Robust Cache Coherence Protocol Verification with Inferno

EECS 578 Project Report

Team FLY-Bee

Xiangfei Kong Zeyu Bu Yao Jiang Chenxi Lou

***Abstract* - Fault tolerant architectures are emerging to guarantee the reliable functionality of vulnerable silicon devices. However, the growing complexity of the coherence protocol and Network-on-Chip (NoC) design come to be a big challenge to pre-silicon verification. In this work, we implemented a custom designed robust MESI and directory based cache coherence protocol in System Verilog and practiced with Inferno - a software tool that operates on a logic simulation trace and automatically extracts transactions of the RTL design to evaluate the tool's help on accelerating the verification process. Experiments show that comparing with verification approaches that only use waveform viewer based debugging tools, adding Inferno into the verification process can effectively reduce the time and efforts verification engineers need to make to detect and locate potential bugs in the design. We also summarize our user experience with Inferno and present suggestions both to Inferno users and developers in this paper.**

Keywords - Robust Cache Coherence Protocol; Network-on Chip; MESI; Pre-silicon Verification; Inferno; Debugging Tool;

I. INTRODUCTION

Modern digital hardware designs tend to become more and more complex due to the customers' continuous demand for performance and the utilization of transistor scaling technology. However, the growing complexity of the hardware design is exacerbating the challenges of verification and debugging. The traditional waveform-based simulation which describes the behavior of a design in terms of changes of signals tends to be inefficient when dealing with the large scale IP and system-level design.

In this project, we present a verification example in which we use a novel verification tool – Inferno to verify a system-level design. Inferno is a more efficient verification solution which is able to establish the connection between a high-level description and low-level module behaviors, eventually, the description of the DUV's behavior is exported in the form of a transaction diagram. Such a description provides engineers great convenience when verifying the design aiming at its functionality. In our project, we decide to utilize inferno to verify a directory-based cache coherence protocol model. Since the basic MESI protocol model can be fully verified by functional verification and Murphi, we will not repeat the verification of MESI model here, instead, we implement a cache coherence protocol with resilient property in a multi-core system and use it as our design under verification (DUV).

The reminder of this report is organized as follows. Related work is discussed in Section II. Section III briefly introduces the implementation of the resilient cache coherence protocol model. The debugging process with Inferno is described in Section IV. Section V presents the experimental results by using Inferno. Some tool development suggestions will be proposed in Section VI. We hope this project can provide inspirations to verification engineers who would like to use Inferno in future.

II. RELATED WORK

A. *Inferno*

The idea to develop a brand new verification tool to automatically extract the properties is proposed in [1]. Transactional verification methodology and several study cases with Inferno are discussed in this paper.

B. *Resilient cache coherence protocol model*

The MESI coherence protocol model that could be completely verified using Murphi is presented in [5]. We decided to verify a cache coherence model with resilient property to see if Inferno gives us a more efficient verification solution when dealing with a more complicated DUV. The motivation and three major properties to bring up such a resilient cache coherence protocol model is proposed in [6]. Several transaction cases are discussed and compared with non-resilient model in the paper, which gives us an explicit understanding when implementing the resilient model.

III. DESIGN DESCRIPTION

We implemented the multi-core system architecture as shown in Fig. 1. Three single-thread cores are attached to the 2x2 mesh router network and one node is reserved for the directory, which is the central control unit connected to data memory to ensure the memory coherence. In the design, each core has its own L1 data cache, instruction cache & memory and all three cores share the data memory via the directory.

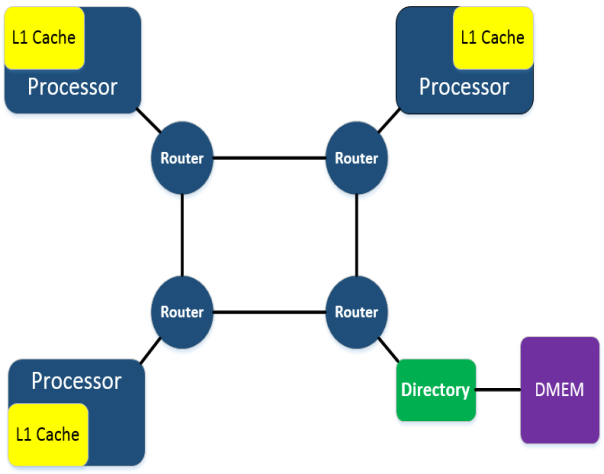

Fig. 1. Proposed Multi-core Network Architeture.

A. *Core & Data Cache Controller*

We started with the single-thread blocking processor design from course EECS 470 project provided by Xiaoming Guo and Sijia He. To modify their snoopy-bus based cache coherence protocol design to directory based design, the data cache controller was redesigned from the ground up, while most pf the other parts of design remained unchanged.

The Data Cache Controller was designed to implement basic MESI Directory Protocol at first. The state machine for implementing such protocol was described in [5], the correctness of which was thoroughly verified by Murphi when Yao took course EECS 570. In this manner, we mainly referred to this book for protocol implementation. However, some modifications were made to ensure the correctness of our design. Our router network can only receive one message from one core at a time. But sometimes the core needs to send messages to both directory and core. To ensure the correctness, two additional states, MI_A_2_SI_A and EI_A_2_SI_A, were added to send the second message. The messages that the core could inject into router network are shown in table I. If the core does not need to send command to router network, CMD_IDLE will show up at the interface.

TABLE I. TABLE OF MESSAGE VIRTUAL CHANNEL ASSIGNMENT

Message	CMD_IDLE, GetS, GetM, PutS, PutM, PutE, Fwd_GetS, Fwd_GetM, Invalid, Put_Ack, Data, Inv_Ack

The second step was to add the robustness feature to our cache design. Paper [6] was referred to for the implementation. The key point is to add transient states and additional hand shaking mechanism to guarantee the robustness of coherence protocol. Additional states and message are presented in table II.

TABLE II. TABLE OF MESSAGE VIRTUAL CHANNEL ASSIGNMENT

States	Sd, Ed, Ma, Md, lp, E_2_S, lpS, Sd_l, Ed_1
Message	Unblock, Done, Inv_Perm, Inv_Conf

Modification were made to ensure three properties.

- Initiating node must remain transient throughout the transaction.
- Previously transmitted messages can be regenerated.
- Nodes can tolerate duplicate messages and produce the same outcome.

We basically implemented the robust coherence protocol as described in paper [6]. But there is one scenario that the paper does not clearly explain, that is: when the directory is currently in E state, and one core sends getS message, the core with E state is supposed to transit to a transient state after providing the data for robust purpose. To ensure the robustness, the state machine was modified as shown in Fig. 2.

Test cases were then developed to prove the robustness of our design. The detail of our verification plan and methodology will be discussed later in this paper.

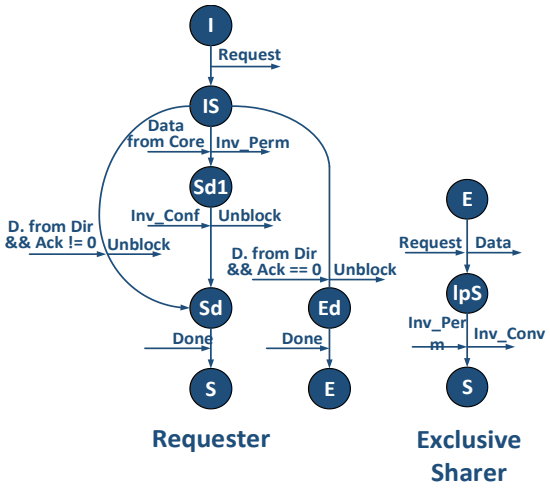

Fig. 2. The modified state machine to implement robust feature.

B. *Directory*

We created the directory and directory controller from scratch. They were designed to implement the basic MESI Directory Protocol at first, which was verified using Murphi in EECS 570. The implementation of the protocol is blocking. The directory controller will not respond to other requests until the current transaction is completed. Moreover, to prevent deadlock, the directory controller has three buffers to receive messages from different virtual channels.

After we added robust feature to the basic protocol, since more coherent messages were included and the protocol was becoming more complex, deadlocks would still happen. Instead of adding more virtual channels, we made some interesting modifications to the protocol itself. Each time the directory controller receives an unexpected message it cannot process, it will drop it from the buffer. Since our protocol is resilient, the same request will be resent and it is safe to do so. It helps to test the robust feature of our design as well.

C. *2x2 Mesh Router Network Design*

In this project, we use the open source Network-on-Chip Router RTL model from Booksim [4] to construct our interconnect network. The router model is configured to be a wormhole based design, using a simple XY routing algorithm and have four virtual channels. Each flit is 64-bit long and can either be a head, body or tail flit.

Interfaces were designed to coordinate the communication between core-router and directory-router. Core-router and directory-router interfaces are responsible for encoding the requests, data, responses into the router injection channel streams and decoding ejection channel packets into messages and data that router/directory can process. Most packets are one-flit long while the ‘Data’ and ‘PutM’ packets, as they contain the 64-bit data, will be encoded into three flits and sent to the injection channel in three consecutive cycles.

To simplify the design and ensure the expected ordering of messages received, we assign each type of message a specific virtual channel, which is shown in Table III.

TABLE III. TABLE OF MESSAGE VIRTUAL CHANNEL ASSIGNMENT

VC 0	GetS	GetM	PutS	PutM	PutE	UnBlock
VC 1	Fwd_GetS	Fwd_GetM	Invalid	Put_Ack	Done	
VC 2	Data	Inv_Ack	Inv_Perm	Inv_Conf		

IV. DEBUG WITH INFERNO

A. Inferno Overview

Inferno, an automatic semantic information extractor, is a software tool developed at University of Michigan which infers the functionality of a hardware design using Verilog source code and simulation traces. It summarizes the behavior of selected signals into transactions and presents them as a compact graph. Inferno provides verification engineers with a high level understanding of the design behavior, simplifying the timing information and categorizing repetitive behaviors as the weight in the graph. As claimed by DeOrio [1], past research has evaluated the performance of Inferno on a broad range of communication intellectual properties as well as the OpenSPARC TI 8-core processor from Sun. The project is a further research on Inferno's efficiency and verification impact.

Inferno reads in the trace file (*.vcd/*.dump) generated in the simulation and analyzes the transition of signals to generate the graph. Users have the freedom to specify some configuration options by changing the settings in *.in file. In this section, we will discuss about our user experience, Inferno's debug impact and configuration options.

B. Signal Selection

Signal selection is expected to be an important influencing factor for Inferno's debugging performance. In our practice, we tested with different selection strategies and finally agreed on the signals we would like to feed into Inferno. The target here is to choose as few signals as possible to simplify the diagram while preserving key information of the communication between cores and directory.

- Monitor signals only at router interfaces

The first choice we made is not letting Inferno probe deep into the design. Instead of choosing signals inside data cache controller or directory design to verify the inner state machines, we only monitor signals at core-router and directory-router interfaces. As the cache coherence control design can vary from protocol to protocol, monitoring only the interfaces give our research more flexibility and scalability to be used in other applications. In addition, we assume the core has previously been verified and in the research we would like to focus on the communication between cores and directory.

- Monitor only messages, sender and receiver

To minimize the number of signals we need to feed into Inferno, we chose to monitor only the sent message and its destination router address at injection channel and the received message and its corresponding requestor at ejection channel. We ignore information like data value and address as errors related to correct message but wrong transferred data can be easily detected and located. In contrast, the sent and received command can be efficiently used to infer the behavior of memory coherence control block as well as checking the NoC network's correctness. In our design, each interface has one send channel and two receive channels, which indicates that we have six fields to monitor in our Inferno diagram. An example for core 0 interface is shown in Fig. 3.

SEND CMD	Destination Router	VC1 CMD	VC1 REQUESTER	VC2 CMD	VC2 REQUESTOR
4 bits	2 bits	4 bits	2 bits	4 bits	2 bits

Fig. 3. Signals being monitored at Core-Router Interface.

- Monitor interfaces separately

Intuitively the best option is to monitor all commands and their destination/initiator in one diagram, which includes all passing messages in the NoC network. The unified graph contains all information and verification engineers only need to read this single transaction diagram. However, we chose to generate diagrams for each interface separately as in the debugging process we found the unified graph is usually too complex to understand. As discussed before, in binary form each interface needs 18 bits to compose the information we need and it comes to be 72 bits for 4 interfaces. Even if we show signals in hexadecimal, there will still be 24 bits, which is still a big number to track through. Also, the behavior of the complete network in many times is too complex to be categorized into a few transactions. Comparing to the separated graphs, it complicates the verification engineer's task of locating the bug source. Another concern is about the scalability. As the NoC network implemented in the project is a simple 2x2 network, we may still be able to understand the unified graph. However, when the design grows into a larger network, the approach will sooner or later be inapplicable. Monitoring interfaces separately provides users with more scalability and simplified transaction diagrams. Typical interfaces like directory and main initiator core interface are likely to have more information than other interfaces and separated transaction diagrams give users the freedom to break the big verification task into small goals and look across different transaction diagrams one after another.

C. Don't Care Signal Masking

To use Inferno efficiently, some minimal changes in the design can help as well. An interface is in idle state when nothing need to be sent and nothing is received. In our original design, even if the command messages are CMD_IDLE, destination or requestor may change, leading to 'redundant states' in transaction diagrams. As shown in Fig. 4. Transaction of sending a single invalid permission message to core 1 is categorized into several 'IDLE states'.

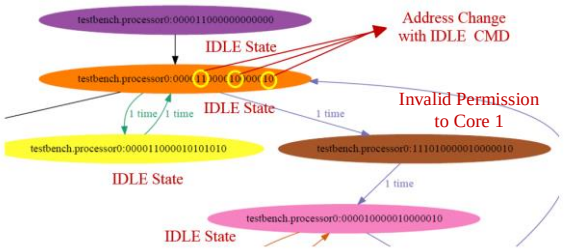

Fig. 4. A snapshot of Core 0 interface transaction diagram with no 'don't care' signal masking (states in binary).

To simplify the transaction diagram, we decided to mask destination and requestor address when no valid command messages is placed at the interface. After the design change, the only interface idle state is always shown as all zeros. Fig. 5 shows the simplified transaction after design change.

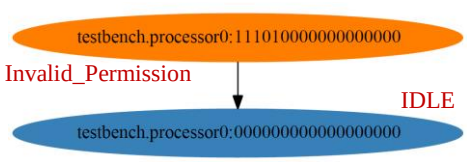

Fig. 5. A snapshot of Core 0 interface transaction diagram after 'don't care' signal masking (states in binary).

D. Inferno Configuration Options

There are some configuration options in Inferno that are user-defined. Main ones are: 1) strip_leading_zeros 2) show_hex 3) ignore_states. We investigated all of these options and made the configuration choices based on our preference.

- Strip_leading_zeros

Option of striping leading zeros gives users a chance to remove leading zeros from diagrams. This can be helpful when leading bits are redundant information in some cases. Since we are not in such a case, the option is not useful for us.

- Show_hex

Showing states in hexadecimal instead of binary can be a great help to some users, especially when user would like to monitor a large number of signals. Inferno provides such an option that users can choose. In our debugging process, we also found the option to be great as in hex each bit in the state now indicates one signal we are monitoring.

- Ignore_states

Ignore_states is another option we found helpful. In fact, the option here let you specify typical signal scenarios that you don't care about and remove these scenarios from the diagram. In our case, we don't care scenarios when sent or received commands are CMD_IDLE. Fig. 3 shows a comparison of diagrams with and without ignore_states.

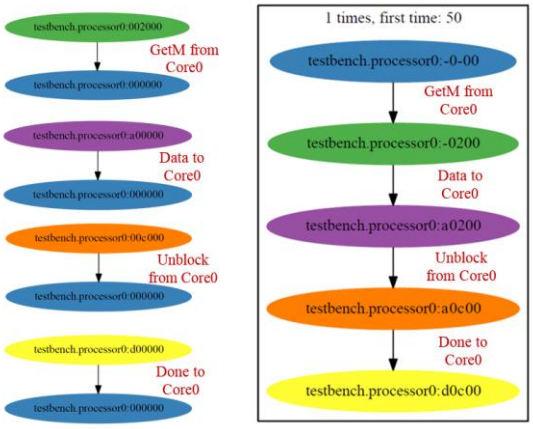

Fig. 6. Comparison between Inferno generated transaction diagrams with different configurations - left) without ignore_states option right) with ignore_states option (states in hex).

E. Debug Examples

- Core/DCache Bug Found with Inferno

TABLE IV. TEST CASE FOR VERIFYING THE PROTOCOL

Core 0	Core 1	Core 2
load 0x20	•	•
	load 0x20	•
		store 0x20

Table IV shows one test case for verifying the correctness of our protocol. The expected message sent and received by Core 2 is as follows:

Send GetM -> Receive Data -> Receive Inv_Ack from Core 0 and Core 1 -> Send Unblock -> Receive Done

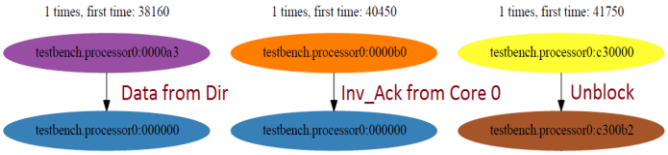

Fig. 7. The Inferno Simulation Result of Core 2 for the test case shown in Table IV

But when we run the Inferno with such test case, part of the result we got is shown in Fig. 7. From the result, it can be seen clearly that Core 2 sends Unblock to Directory right after receiving only one Inv_Ack.

With the help of Inferno, we located the bug easily, and the occurrence of the bug was due to the improper mechanism of comparing the Inv_Ack received and the one expected.

The simulation result after debugging is shown as Figure 8. We can see from this graph that, as expected, the core 2 sends Unblock message to Dir after receiving all expected Inv_Ack.

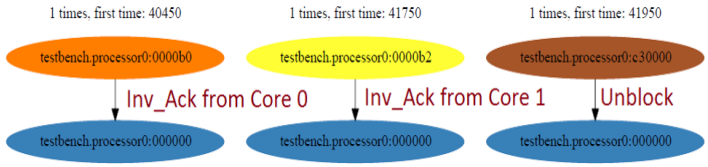

Fig. 8. The Inferno Simulation Result of Core 2 after dubugging

- Directory Bug Found with Inferno

TABLE V. TEST CASE FOR VERIFYING THE PROTOCOL

Core 0	Core 1	Core 2
store 0x20	•	•
	store 0x20	•
		•

Table V shows one simple test sequence which was captured by Inferno in our simulation. The generated transactions are illustrated by Fig. 9. The left graph shows a generated transaction corresponding to our design with a certain bug. The right graph shows a generated transaction corresponding to our design after that bug was resolved. And Fig. 10 illustrates the signals being monitored at Directory-Router Interface.

According to the left graph, the directory receives a GetM message from Core 1 first. Then the directory send a Forward-GetM to Core 0 which is the owner. Later the directory receives repeated GetM messages from Core1 for 45 times without getting any Unblock message. The directory's responses are correct while Core1 cannot finish the transaction. This is more like a design bug because normally there will not be so many repeated messages. It is very likely that Core 1 cannot receive data from Core0 or Core 1 receives Data but responds incorrectly. In this specific situation, Inferno helps us to quickly find and locate a bug even though it cannot be used to further trace the bug. After we correct our design of the cache controller, the generated transaction is shown by the right graph of Fig. 9. The directory successfully receives the Unblock message from Core 1 and sends Done message back. This corresponds to our expected transaction.

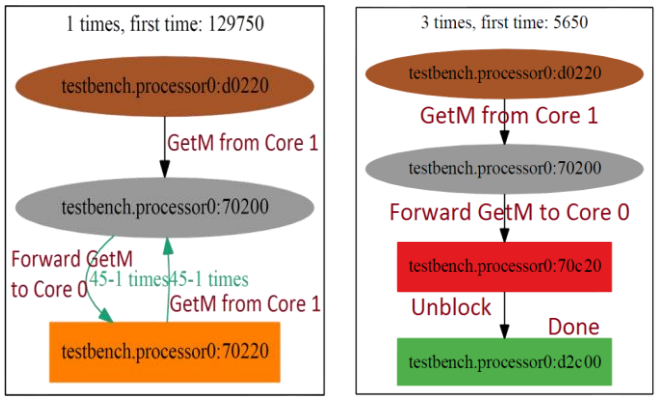

Fig. 9. The Inferno Simulation Result of Directory.

V. Experiments & Results

For our project, we developed assembly code tests to verify both the baseline design and the robustness feature. All major design bugs found during the debugging process were documented in the debug log, an online google form for all team members to record the date, problem found, bug source and efforts made to find the bug. Inferno was put into use in our project after the baseline design was done. In this section, we will give a brief introduction to the tests we developed and our evaluation on Inferno's help to the debugging process.

A. Tests

We developed assembly level tests to test the baseline design, robustness feature and Inferno. As each core has its own instruction memory, a test set includes three *.s files for each core. Total number of test sets developed is around 10 and these tests exercise typical scenarios that will happen between cores and directory.

To verify the robust coherence protocol, we add four packet drop control signals at each router's ejection channel. The control signal is used to explicitly mask the received packets to model conditions like packet drop or packet corruption in an unreliable NoC system. We developed test cases that drop flits, packets or create false positive to send packet duplicates to challenge the robustness of the cache coherence protocol.

B. Debug Log

The entire debug log is attached in the appendix. In total, we found around 65 major design bugs in our debugging process. Each entry contains following fields: 1) Date 2) Time 3) Problem 4) Debug person 5) Time taken to locate the bug 6) Bug source 7) Bug description 8) Use Inferno or not. Bug sources are categorized into Core, Router and Directory. For debug effort, we define the efforts we made to find and locate the bug quantitatively with the equation:

$$Effort = \frac{Time\ Taken}{\#\ of\ Debug\ Person}$$

After analyzing the debug log, we found generally Inferno did help us to locate the bug faster and more easily.

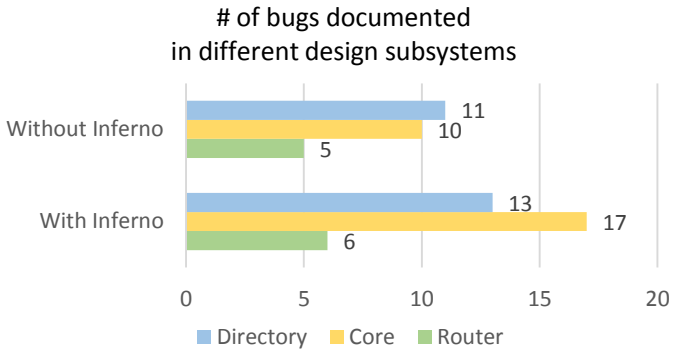

Fig. 10. Numer of bugs found with/without Inferno in each sub design system.

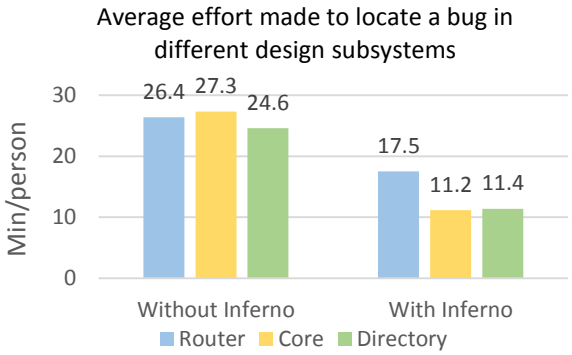

Fig. 11. Average debug effort made for detecting and locating a bug with/without Inferno in each sub design system.

Fig. 11 shows the average effort made to detect and locate a bug in different subsystems without and with Inferno's assistance.

VI. Tool Development Suggestion

Inferno brought a lot of convenience to our debugging process. However, we also found some defects in the current version of the tool that can be further improved in future. In this section, we will present these defects and our suggestions to the tool developers.

A. Transaction Capturing

After some practice with Inferno, we found that in some cases Inferno was unable to capture transactions that were happening only once. The conclusion was confirmed by

comparing Inferno transaction diagram with DVE waveform as well as the source vcd file. At the same time, when the behavior happened a few times, Inferno successfully included the missed transaction, while the weight (happening times) was less than expected.

B. *Mess wth Igonore_states*

We discussed about the configuration option of ignore_states. Unfortunately, although enabling the option can give a compact and high-level transaction diagram, it is not always informative. When the CMD_IDLE message is specified to be ignored, in some cases the generated diagrams can be quite messy to understand. In Fig. 12, the pink oval is the state sending an unblock message to core 3 and the red one is the state receiving the done. The grey block turned out to be a redundant one generated after turning on the ignore_states option.

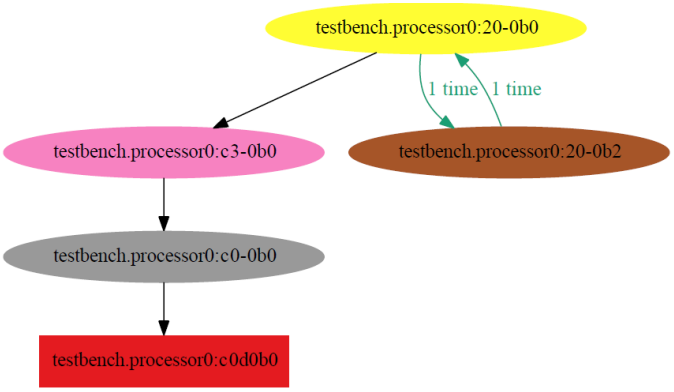

Fig. 12. Snapshot of core interface transaction graph with ignore_states option.

C. *Suggestions for Inferno Development*

Based on our user experience, we would like to provide some suggestions to Inferno developers to make the tool stronger.

- Support Enumeration

Supporting enumeration is considered to be a great feature for Inferno to implement from our perspective. In our case, we hope to see commands as abstract messages instead of binary/hex bits. Fig. 13 shows a comparison between an Inferno generated state form and the state form we expect.

Current	**a**	**2**	**2**	**0**	**0**	**0**
Expected	Data	2	CMD_IDLE	0	CMD_IDLE	0

Fig. 13. Comparison between current state components and proposed state form that supports enueration.

- Signal Highlight

For different signals selected to be analyzed in Inferno, they can have different priorities. In our design, we definitely care more about the send and received transaction messages than the destination and requestor ID. It can be helpful if users can specify typical signals to highlight in configuration setting so that these 'high priority signals' can be identified at first sight.

Current	**a**	**2**	**2**	**0**	**0**	**0**
Expected	Data	2	CMD_IDLE	0	CMD_IDLE	0

Fig. 14. Comparison between current state components and proposed state form that supports enueration and highlight.

- Ignore_states Improvement

As claimed before, the current ignore_states option in fact can only mask unwanted signal values. After correlating different signals into one state, we also expect the tool to have the flexibility of masking the combination. An example can be the interface idle state 000000. We also believe the improvement can potentially eliminate the need of signal masking, requiring no design change to match Inferno verification methodology.

VII. Future Work

In this project, we focused on the robust coherence protocol design and the debugging process with Inferno. As presented in the paper, we investigated Inferno's performance on locating a design bug and our user experience with the tool. Some work can be done in future like the evaluation of transaction coverage, integrating assertions into Inferno and tool development.

VIII. Conclusion

The growing complexity of memory coherence protocols and NoC network design raises the need for novel verification approaches and debugging tools. In this paper, we presented our experience of using Inferno, a semantic extractor in our debugging process and evaluated its effectiveness and potential to be used in future debugging process. We discussed about our design, a MESI, directory based multi-core system with customized data cache and directory state machines to ensure the system's robustness in an unreliable NoC network. We also talked about the configuration options of Inferno to generate a clear transaction diagram and present our suggestions both to Inferno users and developers. As shown in our experimental results section, in average Inferno accelerated the debugging process and reduced efforts engineers need to make to locate the bug. We hope our project can inspire Inferno users and developers and further improve the tool.

Acknowlegement

We send our sincere thanks to Prof. Bertacco and Doowon Lee for their great help and guidance on our project. We also want to express our thanks to Xiaoming and Sijia for providing their EECS 470 project multi-core RTL design to us.

References

[1] A. DeOrio, A. D. Bauserman, V. Bertacco and B. C. Isaksen, "Inferno: Streamlining Verification With Inferred Semantics," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 5, pp. 728-741, 2009.

[2] A. DeOrio, A. B. Bauserman, V. Bertacco and B. C. Isaksen, "INFERNO-An automatic semantic information extractor," University of Michigan, 2009. [Online]. Available: http://wwweb.eecs.umich.edu/inferno/.

[3] D. U. Becker, "Open Source Network-on-Chip Router RTL," Stanford University, 2012. [Online]. Available: http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router.

[4] D. U. Becker, "Efficient Microarchitecture for Network-on-Chip Routers," Stanford University, Stanford, August 2012.

[5] D. J. Sorin, M. D. Hill and D. A. Wood, A Primer on Memory Consistency and Cache Coherence, Madison: Morgan & Claypool, 2011.

[6] K. Aisopos and L.-S. Peh, "A Systematic Methodology to Develop Resilient Cache Coherance Protcols," *MICRO'11*, 2011.

APPENDIX – DEBUG LOG

Without Inferno	Date	Time	Problem	Debug Person	Time taken	Bug Source	Bug Description
Router 5 bugs found Average Debug Effort: 26.4 min/person	11/7/2015	12:30 PM	Only head flit, lost body and tail flit	Xiangfei	20 min	Router	Should include vc info in the flits as well
	11/7/2015	4:00 PM	Wrong routing output port	Xiangfei	45 min	Router	Router ids allocated do not conform to the routing logic
	11/9/2015	11:00 PM	No command on router receive node	Xiangfei	1 hour	Router	Virtual channel allocation
	11/11/2015	3:00 PM	Partial correct router pass, wrong data and command	Xiangfei & Chenxi	20 min	Router	Verilog Bug, forgot begin end in if statement
	11/11/2015	4:00 PM	Wrong ack_cnt at receiver side	Xiangfei & Chenxi	10 min	Router	Router: Wrong decode for channel
Core 10 bugs found Average Debug Effort: 27.3 min/person	11/9/2015	5:30 PM	The data written to cache is incorrect	Zeyu	30min	Cache Controller	The IM_ID state outputs wrong value to cache
	11/9/2015	6:00 PM	The data read from cache has X	Zeyu	20min	Cache Controller	Some previous unused logic cause X
	11/11/2015	9:00 AM	Concurrent M and S in the multicore system	Zeyu & Yao	55min	Cache Controller	Does not collect all Inv_Ack before a transition to M
	11/11/2015	10:40 PM	The Fwd_GetS Message in cache controller input buffer is lost	Zeyu	10min	Cache Controller	Faulty Read_en and of the virtual input buffer
	11/11/2015	10:55 PM	Cache controller met an unexpected message	Zeyu	10min	Cache Controller	Faulty logic of Read_en of the virtual input buffer
	11/11/2015	11:20 PM	Cache controller cannot consume Fwd_GetS	Zeyu	15min	Cache Controller	Faulty state machine transition
	11/11/2015	11:39 PM	Cache controller cannot send response of Fwd_GetS	Zeyu	15min	Cache Controller	Faulty state machine transition
	11/11/2015	11:56 PM	Cache controller sends response of Fwd_GetM to wrong destination	Zeyu	15min	Cache Controller	A false default value is assigned to destination
	11/11/2015	12:35 AM	Faulty memory state after execution	Yao & Zeyu	35 min	Core	"Halt" is raised before writing back cache data
	11/11/2015	5:20 PM	Cannot end the execution of the core until timeout	Yao & Zeyu	30 min	Integration	Halt signal of the core is not connected
Directory 11 bugs found Average Debug Effort: 24.6 min/person	11/9/2015	7:50 AM	Directory does not consume the GetM message	Yao	10 min	Directory	Faulty Write_en of the virtual input buffer
	11/9/2015	8:11 AM	Directory does not consume the GetM message	Yao	15 min	Directory	Faulty Read_en and Reset of the virtual input buffer
	11/10/2015	4:30 PM	Directory does not return the response to the GetM message	Yao	20 min	Directory	Faulty state machine transition
	11/10/2015	4:50 PM	Directory does not return the response to the GetM message	Yao	10 min	Directory	Address is not given to DMEM
	11/10/2015	5:00 PM	Response to the GetM message has unknown "ack" signal	Yao	10 min	Directory	"Ack" is not assigned
	11/10/2015	6:00 PM	The GetS Message in directory input buffer is overwritten	Yao	30 min	Directory	Faulty logic of the Write_en of the virtual input buffer

	11/10/2015	7:00 PM	Duplicates of responses to GetS Message	Yao	20 min	Directory	Does not clear signals on the interface
	11/11/2015	11:00 AM	Directory does not consume the written back Data	Yao	30 min	Directory	Faulty state machine transition
	11/11/2015	6:40 PM	Cache controller cannot consume Fwd_GetM	Zeyu & Yao	20min	Directory	False destination of Fwd_GetM is assigned
	11/11/2015	8:40 PM	Cache controller cannot consume Fwd_GetM	Zeyu & Yao	35min	Directory	Owner is not written corresponding directory entry
	11/11/2015	9:35 PM	Faulty memory state after execution	Zeyu & Yao	45min	Directory	The data in a PutM is not written back to DMEM
With Inferno	**Date**	**Time**	**Problem**	**Debug Person**	**Time taken**	**Bug Source**	**Bug Description**
Router 6 bugs found Average Debug Effort: 17.5 min/person	11/15/2015	9:00 PM	After sending several messages using one channel, channel completely corrupted	Xiangfei	30 min	Router	Incorrect flit sink scheme in previous design
	11/18/2015	1:00 PM	Received but virtual channel changed during transmission	Xiangfei & Chenxi	10 min	Router	VC allocator issue
	11/19/2015	11:30 PM	Wrong requester encoded in the transmitted message	Xiangfei	15 min	Router	Directory interface design error
	11/22/2015	10:00 AM	Didn't receive invalid_confim message as expected	Xiangfei	5 min	Router	Packet arrived, but forget to identify the new added message type
	11/27/2015	10:43 AM	Do not receive requests at destination node at receiver side	Xiangfei	10 min	Router	Forget to put drop_packet_ctrl signal back to 1
	11/28/2015	11:30 AM	Core keeps sending request, but not arrive at destination node	Xiangfei & Chenxi	30 min	Router	Virtual channel encryption error after changing to resilient
Core 17 bugs found Average Debug Effort: 11.2 min/person	11/15/2105	3:00 PM	Data message corrupted	Xiangfei	20 min	Core	Sending data message needs 3 cycles for Router to process
	11/15/2015	5:13 PM	State Transition from IS_D is wrong	Zeyu	10 min	Cache Controller	Used wrong signal as "if" conditions
	11/15/2015	6:17 PM	proc2router_dest signal sends XX to router	Zeyu	3 min	Cache Controller	The stall signal for FIFO is not set to 1
	11/15/2015	6:30 PM	State Transition from I to E is wrong	Zeyu	17 min	Cache Controller	Assignment to next state is wrong
	11/15/2015	7:39 PM	current state is stuck at XX	Zeyu	21 min	Cache Controller	Assignment to current state miss the consideration of bus command
	11/15/2015	9:11 PM	core didn't response fwd_getM	Zeyu	9 min	Cache Controller	Miss the process of fwd_getM
	11/15/2015	10:44 PM	State stuck at IM_A	Zeyu	21 min	Cache Controller	Didn't take the ack received at IM_AD state into account
	11/18/2015	3:16 PM	Didn't send unblock signal	Zeyu	9 min	Cache Controller	The condition for sending unblock is wrong
	11/18/2015	5:07 PM	cache controller skip the store command from core	Zeyu	19 min	Cache Controller	the is_done signal sent to core is wrong
	11/18/2015	9:41 PM	Data sent to router is wrong	Zeyu	7 min	Cache Controller	address sent to cache is wrong
	11/28/2015	2:00 PM	Always dropping the packet, but the initiator node didn't keep resending requests	Xiangfei	15 min	Core	Core's state machine problem

	11/29/2015	6:53 PM	unblock signal is not resent when the core didn't receive done for long time	Zeyu Bu	11 min	Cache Controller	The waiting time is set too long
	11/29/2015	8:39 PM	invalid permission is not resent when the core didn't receive ack for long time	Zeyu	7 min	Cache Controller	Forgot reset the waiting time
	12/3/2015	4:14 PM	cannot stand duplicate inv_ack signal	Zeyu	8 min	Cache Controller	Should not use ack count anymore
	12/3/2015	5:49 PM	Didn't go to Md state	Zeyu	3 min	Cache Controller	State transition condition wrong
	12/5/2015	9:27 PM	Data cannot be resent from E state to S state	Zeyu	5 min	Cache Controller	Didn't consider such transition as robust
	12/5/2015	11:38 PM	proc2router_dest signal sends XX to router	Zeyu	6 min	Cache Controller	buffer stall signal is not set to 1 at previous state
Directory 13 bugs found Average Debug Effort: 11.4 min/person	11/15/2015	11:30 PM	Invalidate message has wrong requester field	Yao	5min	Directory	Does not assign requester field in that state
	11/15/2015	11:40 PM	When directory is in M and forwards a GetM, it directly goes to stable M state	Yao	15min	Directory	Does not add robust feature in that situation
	11/16/2015	9:42 AM	When directory is in S and responds to a GetM, it directly goes to stable M state	Yao	8min	Directory	Does not add robust feature in that situation
	11/16/2015	10:15 AM	The repeated GetM message is blocked by other requests in the buffer	Yao	18min	Directory	Should drop other requests
	11/16/2015	3:15 PM	Directory receives repeated GetS but sends done back	Yao	18min	Directory	Wrong "if else" block in that state
	11/16/2015	5:00 PM	When directory is in E and forwards a GetM, it directly goes to stable M state	Yao	15min	Directory	Does not add robust feature in that situation
	11/17/2015	3:15 PM	When directory sends Invalidate message to unexpected node	Yao	13min	Directory	Assign the wrong destination
	11/17/2015	4:20 PM	When directory is in S and receives repeated GetM, it does not resend invalidate message	Yao	11min	Directory	Clear sharer list too early
	12/3/2015	3:05 PM	When directory is in S and receives repeated GetE, it does not resend invalidate message	Yao	10min	Directory	Clear sharer list too early
	12/3/2015	3:45 PM	When directory is waiting for one node's unblock, it responds to other GetM	Yao	11min	Directory	Directory mistakes it for a repeated GetM because it does not compare the requester.
	12/3/2015	4:05 PM	When directory is waiting for one node's unblock, it responds to other GetS	Yao	2min	Directory	Directory mistakes it for a repeated GetS because it does not compare the requester.
	12/4/2015	3:45 PM	When directory receives a repeated GetS, it sends an invalid Data back	Yao	11min	Directory	Directory does not retain the data gotten from memory
	12/4/2015	4:21 PM	Directory receives an unblock message, but sends Data back.	Yao	9min	Directory	Mistakes the unblock for a repeated GetS because of typo in that situation