Electronics Conference (BEC), 2010 12th Biennial Baltic
Digital Object Identifier: 10.1109/BEC.2010.5630292
Publication Year: 2010 , Page(s): 181 - 184

2010 12th Biennial Baltic Electronics Conference (BEC2010)
Tallinn, Estonia, October 4-6, 2010

Graphical Interface for Debugging RTL Networks-on-Chip

L. Möller[1], H. Jesus[1], F. Moraes[2], L. S. Indrusiak[3], T. Hollstein[1], M. Glesner[1]

[1]*Institute of Microelectronic Systems, TU Darmstadt, Merckstr. 25, 64283 Darmstadt, Germany*
[2]*Faculty of Informatics, PUCRS, Av. Ipiranga 6681 P.32, 90619-900 Porto Alegre, Brazil*
[3]*Department of Computer Science, University of York, York YO10 5DD, UK*
E-mail: moller@mes.tu-darmstadt.de

ABSTRACT: One problem of Multiprocessor Systems-on-Chip (MPSoCs) based on Networks-on-Chip (NoCs) is tracing the dozens of parallel communications that are transferred in the system. The goals of tracing communications are usually either debugging or monitoring the NoC for design space exploration. On Register Transfer Level (RTL) NoCs the tracing is frequently verified by waveforms, which provides limited useful information about the global status of the NoC. The goal of this work is to improve the tracing capabilities of RTL NoCs and provide a global picture of what is happening in the NoC. This is accomplished by using a Java tool to represent graphically relevant events of the NoC. The input of this tool is a list of relevant events generated by the RTL simulator during the simulation of an MPSoC. The HERMES NoC is used as test case for the tool.

1. Introduction

Tracing parallel flows of communication is one critical challenging task of future hundred-cores Multiprocessor Systems-on-Chip (MPSoCs) based on Networks-on-Chip (NoCs). The objectives of tracing communications are usually either debugging the NoC or monitoring the NoC for design space exploration. On Register Transfer Level (RTL) NoCs the tracing is frequently verified by waveforms, which are extremely complex due to the size of the NoC and provide limited useful information about the global status of the NoC. Additionally, NoCs demand the analysis of a number of different signals from different routers and at different moments for tracing communication flows.

The goal of this work is to improve the tracing capabilities of RTL NoCs and provide a global picture of what is happening in the NoC. This is accomplished by using a Java tool to represent graphically relevant events of the NoC. This tool is called HermesDebugger and requires as input a list of relevant events acquired from NoC router signals during the RTL simulation of the MPSoC. HermesDebugger interprets this list of events and displays meaningful information in seven different scopes. These scopes provide status information of each port of each router of the NoC concerning input traffic, output traffic, hot spot, power consumption, buffer occupation, end to end and point to point connections.

This paper is divided as follows. Section 2 presents the related works. The simulation of the NoC most important signals and information on how to export these signals are presented in Section 3. Section 4 presents how the HermesDebugger interprets this list of events and displays meaningful information in seven different scopes. Section 5 presents the main parameters for controlling the visualization of NoCs with the HermesDebugger. Finally, conclusions are drawn in Section 6.

2. Related Works

A graphical interface for debugging the actor oriented RENATO NoC was presented at [1]. This graphical interface is called NoCScope and provides seven different scopes:

- EndToEndScope: presents which IP core is communicating with which other IP core without considering the path of communication.
- PointToPointScope: presents which input port is connected to which output port in each router used in the communication. This scope considers the path of communication.
- HotspotScope: emphasizes and quantifies the output ports of the NoC routers that are trying to send packets ahead to a neighbour router, but are being blocked due to congestion.
- PowerScope: presents the power consumption of the routers by analyzing the transition activity on the channels.
- BufferScope: measures the buffer occupation of the NoC routers.
- InputScope: captures the activity of each input port of the NoC routers.
- OutputScope: captures the activity of each output port of the NoC routers.

The NoCScope presented on [1] is reused in this work, having exactly the same graphical interface, which is presented in Figure 1. The difference is that in this work the NoCScope presents debugging information of a RTL NoC called HERMES [2], while [1] presents debugging information for an actor oriented NoC model implemented in Java.

978-1-4244-7359-5/10/$26.00 ©2010 IEEE

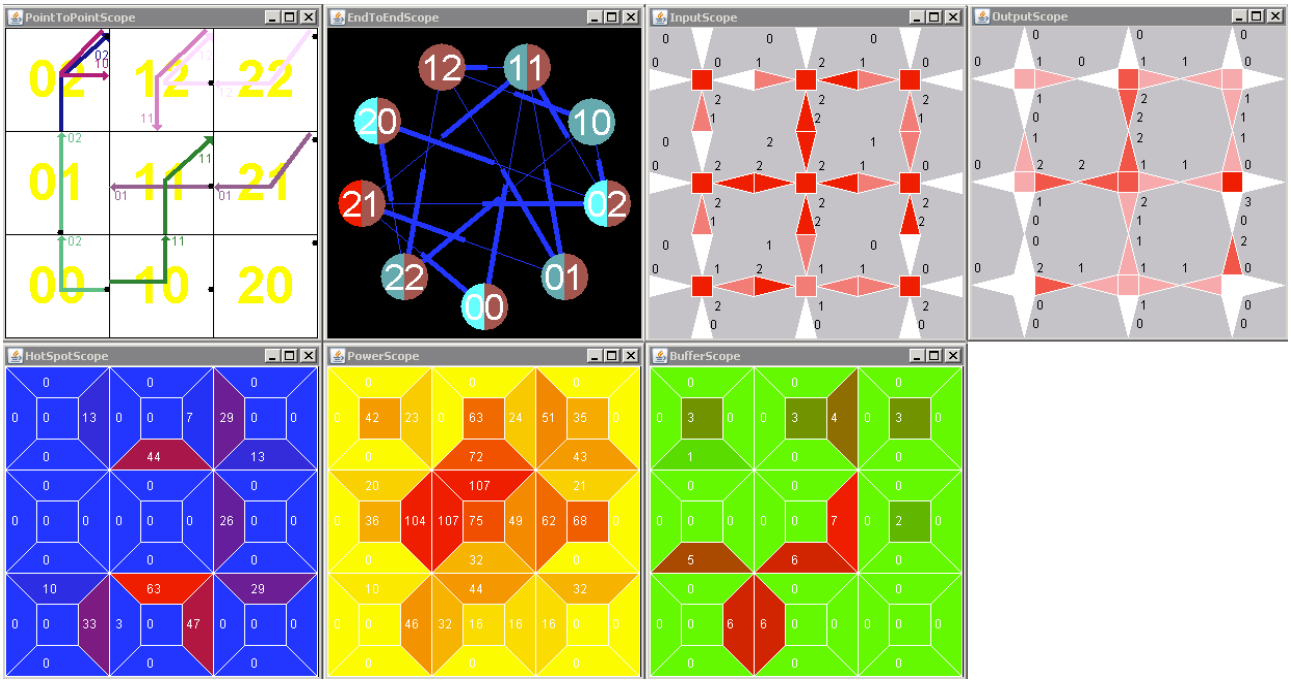

Figure 1 – Screenshot of the scopes used by the HermesDebugger.

3. Simulation

The most desirable solution would be to visualize the NoCScope information at the same time the HERMES NoC is being simulated, however no method was found on ModelSim simulator to send data directly to a Java code. ModelSim allows exporting values of certain chosen signals during the simulation in three different ways. The first one is to use the native VHDL log library to force the simulator to write the values of the signals in a log file every time some process is activated. However this solution required the modification of source NoC files. Second alternative is to write TCL codes that get the value of the signals during the simulation and export them. This solution was not used in favor of a third solution, that besides simpler, required few alterations in the project. This third solution is the use of the ModelSim log feature, which makes all results from a simulation accessible by ModelSim commands.

There are four ways of using the log information: through log files (*wlf* or *vcd* files), waveforms (*waves*) or lists (*lst* files). The *wlf* and *wave* files are better accessed using the ModelSim graphical tool. Only *vcd* and *lst* files can be easily read by text analysis.

The first choice was naturally the *vcd* (Value Change Dump) file. This is an ASCII-based format for dumpfiles that defines names for each signal and, next, prints the changes in each signal in a time-ordered sequence. The advantages of this format are an output file with a great readability and low file size. However, because of a restriction in the ModelSim simulator, this file could only deal with *bit*, *bit_vector*, *std_logic* or *std_logic_vector* signals. Other types of signals like memory were silently ignored.

The next option is the list file (*lst* file), which consists of a file with the name and values of all chosen signals to be printed at all timestamps of the simulation. The problem lies in the size of the output file, which for long simulation periods could reach hundreds of megabytes. This would demand a high volume of free memory to be read by a Java program.

Finally, the chosen solution was the use of an event list file. The structure is the same of a list file, but signals that do not suffer any modification in a certain timestamp are omitted. In fact, the event list file resembles the *vcd* file, but it is not so clean and organized. On the other hand, it works with all types of signals. An example of this file is presented in Figure 2.

```
@0 +3
/TOPNOC/NOC/ROUTER0001/DATA_IN
{0000000000000000 0000000000000000
0000000000000000 0000000000000000
0000000000000000}
/TOPNOC/NOC/ROUTER0001/RX 00000
/TOPNOC/NOC/ROUTER0001/SWITCHCONTROL/FREE 11111
/TOPNOC/NOC/ROUTER0001/CREDIT_O 11001
@20 +3
/TOPNOC/NOC/ROUTER0001/DATA_IN
{0000000000010000 0000000000000000
0000000000000000 0000000000000000
0000000000000000}
/TOPNOC/NOC/ROUTER0001/RX 10000
```

Figure 2 – Event list file example.

Lines starting with the symbol @ present the current simulation time and after the symbol + a delta time is presented. ModelSim uses delta times when many events

need to be processed in the same simulation time, which may happen when some signal is sensitive to another signal that is changed in the current simulation time. All lines that follow a line which starts with symbol @ represent the complete name of a signal and its new value received in the current simulation and delta time.

The important VHDL signals required from the HERMES NoC to be exported by ModelSim and interpreted by HermesDebugger are the following:

- *buf*: temporarily stores all flits that have not been forwarded by the router yet
- *first*: pointer that indicates the position of the first flit stored in the buffer
- *last*: pointer that indicates the position of the last flit stored in the buffer
- *rx*: signal that indicates there is a flit in the input port waiting to be stored in the buffer
- *tx*: indicates a flit in the output port waiting to be sent
- *credit_o*: only for credit based HERMES NoC, this signal indicates if the input buffer associated to this port still has space left to store an incoming flit
- *ack_rx*: only for handshake HERMES NoC, this signal indicates if the receiver accepted the flit received by this port
- *data_in*: flit received by the input port of a router
- *free*: indicates if an output port is occupied
- *mux_in*: indicates the output port connected to a specified input port
- *sel*: indicates the input port selected to send a packet
- *targetx*: stores the X coordinate of the target router from the received packet
- *targety*: stores the Y coordinate of the target router from the received packet
- *localx*: stores the X coordinate of the current router
- *localy*: stores the Y coordinate of the current router

4. HermesDebugger Scopes

The main task of HermesDebugger is to interpret the changes of the NoC signals through time and update every scope of the NoCScope accordingly. The NoCScope is presented in Figure 1 and the next subsections present detailed information of which NoC signals are important for each scope of the NoCScope.

4.1. BufferScope

The BufferScope measures the number of occupied positions in each port of each router. This scope demands a very low computation power, since each buffer is controlled by two pointers: *last* and *first*. *Last* points to the last occupied position of the buffer, while *first* points to the first occupied position. Thus, in order to find the number of flits stored in the buffer, one simple subtraction is enough (*buffer_size = last – first*).

Since the buffer used in the HERMES NoC is a circular buffer, when *last* reaches the size of the buffer, the next position will be again "zero". This is employed to simplify the hardware of the buffer. Therefore, if the result of the subtraction is negative, the value of buffer size must be corrected by adding the buffer depth to it.

4.2. InputScope

The InputScope counts the number of flits received by each port of each router. In order to know if a packet arrived in a specific input port, the program just needs to analyze the *rx* signal. If the bit associated to the chosen port is asserted, then the counter of this port should be incremented. However, the *rx* signal can remain set by many cycles for the same flit. Thus, only positive edge transitions are considered in this scope.

4.3. OutputScope

The OutputScope counts the number of flits sent by each port of each router. In order to know if a packet left a specific output port, the program just needs to analyze the *tx* signal. If the bit associated to the chosen port is asserted, then the counter of this port should be incremented. However, the *tx* signal can remain set by many cycles for the same flit. Thus, only positive edge transitions are considered in this scope.

4.4. EndToEndScope

The EndToEndScope only requires information about the source and the destination routers of a packet. Then, this scope is based on detecting whenever a packet enters or exits the network, and storing the information contained in its header.

When a packet enters the network, the local port of the router receives the first flit of packet, stores it in the input buffer and requests arbitration and routing. When the *ack_h* signal of the local port is activated, the packet has successfully entered the network and the *localx*, *localy*, *targetx* and *targety* signals are stored in a table as a new packet currently inside the network. When the packet arrives to its destination (*localx=targetx* and *localy=targety*) the connection entry is removed of the table.

4.5. PointToPointScope

Among all scopes, the PointToPoint is the one that demands more computation power, since the HermesDebugger has to store all information about all active router connections in a table. Every router and port passed by a packet is stored in this table, so the HermesDebugger can paint the entire path of the packet in the scope. When a packet is passing by a local port of a router, as explained in Section 4.4, this router is either the input or output access point of a packet in the network. If none of the previous cases are true and a packet is forwarded by a router, it means the current router is an intermediate hop. In all cases, after routing is done (*ack_h* is activated) and an input port (informed by signal *sel*) is connected to an output port (indicated by using *sel* as an index of the

mux_in array), this information is stored in a table and used to update the PointToPointScope.

4.6. HotSpotScope

The HotSpotScope quantifies the output ports of the NoC routers that are trying to send packets ahead to a neighbour router, but are being blocked due to congestion. The calculation is based on a simple analysis of two signals. If the signal *rx* of a port is set, while the acknowledgment signal (*ack_rx* for handshake and *credit_o* for credit based) is not, that means some port tried to communicate with this one, but the connection was denied due to congestion problems. Thus, the hotspot counter should be incremented.

4.7. PowerScope

The PowerScope estimates the power consumption of each port by comparing the current flit with the previous flit and, then, counting the number of transitions occurred in each bit. Therefore, the implementation consists only in storing the two most recent flits received by the *data_in* signal, applying the "exclusive or" operation between both values, and counting the number of bits '1' in the resulting vector.

5. HermesDebugger

The main control window of HermesDebugger, presented in Figure 3, allows the user to choose basic parameters, such as location of the list file, start time, step time, window size and speed of the scope execution. Besides, the user can run, pause and resume the program through the visual interface of HermesDebugger.

The "start time" parameter sets the simulation time on which the HermesDebugger should initiate presenting visual status of the NoC. This allows for example to jump the operating system booting process and debugging only communications related to the application under test. The "step" parameter defines the interval between two moments that have to be displayed by the scopes. This allows the user to configure if all events will be displayed by the scope or just an overview of the behaviour of a simulation will be presented when the simulation is too long. The "window size" parameter sets the number of simulation cycles on which HermesDebugger should accumulate information by a given scope before considering the information old and deleting it. This feature is useful when the real time mode is too fast and a simple accumulation of data is meaningless for debugging purposes. The "window size" parameter affects only the scopes that work in sliding window mode as explained in [1], which are the EndToEndScope, HotSpotScope, InputScope, OutputScope and PowerScope. The "window size" parameter is measured in simulation cycles and if this parameter is set to zero it works in real time mode. The "speed" parameter sets the amount of time that the scopes should stay frozen for visualization purposes before updating the next event on the scope.

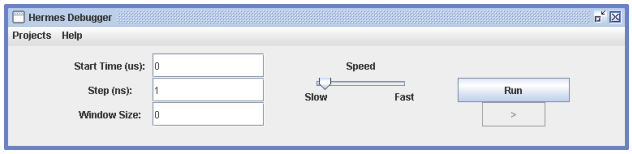

Figure 3 – Screenshot of HermesDebugger.

For example, if a program execution is initiated with parameters "step time" equals to 100 ns and "speed" equals to 100 ms, that means, the scopes will first display the results for simulation time equals to 0 ns and keep this result in the screen for at least 100 ms; then, the monitors will display the results for simulation time equals to 100 ns and keep this result in the screen for at least another 100 ms; then, again the time is increased by 100 ns and so on. This kind of control is required for not loosing the concept of time during the visual debugging of the NoC.

6. Conclusions

This work presented a debugging tool for RTL NoCs called HermesDebugger. HermesDebugger makes use of a list of events exported from an MPSoC simulated by ModelSim as an input and provides visual scopes to supply a global picture of what is happening in the NoC. Congestion problems, low buffer capacity or even high power consumption are issues that can be identified by HermesDebugger without being necessary to analyze waveforms.

Even though HermesDebugger was specially developed for the HERMES NoC, the tool can be extended to other NoCs that are simulated by ModelSim and follow similar regular mesh topologies. The main step to extend the HermesDebugger is to modify the signal names used by the NoC as presented in Section 3.

Future work will deal with the implementation of new scopes and gathering the information directly from the NoC executing on a FPGA board to be presented by the scopes.

References

[1] L. Möller, L. S. Indrusiak, and M. Glesner, "NoC-Scope: A Graphical Interface to Improve Networks-on-Chip Monitoring and Design Space Exploration". In 4th International Design and Test Workshop (IDT'09, Riyadh, Saudi Arabia, Nov. 2009), pp. 1-6.

[2] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip". Integration, the VLSI Journal, vol. 38, No. 1, pp. 69-93, 2004.