NoCem User Guide and Release Documentation

Graham Schelle and Dirk Grunwald

April 5, 2007

1 Introduction

NoCem is an integrated emulation environment for Network on a Chip research. Network on Chips are used for processing elements on a single die to communate over a packet switched network. This is in contrast to the standard bus protocols used today that have the same scaling issues seen with any bus architecture.

NoCem is implemented as an open source NoC emulator that allows a variety of configurations in terms of network configurations, buffering schemes, and arbitration logic. The complete enumeration of parameters will be described in this document. NoCem is targeted for Xilinx FPGA platforms and written in VHDL heavily using generics and generate statements to create full NoC designs.

This project is protected under a GNU General Public License. More information about this license can be found at: http://www.gnu.org/copyleft/gpl.html.

2 Release Directory Structure

This release contains the source code for NoCem. The directory structure is as follows:

- **./VHDL** The source code for NoCem. This includes all the vhdl files needed for generating NoCem network on chips.
- **./doc** This document and an accompanying publication from WARFP2006 where NoCem was first presented are in this folder. The GNU GPL is also included in this folder.
- **./sim** This folder contains a testbench wrapper around NoCem that can be used to test the system. The testbench will work as is if you wish to initially test the nocem source code.
- **./bridge** The processor bridge is included in this folder. This VHDL module can be used to connect the NoC up to a Xilinx OPB or PLB bus which in turn would be used to connect with either the Microblaze or PowerPC processors available on Xilinx FPGAs. We do not include the actual IPIF module, due to licensing constraints, but it is fairly intuitive from the interface to generate that component using Xilinx EDK tools.

3 Configuration Options

NoCem is configurable on a variety of parameters that are modified in a single package file *nocem_pkg.vhd*. These configuration options are shown in table 1 (no processor attached to NoC) and table 2 (with processors

Table 1: Configuration Parameters and Possible Values (with no microprocessor attached).

Parameter	Valid Values
Microprocessor Datasize (bytes)	N/A
NoC dataword size	1-256 bits
Packet Control word size	1-256 bits
Packet Length	2, 4, 8, 16 datawords
Virtual Channels	Y/N
Number VCs	2, 4
Channel FIFO length	2, 4, 8, 16
Topology	Mesh, torus, double torus
QoS	N
Arbitration	RoundRobin (not timesliced)
Packet Length	2, 4, 8, 16
Grid Configuration	Rectangle or Square
Peripheral Bus	N/A
Processor Type	N/A

Table 2: Configuration Parameters and Possible Values (with microprocessor attached).

Parameter	Valid Values
Microprocessor Datasize (Bytes)	4
NoC dataword size	1, 2, 4 bytes
Packet Control word size	1-256 bits
Packet Length	2, 4, 8, 16 datawords
Virtual Channels	Y/N
Number VCs	2, 4
Channel FIFO length	2, 4, 8, 16
Topology	Mesh, torus, double torus
QoS	N
Arbitration	RoundRobin (not timesliced)
Packet Length	2, 4, 8, 16
Grid Configuration	Rectangle or Square
Peripheral Bus	OPB, PLB
Processor Type	Microblaze, PPC

attached to NoC). Setting these parameters is all that is necessary to change the underlying NoC. No hand modifications to the actual code is necessary.

3.1 Qualifying NoCem Constraints

Many constraints are in place due to how large a NoC can grow (i.e. how much logic is required to implement them) on a FPGA. Large number of virtual channels can cause the design to explode in size, creating designs that do not fit on modern FPGAs. Some of these results are reported in [SG06]. Packet lengths are typically restricted for the same reason. Each node has 5 outgoing channels and increasing FIFO sizes again causes the design to become too large. LUT-based FIFOs are used (to save BRAM for other purposes) and use a large deal of logic for larger FIFO lengths. Other constraints unrelated to design size are discussed below:

Processing Element Datasize. For bridging to a processor, both the powerPC and Microblaze processors only accept 4B datawords. Therefore, that datawidth is set. With only dedicated logic connection to the

Table 3: NoCem Ports.

Port	Direction	Description
arb_req	in	request to write to the NoC
arb_cntrl_in	in	communicates virtual channel information, unused in non-VC designs
arb_grant	out	granting access to the access point, analogous to a Read Enable
arb_cntrl_out	out	communicates virtual channel information, unused in non-VC designs
datain	in	data coming in from access points
datain_valid	in	data coming in from access points is valid
datain_recvd	out	data coming in is accepted by NoC
dataout	out	data going to access points
dataout_valid	out	data going to access points is valid
dataout_recvd	in	data going to access points is accepted by access point
pkt_cntrl_in	in	pkt_cntrl coming in from access points
pkt_cntrl_in_valid	in	pkt_cntrl coming in from access points is valid
pkt_cntrl_in_recvd	out	pkt_cntrl coming in is accepted by NoC
pkt_cntrl_out	out	pkt_cntrl going to access points
pkt_cntrl_out_valid	out	pkt_cntrl going to access points is valid
pkt_cntrl_out_recvd	in	pkt_cntrl going to access points is accepted by access point
clk	in	system clock
rst	in	system reset

NoC, there is no restriction on datawidth.

Topology. NoCem only supports the traditional topology schemes. No 3D torus or hexagonal configurations were considered.

Varied Arbitration and QoS. These aspects of a NoC were left out, yet can be added within switching nodes.

4 NoCem Interface

NoCem masks away a great deal of the underlying protocols and networking that occur. This section describes the ports and signaling necessary to interface with NoCem. Table 3 shows the port listing for NoCem

The ports mask away a FIFO interface that connects to the channel FIFO. The arb_cntrl_in and arb_cntrl_out signals are used to mux data in and out of the various virtual channels. The virtual channels also communicate to the access point when they have become deallocated (i.e. the packet has wholly been read out of the virtual channel.

4.1 Packet Format

Currently, the only needed metadata to send with packets are a SOP (start of packet), EOP (end of packet), and the destination address. These fields are listed out in *pkg_nocem.vhd*. Other metadata can be added and

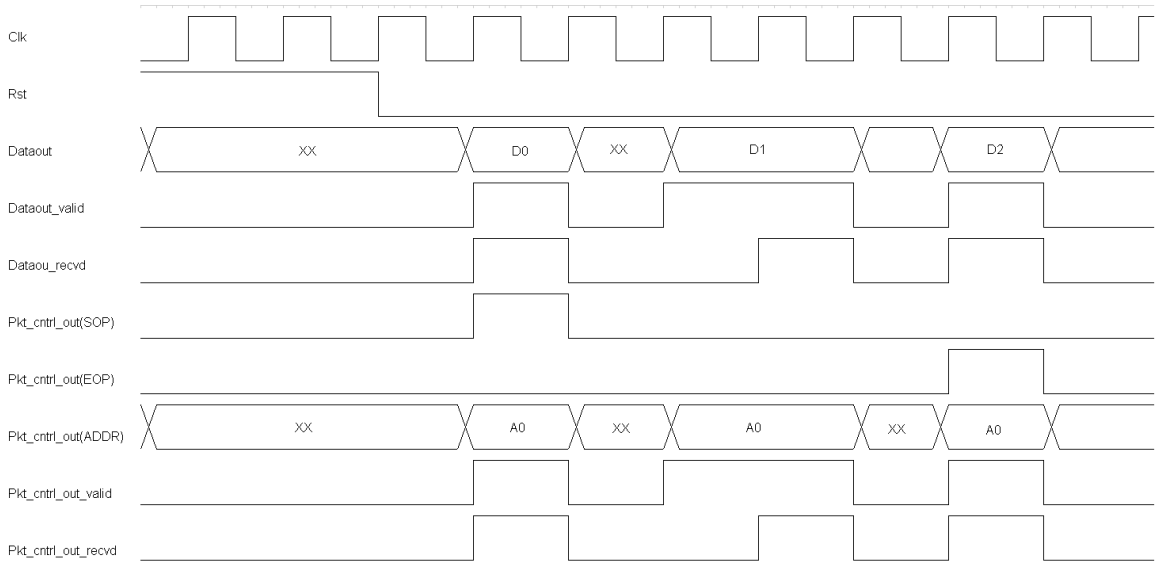

Figure 1: NoCem Timing Diagram – an Access Point Sending a Packet to the NoC

sent with packets, but the 3 named fields above are needed within NoCem to route packets.

Specifically, other metadata can be sent modifying the pkt_cntrl dataword structure in pkg_nocem.vhd. The additional metadata will not be touched by the underlying NoC architecture or control.

4.2 Writing a Packet to NoCem

The access point is required to keep track of virtual channel state, which is communicated by the arb_cntrl_* signals. The channel FIFOs connected to access points can each hold an entire packet per virtual channel. With that information, writing a packet to NoCem only requires a free virtual channel and the data to write. Figure 1 shows a timing diagram for sending a packet to NoCem.

4.3 Reading a Packet From NoCem

The access point must pull packets from NoCem for processing. This is done using the datain_recvd and pkt_cntrl_recvd signals to take data from the NoC. Keeping track of the virtual channel state for packets traveling to the access points can be done using the arb_cntrl signals. Figure 2 shows a timing diagram for reading a packet from NoCem.

5 NoCem Architecture

NoCem consists of building blocks that linked together create a fully functioning Network Switched Architecture. At the base level, a NoC consists of switching nodes and channels that connect those nodes.

The channels have a common interface into the switching nodes and can easily be connected (see *ic_pkt_nocem.vhd*). There is some complexity in light of virtual channels, but this is only apparent on the interface to NoCem external ports.

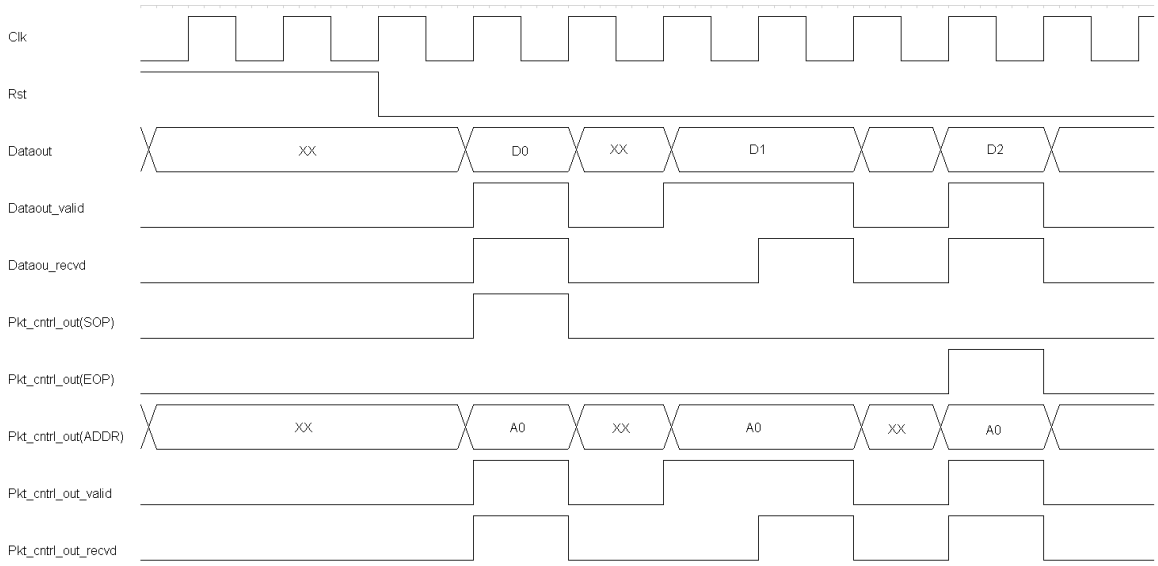

Figure 2: NoCem Timing Diagram – an Access Point Receiving a Packet from the NoC

The nodes themselves consist of a virtual channel allocator *vc_node_vc_allocator* and a switch allocator *vc_node_ch_arbiter*. In a NoC with no virtual channels, a switch allocator is the only component in a node.

5.1 The Channel

A channel is truly a series of FIFOs going to and from the nodes. However, when virtual channels are introduced, a controller is needed for each virtual channel. In a simple single FIFO physical channel implementation, the nodes themselves can manage and snoop on packet movement and make routing decisions. With virtual channels, it is traditional to make the collection of virtual channels look as if it is one physical channel. Therefore, each virtual channel has a controller that does virtual channel and switch allocation tasks.

5.2 Channel Control

In order to connect these components easily, common interfaces are created that can connect the channel to a node. This interface is called the channel_cntrl path and is a sideband path to the normal datapath. The channel_cntrl path communicates a variety of information to a node such as virtual channel requests and switch allocation requests.

With a 4-virtual channel per physical channel implementation, a 52b signaling line is needed per physical channel. While the typical read and write enable signaling is needed, virtual channel allocation and switch allocation signals are also needed. For both cases, the addition of pipelined switching requires more information.

5.3 Pipelining In a NoCem Node

Clearly, without pipelining, there exists a large amount of muxing and demuxing to communicate information in an all-to-all fashion. The switch itself without virtual channels is truly a 5-to-5 switch. Adding 4 virtual channels on a per channel basis, the switch grows to a 20-to-20 switch. In initial prototypes of this size NoC, a Virtex-II architecture could only support a 6 MHz clockspeed. This speed is unacceptable to communicating with OPB and PLB buses that can run up to 100 MHz. Therefore, using minimal pipelining within the switch and the channels, the sustainable clockspeed reaches upto 100 MHz, depending on the number of nodes and virtual channels in the NoC.

Pipelining does add complexities to switching decisions. A decision made 2-3 clock cycles before the decision is actually executed may not be correct at execution time. For example, on clock cycle 1, a decision is made to read data from a virtual channel coming in from the east physical channel. On clock cycle 2, that same virtual channel is read from, emptying out the FIFO. on clock cycle 3, the switch attempts to read data from an empty FIFO, causing no data to be written into an egress virtual channel. With this example in mind, it is clear that decisions made earlier in time need to be re-examined at execution time.

We accomplish this in NoCem by canceling decisions at execution time based on the CURRENT state of virtual channels. Another option would be to keep a queue of decisions and cancel them dynamically as conflicts arise. However, to do so is a rather complex process and changes depending on the pipeline depth and requires state from the virtual channels (i.e. their data counts for this and successive clock periods). Pipelining is necessary to increase sustainable clockspeed and a decision making policy was decided that was simple, yet supported any depth of pipelining.

6 Processor Bus Bridges

NoCem can bridge to either the PLB or OPB buses. These buses are attached to either PowerPC or Microblaze processors found on Xilinx FPGAs. The technical details of the bridges are discussed here. The same bridge is used for either bus specification, but uses a different IPIF connection to interface with the difference buses. This is done by pulling out all the needed IPIF signals into the bridge and letting the bridge itself manage the signaling. It is highly recommended that the user look through noc2proc_bridge2.vhd. The code is heavily commented.

The bridge noc2proc_<plb,opb> are the IPIF blocks that simply pull out all the Bus2IP_* and IP2Bus_* signals. These signals are then attached to a single noc2proc_bridge (*noc2proc_bridge2*). This bridge contains an ingress and egress packet buffer and communicates directly to the NoC.

6.1 Processor –> NoC flow

The processor writes to a set address offset with packet length (in bytes) and destination specified (0x0). Then the processor writes the data to another address offset (0x4). The noc2proc_bridge is able to take the data and inject it into the packet buffer. Once the packet_buffer has a full packet, it is written out to the NoC.

6.2 NoC –> Processor flow

The noc2proc_bridge pulls a packet from the NoC and stores it in a packet buffer. The Processor will continually poll the noc2proc_bridge for new packets at offset (0x80). When a packet is ready, the processor

will receive a packet length (non-zero value), that tells the processor that a packet is ready. Then the processor can at any point read the packet out from offset (0x84).

6.3 Polling versus Interrupt Based

The current version of the bridge is polled in order to push packets TO the processor. The ability to interrupt the processor is available and is currently commented out in *noc2proc_bridge2.vhd*. We found the polling version to be more efficient and allow more flexibility in EDK designs. More specifically, using the PowerPCs exception table and initialization code typically requires offchip memory.

7 How to Begin Using NoCem

This section details how to best start using NoCem.

- **Simulate NoCem.** Using the testbench.vhd file found in the ./sim directory, a Modelsim simulation can be ran showing the functionality of NoCem.
- **Modify NoCem.** In *pkg_nocem.vhd*, there are multiple parameters that can be changed and modified to change the behavior of NoCem. This is a good place to start experimenting with NoCem's reconfigurability. The naming system should be self explanatory.
- **NoC to Processor Integration.** We can not ship Xilinx IP with NoCem. There is only one block in NoCem that requires the standard Xilinx Coregen licenses that come with Xilinx ISE, but can be avoided using IP we created (less efficient implementaiton however).

  Using Coregen, the user can then create a 32b wide FIFO that must have First-Word-Fallthrough enabled. In the design, a *fifo32.vhd* file is absent. I have fully tested the design with a BRAM implementation of that FIFO. If this is not possible, fifo_gfs.vhd is a LUTRAM-based FWFT FIFO implementation that I created.

  Using the bridges as EDK peripherals, NoCem can be connected to an EDK project. There is no best way to do this, but we use an EDK project and export it to ISE, where the *noc2proc_bridge2.vhd* is connected to the processor and NoCem itself.

References

[SG06] Graham Schelle and Dirk Grunwald. Onchip interconnect exploration for multicore processors utilizing fpgas. In *2nd Workshop on Architecture Research using FPGA Platforms*, 2006.