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

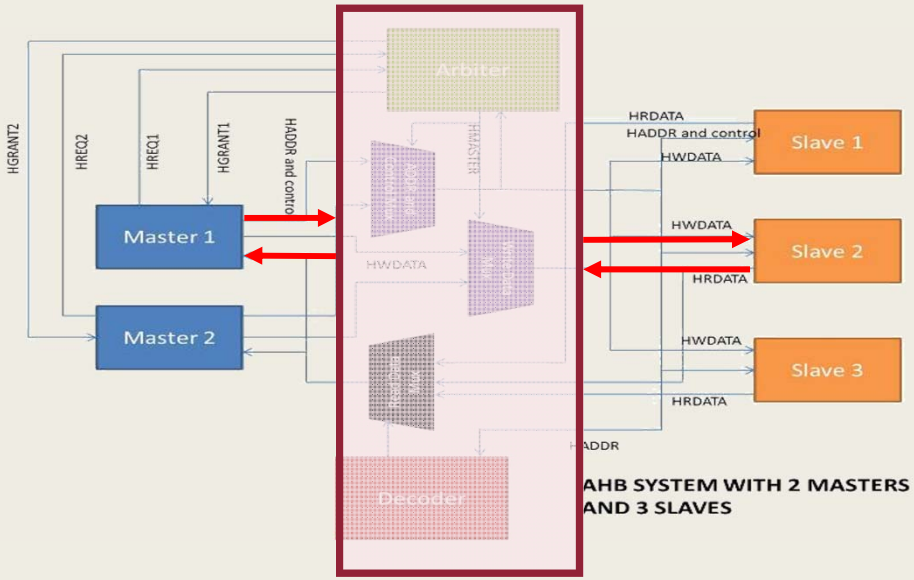
- Debug Method & Requirements
- Debug Infrastructure
- Approach Limitation
- Implementation

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

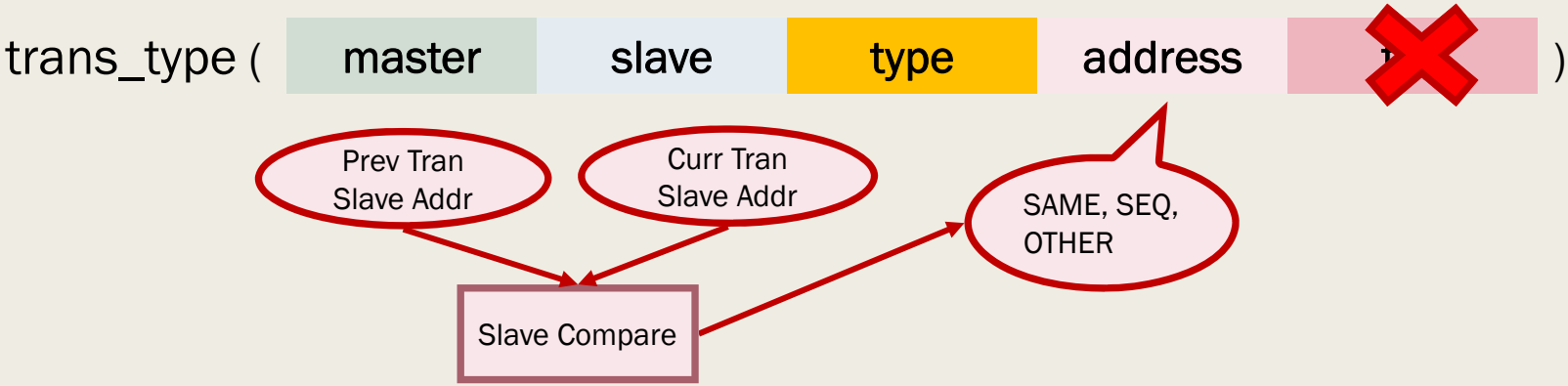
- Overview of Transaction–based Debug
- TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

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

TDPSL – Transaction Debug Pattern Specification Language

- Boolean Layer



- 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

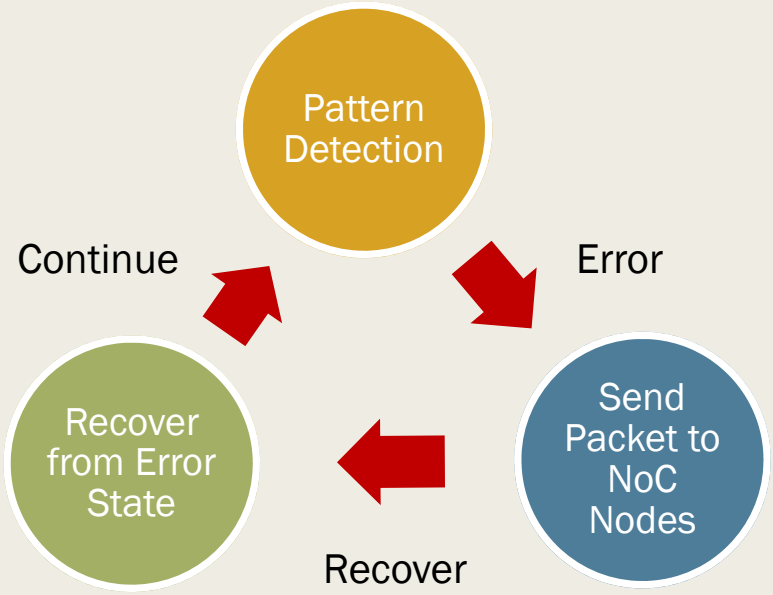
Background

- 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

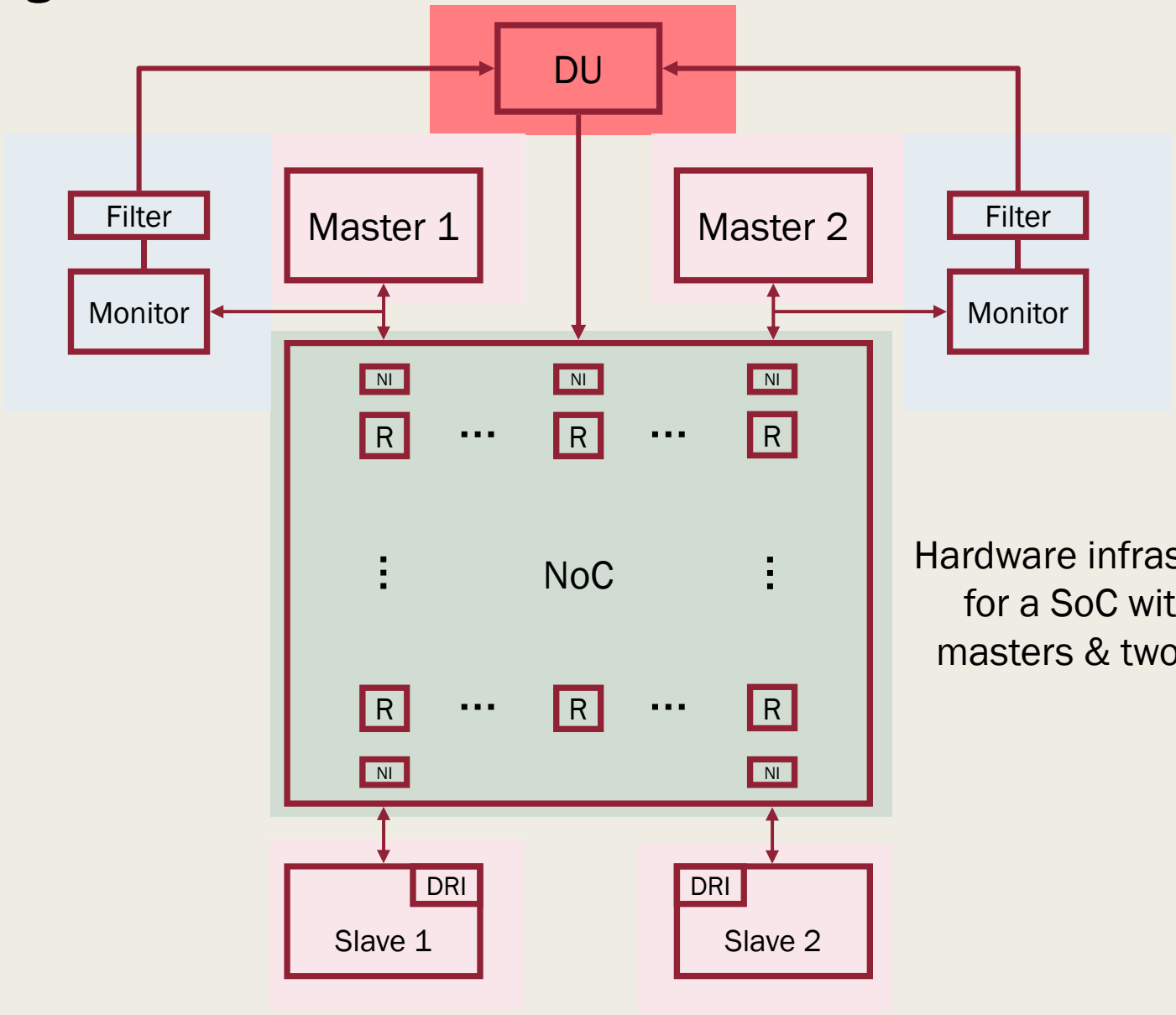
- Overview of Transaction–based Debug
- TDPSL (Transaction Debug Pattern Specification Language)

Transaction Based Online Debug

- Debug Method & Requirements

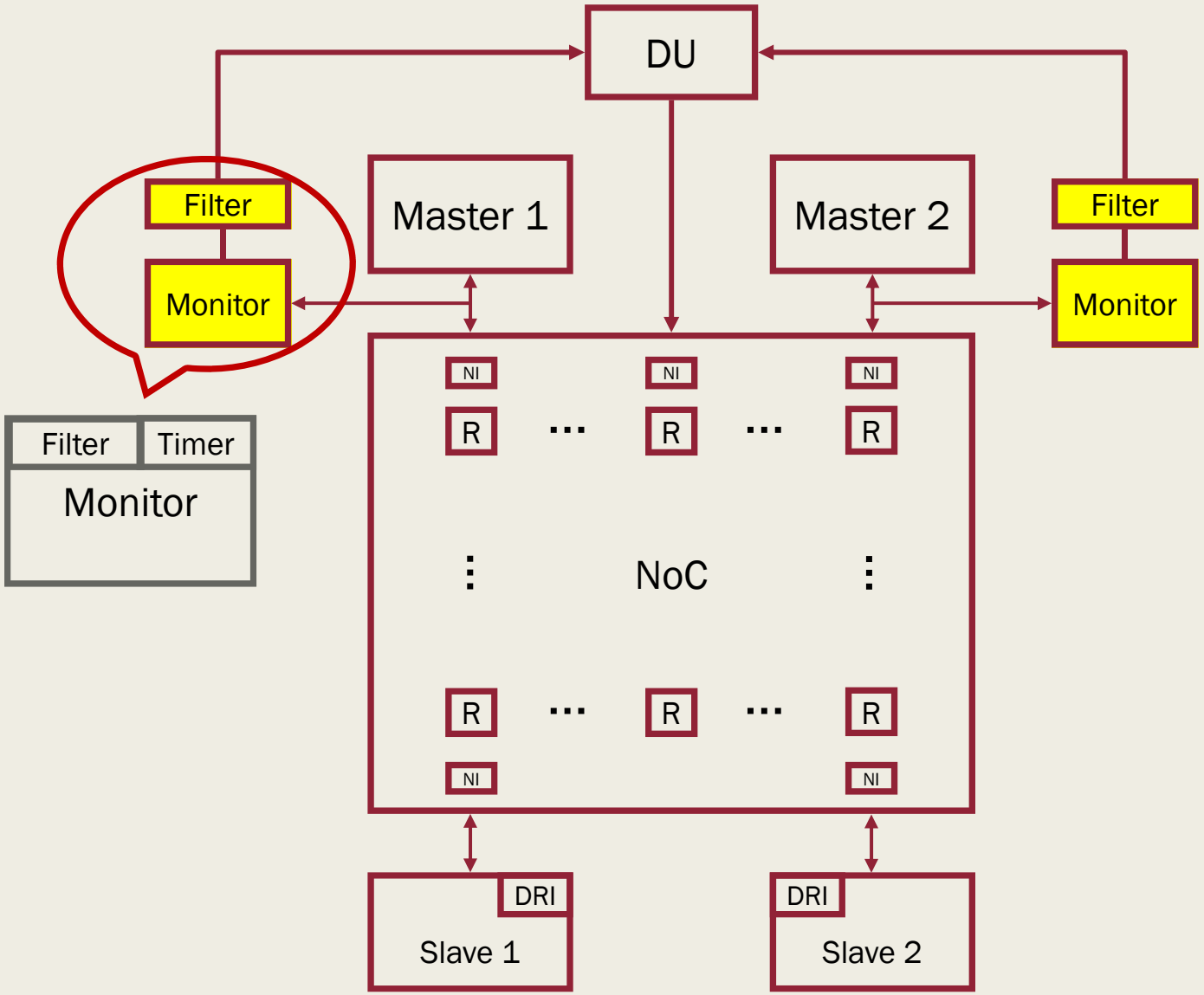
- Debug Infrastructure
- Approach Limitation
- Implementation

Debug Infrastructure



Hardware infrastructure for a SoC with two masters & two slaves

Debug Infrastructure



Monitor

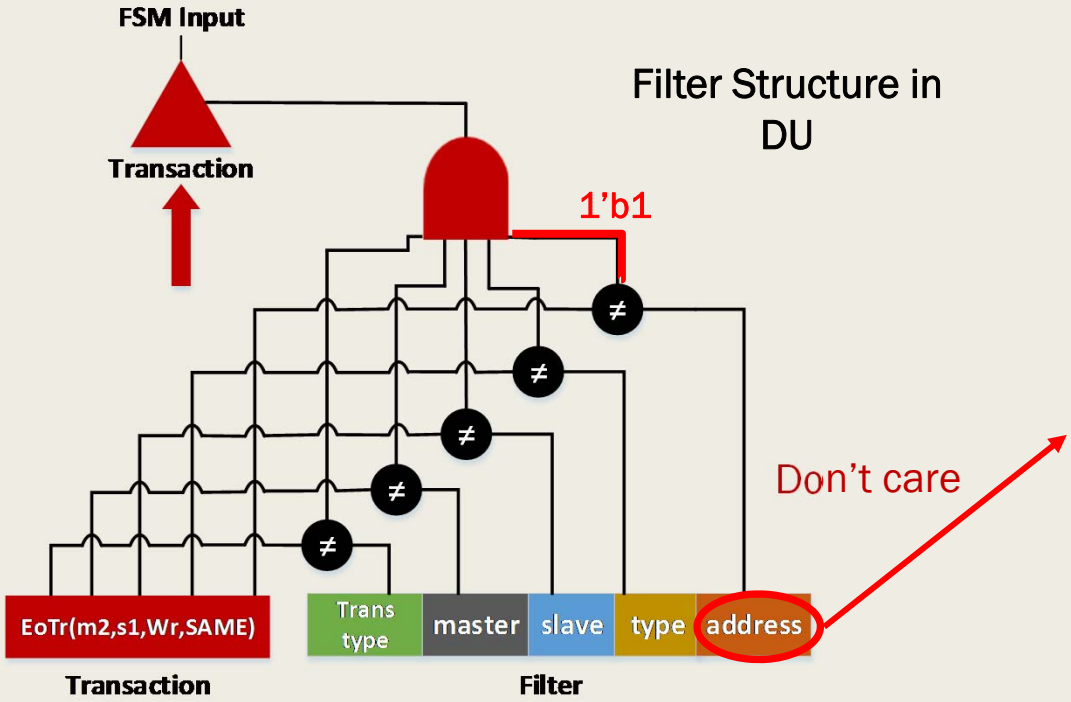
- Observe master interconnects to extract packet elements

Master Interconnect

Timestamp

Filter

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

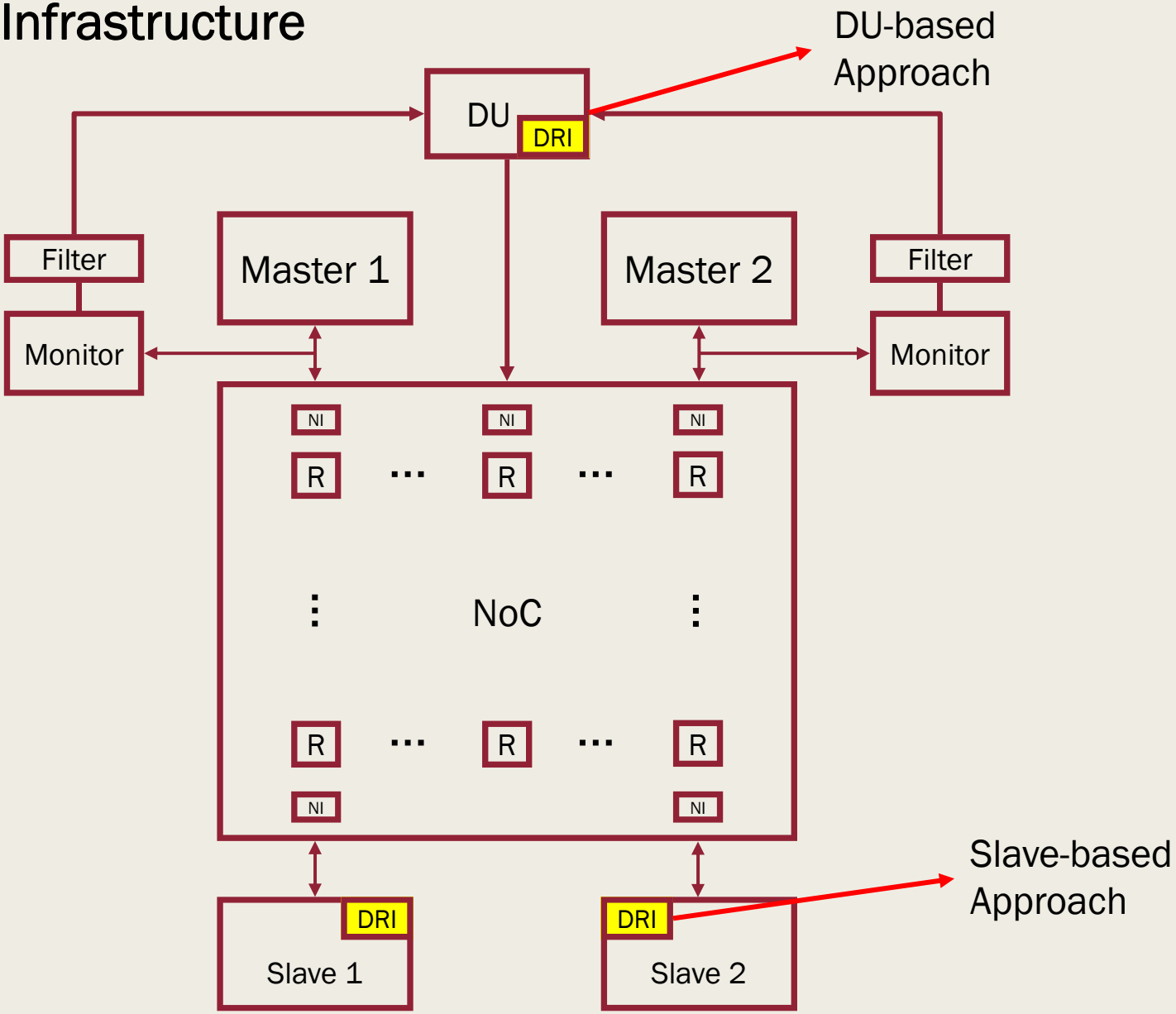


Filter Structure in DU

Don't care

Choose an always different value

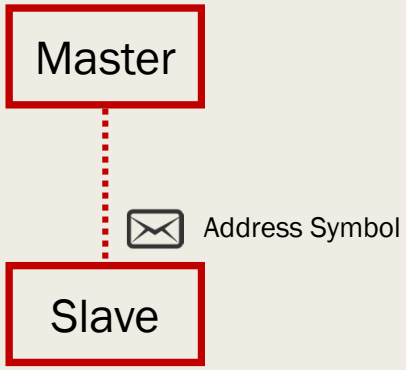
Debug Infrastructure



DRI (Debug Redundant Information)

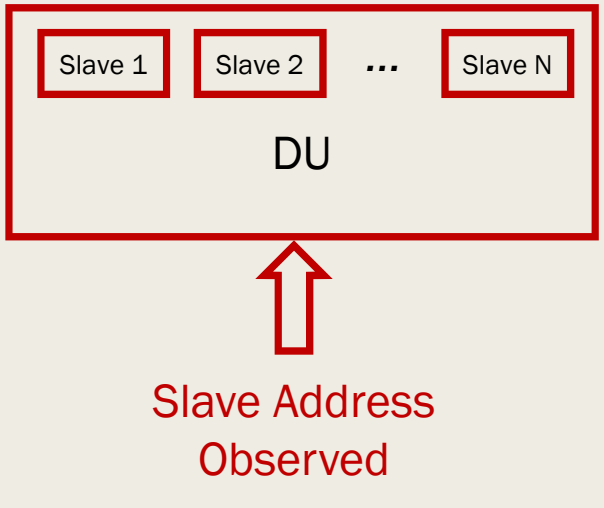
- Extract and transfer element address of a transaction

1. Slave - based Approach



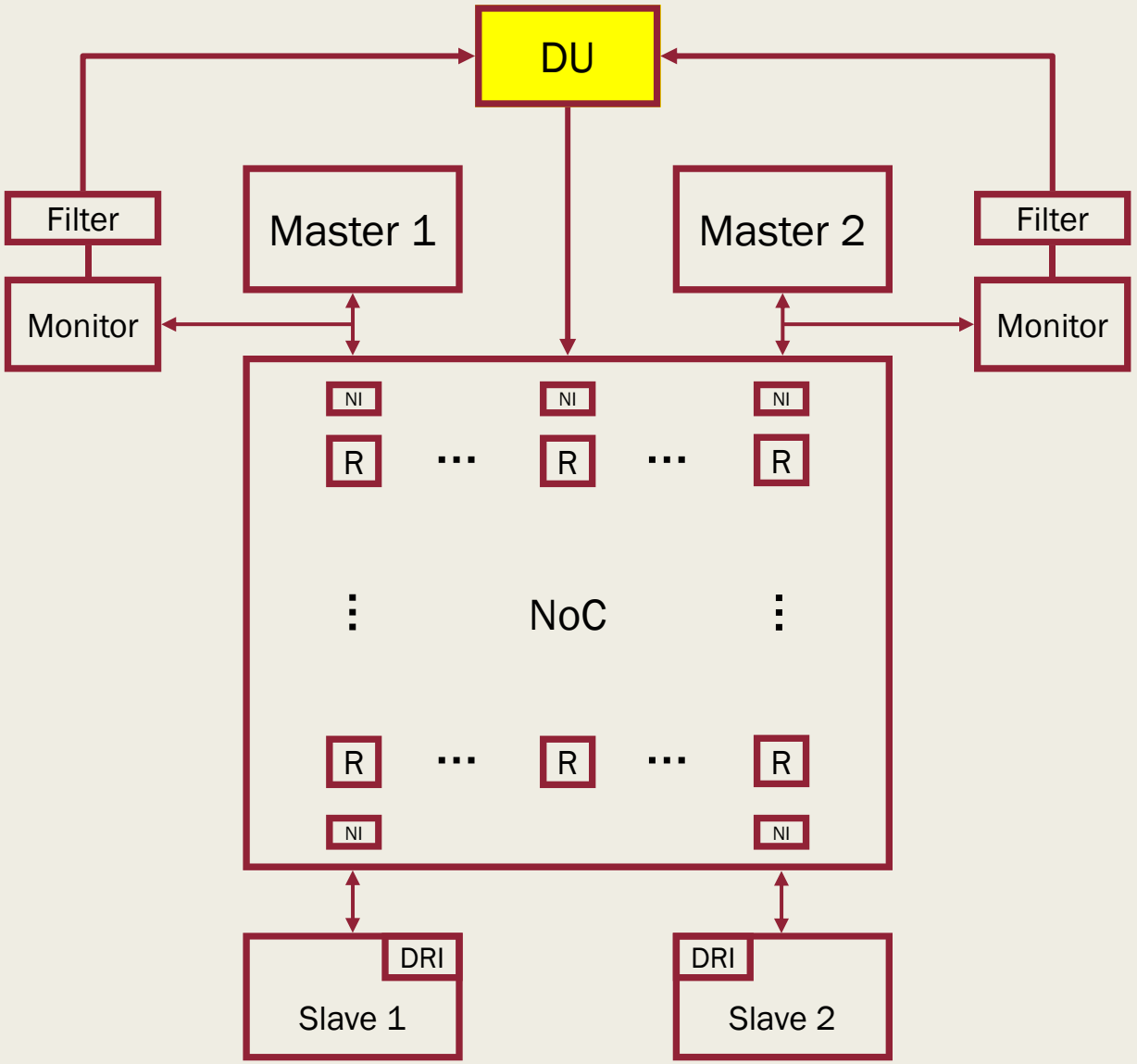
- DRI in slave
- Low transfer cost
- Address info only in EoTr, wait to receive EoTr from slave

2. DU - based Approach

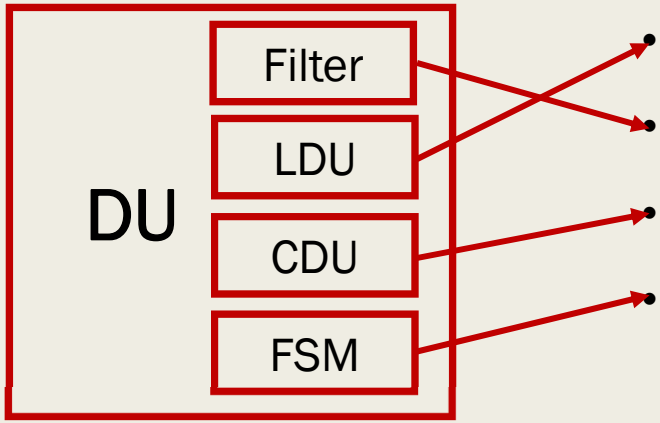


- DRI in DU
- Non-Intrusive to SoC
- More bandwidth needed to transfer slave address
- Larger memory storage in DU

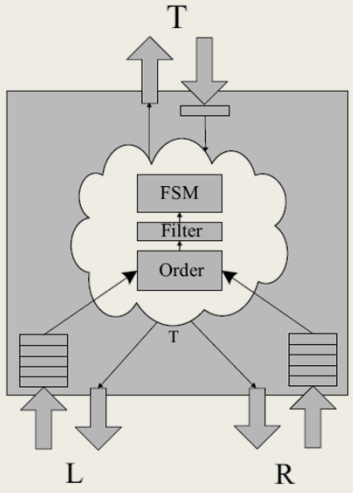
Debug Infrastructure



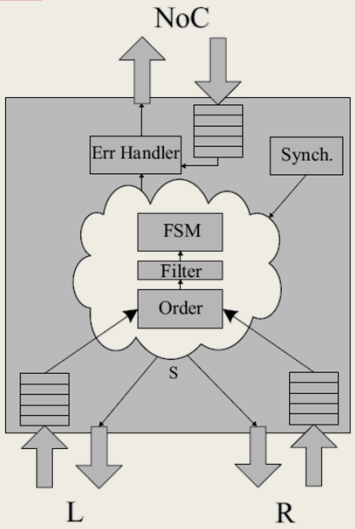
DU (Debug Unit)



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



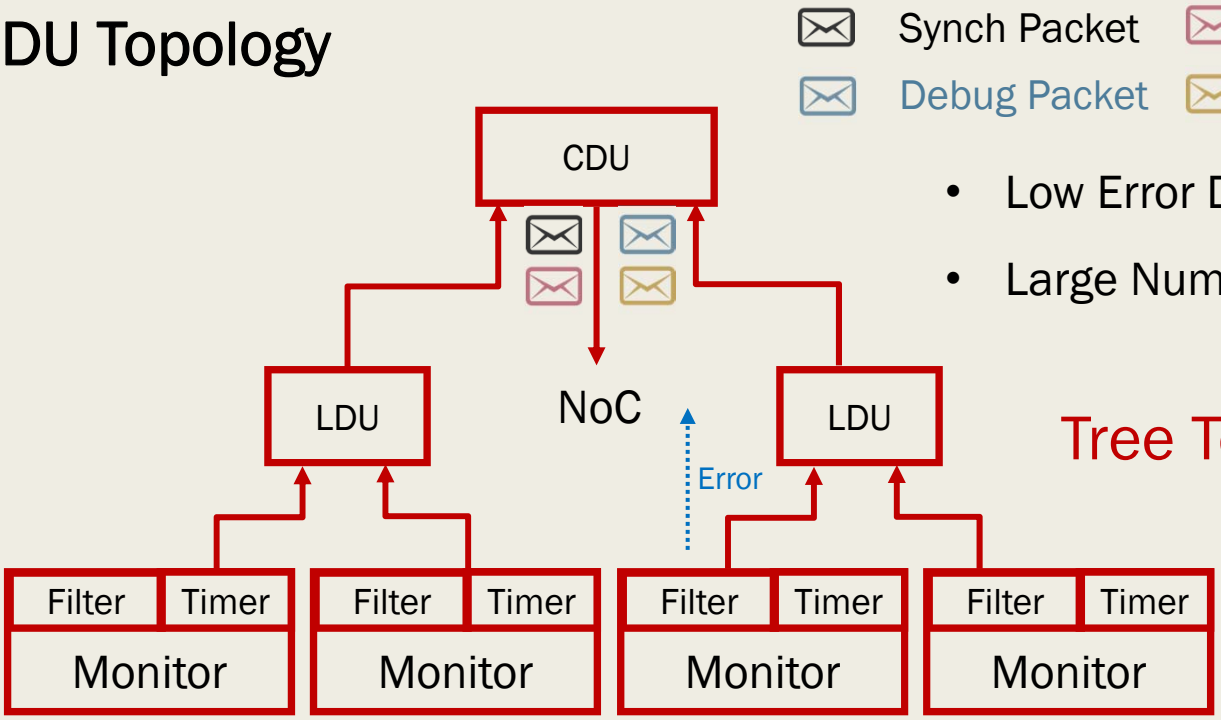
LDU Structure
(Local Debug Unit)



CDU Structure
(Central Debug Unit)

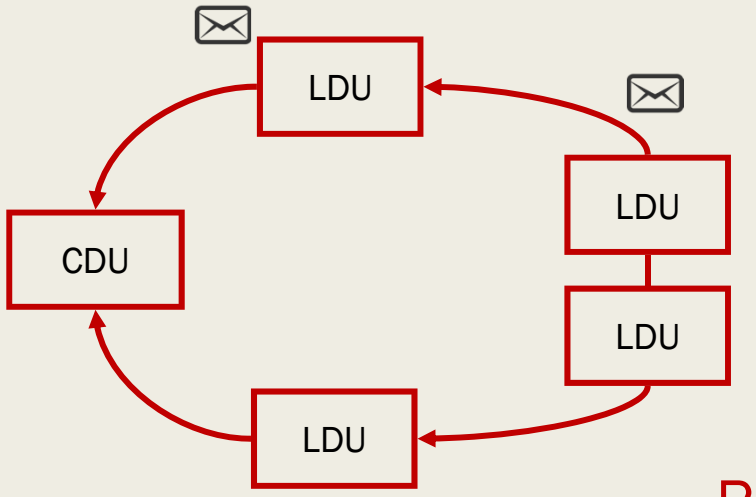
Transaction ordering
based on timestamps

DU Topology



- Low Error Detection Latency
- Large Number of DUs

Tree Topology



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

Ring Topology

Debug FSM

- *Programmable FSMs utilized to investigate assertions online*

Address		Data
Current State	Input	Next State
Start	Tr1	A
Start	Other	Start
A	$Tr2_1$	B
…	…	…
Err	-	Err

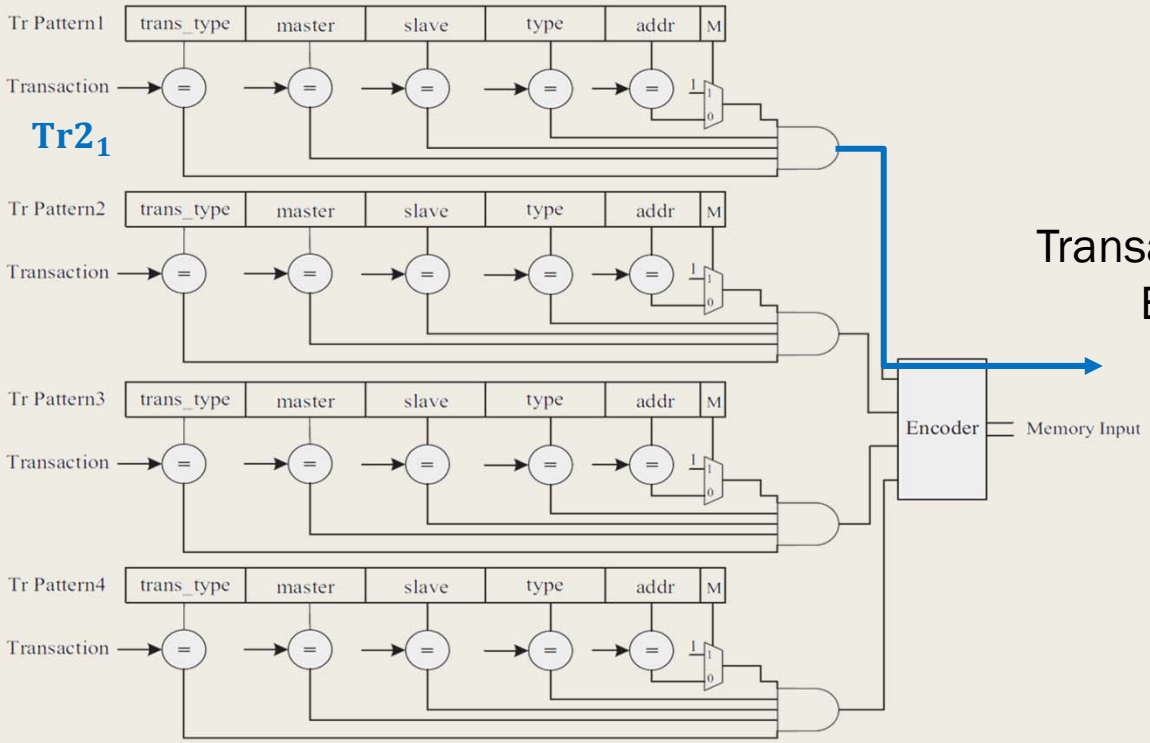
FSM Memory Overhead

Worst Case Transaction # = t

Worst Case State # = s

Total Memory = $(\lceil log_2 s \rceil + \lceil log_2 t \rceil)^2 * \lceil log_2 s \rceil$

Total Memory = $2^{\lceil log_2 s \rceil + \lceil log_2 t \rceil} * \lceil log_2 s \rceil$



Transaction Pattern Encoding

Outline

Background

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

Transaction Based Online Debug

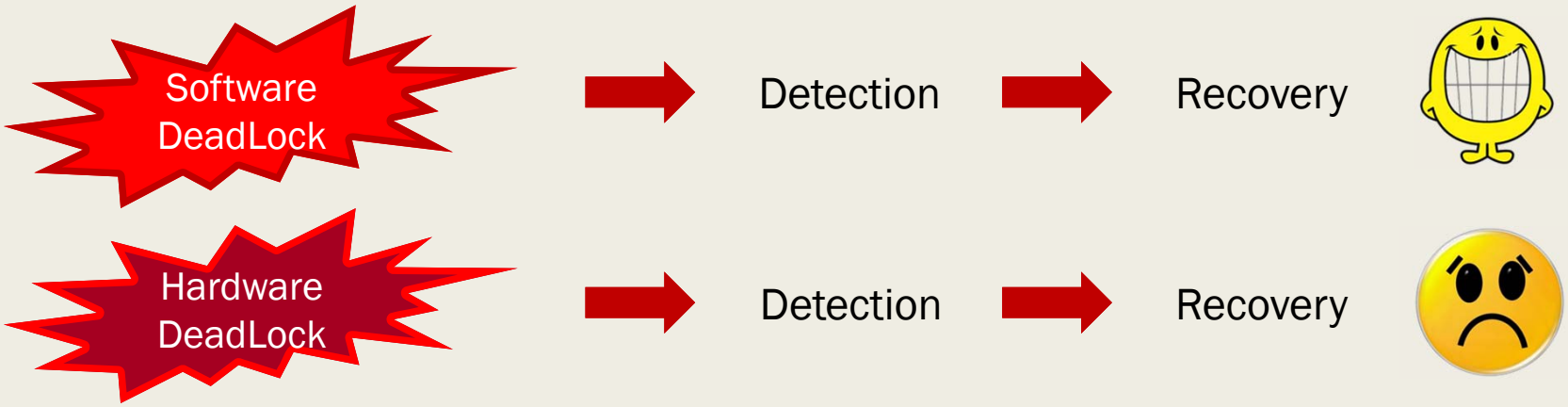
- Debug Method & Requirements
- Debug Infrastructure

- 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

Race	Deadlock	Livelock
Assert never { SoTr(m1,s1,Wr,-); SoTr(m2,s1,Wr,SAME); EoTr(m1,s1,Wr, SAME) } filter (*,*,*)	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) |EoTr(m2,s1,Rd,SAME); EoTr(m1,s2,Rd, SAME) }[+] filter(*,*,*)	Similar to Deadlock

Simulation Results

- Area Overhead

Debug Pattern	Lookup Table Size (# of bits)		#Tr Patterns
	Address	Data	
Race Pattern 1	4	2	3
Race Pattern 2	6	3	8
Deadlock Pattern 1	6	3	6
Deadlock Pattern 2	7	4	6
Livelock Pattern 1	7	4	6
Livelock Pattern 2	7	4	6

- Effect of Online Recovery

	Without Recovery	With Recovery
# Eating	6	3276
# Resolved Deadlock	0	77

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 ?

[22] A. Ghofrani, R. Parikh, S. Shamshiri, A. DeOrio, K,-T. Cheng, V. Bertacco, Comprehensive online defect analysis in on-chip networks, in: VLSI Test Symp., 2012, pp.661-666