
Outline

Background

■ Overview of Transaction-based Debug
■ TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

■ Debug Method & Requirements
■ Debug Infrastructure
■ Approach Limitation
■ Implementation
Outline

Background

■ Overview of Transaction–based Debug
■ TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

■ Debug Method & Requirements
■ Debug Infrastructure
■ Approach Limitation
■ Implementation
Transaction – Based Debug

- Why do we need transaction based debug in NoC based multiprocessor SoCs?
  - Growing complexity of interconnects & IP communication
  - Monitoring transaction packets at SoC level is relatively easy

- Lots of research in the area!
  - Transaction-based communication-centric debug
  - Debug pattern detection with TDPSL
  - Transaction back tracing using Bounded Model Checking

- Problems we currently have
  - Online debug method that can debug & recover at run time
  - An approach that is less intrusive to the NoC network
Transaction – Based Debug

- Master – Slave
  - Example: ARM AMBA AHB Protocol
  - Masters request, Slaves respond

- Transaction Elements
  - 4 Basic Elements:
    - Start of Request (SoRq)
    - End of Request (EoRq)
    - Start of Response (SoRp)
    - End of Response (EoRp)
  - 2 Additional Elements:
    - Request Error (ErrRq)
    - Response Error (ErrRq)
Outline

Background

- Overview of Transaction-based Debug
- TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

- Debug Method & Requirements
- Debug Infrastructure
- Approach Limitation
- Implementation
TDPSL – Transaction Debug Pattern Specification Language

- **Boolean Layer**

  \[
  \text{trans\_type (master slave type address )}
  \]

- **Temporal Layer**

  - *define transaction sequence properties*

    concatenation ( ; )  fusion ( : )  or ( - )  and ( & )  repetition ( [6] )

- **Verification Layer**

  - **Assertion**

    assert never

    eg. EoTr(m2,s1,Wr,-) ; SoTr(m1,s1,Rd,-)

  - **Filter Expression**

    defines over masters, slaves & trans types

    eg. Filter(*,*,*)
Outline

Background

■ Overview of Transaction-based Debug
■ TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

■ Debug Method & Requirements
■ Debug Infrastructure
■ Approach Limitation
■ Implementation
Highlights

- A debugging infrastructure that is non-intrusive to NoC
- Finding & analyzing transaction-based patterns at speed
- Present an online transaction ordering mechanism
- Online system recovery without stopping/interrupting NoC
Outline

Background

■ Overview of Transaction-based Debug
■ TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

■ Debug Method & Requirements
■ Debug Infrastructure
■ Approach Limitation
■ Implementation
Debug Method

Debug Requirements

- Be able to collect transaction elements at run-time
- Be able to order transactions online
- Be able to assert debug patterns online
Outline

Background

- Overview of Transaction-based Debug
- TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

- Debug Method & Requirements
- Debug Infrastructure
- Approach Limitation
- Implementation
Debug Infrastructure

Hardware infrastructure for a SoC with two masters & two slaves
Debug Infrastructure
Monitor

- Observe master interconnects to extract packet elements

Filter Structure in DU

Filter

- Filter unrelated transaction, both in monitor & Debug Unit (DU)

Choose an always different value

Don’t care
Debug Infrastructure

- DU-based Approach
- Slave-based Approach

- Master 1
- Master 2
- Filter
- Monitor
- NoC
- Slave 1
- Slave 2
- DU
- DRI
- NI
- R
DRI (Debug Redundant Information)
- Extract and transfer element address of a transaction

1. Slave - based Approach

- DRI in slave
- Low transfer cost
- Address info only in EoTr, wait to receive EoTr from slave

2. DU - based Approach

- DRI in DU
- Non-Intrusive to SoC
- More bandwidth needed to transfer slave address
- Larger memory storage in DU
Debug Infrastructure

![Diagram of debug infrastructure with nodes labeled Master 1, Master 2, Slave 1, Slave 2, DU, Monitor, Filter, and NoC. Connections between these nodes are indicated by arrows.]
DU (Debug Unit)

- Transfer data observed to top level
- Drop assertion unrelated transactions
- Synchronize Timer & Handle Error Cases
- Investigate Assertion Online

**LDU Structure**
(Local Debug Unit)

**CDU Structure**
(Central Debug Unit)

Transaction ordering based on timestamps
**DU Topology**

- **CDU**
  - Monitor
  - Filter
  - Timer

- **LDU**
  - Monitor
  - Filter
  - Timer

**NoC**

**Tree Topology**

- Low Error Detection Latency
- Large Number of DUs

**Sync Packet**
- Recovery Packet
- Debug Packet
- Restart Packet

**Ring Topology**

- Low hardware cost
- Need traffic balance approach
- Wait until all transactions arrive
- Transaction ordering challenge
Debug FSM

- Programmable FSMs utilized to investigate assertions online

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current State</td>
<td>Input</td>
</tr>
<tr>
<td>Start</td>
<td>Tr1</td>
</tr>
<tr>
<td>Start</td>
<td>Other</td>
</tr>
<tr>
<td>A</td>
<td>Tr2₁</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Err</td>
<td>-</td>
</tr>
</tbody>
</table>

FSM Memory Overhead

Worst Case Transaction # = t
Worst Case State # = s

Total Memory = \((\log_2 s + t)^2 \times \log_2 s\)

Total Memory = \(2^{\log_2 s + \log_2 t} \times \log_2 s\)

Transaction Pattern Encoding
Outline

Background

- Overview of Transaction-based Debug
- TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

- Debug Method & Requirements
- Debug Infrastructure

- Approach Limitation
- Implementation
Approach Limitations

- Only CDU has a comprehensive assertion checking for all masters
- Does not work for hardware faults
- Does not detect deadlocks between different threads of a single core
Outline

Background

- Overview of Transaction-based Debug
- TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

- Debug Method & Requirements
- Debug Infrastructure
- Approach Limitation
- Implementation
Implementation

➢ Experiment Setup

Tool - Nirgram NoC Simulator
Network - 3 x 3 mesh network
SoC Setup - Four masters, Four Slaves
Debug Pattern - Race, Deadlock, Livelock

➢ Assertion in TDPSL

<table>
<thead>
<tr>
<th>Race</th>
<th>Deadlock</th>
<th>Livelock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assert never { SoTr(m1,s1,Wr,-); SoTr(m2,s1,Wr,SAME); EoTr(m1,s1,Wr, SAME) } filter (<em>,</em>,*)</td>
<td>Assert never { EoTr(m1,s1,Rd,-); EoTr(m1,s1,Wr,SAME); EoTr(m2,s2,Rd,-); EoTr(m2,s2,Rd,SAME); {EoTr(m1,s2,Rd,SAME); EoTr(m2,s1,Rd, SAME)</td>
<td>EoTr(m2,s1,Rd,SAME); EoTr(m1,s2,Rd, SAME) }[+] filter(<em>,</em>,*) }</td>
</tr>
</tbody>
</table>
Simulation Results

- Area Overhead

<table>
<thead>
<tr>
<th>Debug Pattern</th>
<th>Lookup Table Size (# of bits)</th>
<th>#Tr Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Address</td>
<td>Data</td>
</tr>
<tr>
<td>Race Pattern 1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Race Pattern 2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Deadlock Pattern 1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Deadlock Pattern 2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Livelock Pattern 1</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Livelock Pattern 2</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

- Effect of Online Recovery

<table>
<thead>
<tr>
<th></th>
<th>Without Recovery</th>
<th>With Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td># Eating</td>
<td>6</td>
<td>3276</td>
</tr>
<tr>
<td># Resolved Deadlock</td>
<td>0</td>
<td>77</td>
</tr>
</tbody>
</table>
Conclusions

- An effective approach for NoC-based multiprocessor SoC online debugging
- Non-intrusive way to investigate, debug & recover from error states at run time
- Design tradeoffs & limitations
- Debug pattern exercise with Nirgram NoC simulator
Thank you for your listening!

Presentation By
Xiangfei KONG, Chenxi LOU

11/17/2015
Questions ?
Debate

1. Will judging the DU FIFO size be a design challenge when using the proposed online debug approach?

2. As the recovery algorithm does not work for hardware deadlock faults & inner core multithread deadlocks, is it worth to use when another approach is available?