Preface

This master's thesis was conducted at the *Computer Science and Engineering* division of Informatics and Mathematical Modelling Department at the Technical University of Denmark from March to September 2004. The thesis was a part of the Systems-On-Chip design work at this department. It is defined with the help of Jens Sparsø, who acted as my supervisor, Tobias Bjerregaard, and Shankar Mahadevan, who are PhD students working on related issue of the System-On-Chip design.

I would like to thank Tobias Bjerregaard and Shankar Mahadevan for providing me with ideas and guidelines for this project. I would also like to thank Jens Sparsø for his support and instructions. Finally, I would like to thank Martin Hans and Jacob Gregers Hansen, who shared the same office with me and had made my final few months at DTU a very memorable experience.

Juliana Pei Zhou, Copenhagen Denmark

September 17, 2004

Abstract

This thesis is part of a greater effort at The Technical University of Denmark to investigate on-chip networks. In particular, it describes the design and implementation of a network adapter which by the means of an OCP interface connects an IP core to the network. The features of the network adapter include providing the IP core with differentiated services in the form of guaranteed and best-effort services and translating between the message based format at the OCP interface and the packet based format at the network interface.

KEYWORDS: SoC, NoC, OCP compliant, network interface, network adapter, guaranteed service, best-effort service, On-Chip Network.

Resumé

Dette eksamensprojekt er del af den forskning på Danmarks Tekniske Universitet der har til formål at undersøge *on-chip networks*. Projektet omhandler design og implementering af et netværksinterface som forbinder en IP-core til netværket. Interfacet understøtter differentierede services i form af *guaranteed services* og *best-effort services* og oversætter mellem OCP-interfacets *message*-baserede og netværkssidens pakkebaserede format.

KEYWORDS: SoC, NoC, OCP-konform, netværksinterface, guaranteed service, best-effort service, On-Chip Network.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

The focus of this chapter is to describe to the reader the motivation for this thesis, the scope of the thesis work, and how the thesis is organized. To do this, the reader will first gain some background understanding of System-On-Chip (SoC), their usages and benefits, and some of the current challenges faced by SoC designers in terms of on-chip communication and interconnect. Network-On-Chip (NoC) will be introduced as a possible solution to SoC communication problems, which will then lead us to the discussion for the motivation of this thesis project.

1.1 System-On-Chip and its applications

In today's dynamic world of computer technology, we are witnessing the convergence of multiple traditionally unrelated applications such as computation, communication (videophone, networking), and multimedia (audio, video, photography) in an embedded environment. This convergence leads to increased demands on the functionality of embedded devices and expands their heterogeneity. Embedded systems become much more dynamic and unpredictable as new algorithms shift to higher semantic levels. In addition to providing multifaceted functionalities, the users of embedded systems expect a predictable behavior in its performance. For example, a consumer electronic device must deliver correct information within a reasonable amount of time, and should not crash or be unresponsive. This notion of quality of service (QoS) is in place to ensure that developers of electronic devices will aim to offer a predictable system behavior to the users while meeting the users' functional needs. The important question now is : How can designers create such an embedded system that will incorporate the dynamic functionalities of many unrelated tasks of unpredictable nature while guaranteeing a certain acceptable level of services to the users. System-On-Chip (SoC) designers attempt to answer this question in the most cost effective manner.

According to Moore's Law, the number of transistors that can be integrated in an IC will grow exponentially over time. It predicts that chips in 2010 will count over 4 billion transistors, operating in the multi-GHz range [1]. It is this unprecedented computational power that gives rise to the possibility of having System-On-Chip in which multiple resources (cores) such as processing cores, storage devices, FPGAs and other kinds of Intellectual Property (IP) cores, reside on one silicon chip in an embedded environment. These multiple resources will then interact with each other, working together to fulfill the requirements and needs of the users.

1.2 Design challenges for implementing SoC

Although Moore's Law fuels the convergence of resources on a single chip, in actual fact, it is only a prediction. When it comes to implementing SoC, designers face the following challenges:

- *Synchronization and wire delay* - For SoCs that occupy a large area and requires that wires be stretched over long distances, a increasing ratio of the delay of long wires with respect to the transistor gate delay results. Long wire delay makes global distribution of fast clocks difficult resulting in clock skew which creates timing closure and synchronization problems among the different components.

- *Power delivery and dissipation* - As the number of transistors increase exponentially on a single die, the overall static power increases due to the individual static power dissipation of each transistor. Dynamic power dissipation increase as well because components on the chip are required to perform a lot of computations. In a embedded SoC environment, where power supply is limited, this increase in both static and dynamic power dissipation poses a problem for maintaining battery life and removing heat on a small die.

- *Predictability and performance* - IP cores operating together need to meet an expected level of performance and its behavior must be predictable. The dependence of the propagation delay on the interconnect architecture and chip topology is making predictability of the system increasingly difficult because an unreliable interconnect architecture will not be able to meet the stringent performance requirements such as throughput and latency.

- *Flexibility and time-to-market pressure* - With the requirements of the over functionality of SoC ever changing and unpredictable depending on consumer needs, flexibility in changing custom-made IP cores within a system without having to reprogram ever time an interconnect structure changes can save a lot of design reiteration time and hence cost. Time-to-market pressures forces SoC designers to quickly adapt new IP cores to an SoC system, meaning that IP cores should be able to be reused in a different SoC environment without having to spend too much time to reprogram. This is also referred to as *design reuse*.

It can be shown that the above mentioned challenges are global in nature, are strongly dependant on how the interconnect technology is implemented, and can therefore be classified as interconnect problems. An illustrating example is component synchronization. When meeting timing constraints for IP cores, it is not possible to verify the timing for IP cores independently because they are interconnected. Meeting the timing constraints for one IP core may violate those of another, resulting in a global timing closure problem. The manner in which the interconnect is implemented can greatly affect the SoC's timing performance.

Due to the exponential complexity property of global design methods, there is an increasing need to divide the global problem into localized and decoupled subproblems where solutions are scalable and can be found locally.

In the following section the reader will be introduced to the concept of Network-On-Chip (NoC) and how researchers today propose it as the solution to SoC design challenges discussed in this section.

1.3 Network-On-Chip as a proposed solution for SoC interconnect / communication design

From our discussion in the previous section, we have seen that interconnect technology is a limiting factor for achieving SoC's operational goals. Network-On-Chip (NoC), a hardware architecture for interconnect with a protocol model, has received considerable attention in the recent years as a promising solution to solving SoC interconnect problems for the following reasons.

Firstly, today's dominant interconnect for SoC is the conventional shared-bus architecture. However, because of its inherent non-scalable nature, buses can no longer support the communication demands of complex SoCs where tens or hundreds of IP cores need to be synchronized and communicate in parallel [2]. The bandwidth of a bus is shared by all attached devices and it has serious scalability limitations when the number of IP cores increases. Shared-bus architecture is also energy inefficient because every data transfer is a broadcast, meaning that data must reach every possible receiver wasting lots of unnecessary energy cost. Hence, on-chip packet-switched network is exploited as it is a technology which originates from parallel computing and is well suited for heterogeneous communication among IP cores. NoC is based on the idea of packet-switching and using routers and links as communication framework between components.

Secondly, as introduced in the previous section, complex SoC design requires a modular, component-based approach to both hardware and software design where global problems or quality-of-service requirements such as reliability, performance, and power bounds are subdivided into smaller and more manageable tasks. NoC achieves this by partitioning the communication between IP cores into abstraction layers (protocol stacks) and reconfigurable micronetworks, which utilize the methods and tools for general computer networks [3]. Implementing the protocol stack allows a standardized communication protocol across components and opens possibility for specialization and optimization for the target application domain, therefore achieving efficient communication in SoCs .

Thirdly, in regards to power consumption minimization, network traffic control and monitoring for NoC can help better manage the power that networked computational resources consume. For example, the clock speed and end node voltages can vary according to available network bandwidth [4].

Lastly, NoC will also be able to address time-to-market pressure for SoC design. NoC offers flexible scalability and network reconfigurability, which is absolutely critical when it comes to reuse of already designed components or IP cores. The goal is to decouple computation from communication through a well defined interface between the IP cores and the underlying NoC network infrastructure. This decoupling allows flexible reconfiguration and programming of the SoC in a *plug-and-play* manner without having to go through tedious reprogramming and readjustments every time IP cores are changed or replaced. In addition, decoupling permits IP cores to be bought or premade. Using them and making them to work with other IP cores in a SoC is a simple matter of plugging it into the NoC's standardized interface, also known as the *network adapter* (NA), which will be the focus of this thesis. The NA allows this plug-and-play feature of the NoC that will enable SoC designers to save a significant amount of design overhead time and therefore cost when SoCs are reconfigured with different IP cores [5].

1.3.1 Quality of Service guarantee in NoC

The reduction of cost of system design through reuse of application and architecture is the aim of *platform-based* design. With increasing demands on functionality for SoCs and increasing diversity and dynamics of resource usages, SoC must provide differentiated services to maintain predictability and reliability of system performance. NoC designers use the NoC's protocol stack as a layered approach to offering differentiated services between different client IP cores based on a common network [6]. Guaranteed services in addition to best-effort services is a requirement for NoC hardware architecture. One of the main tasks for the network adapter apart from decoupling computation from communication is to ensure that the NoC will be able to offer a set of guaranteed services on top of which different kinds of communication can be implemented.

1.4 Project Motivation

Currently, the System-On-Chip group at IMM, Technical University of Denmark, is designing a NoC for research purposes. From the introduction to NoC, we have seen that in order to reduce the cost of system design in terms of reducing time-to-market and allowing component reuse in a plug-and-play fashion, a network adapter with a standardized and well-defined interface to the IP cores is required. Furthermore, the network adapter is also responsible for providing differentiated services to the attached IP cores because an expected level of quality-of-service is expected for a well performing SoC design.

The objective of this thesis is to investigate the design of a network adaptor (NA) for the DTU NoC using the Open Core Protocol (OCP) [7] as the standardized communication protocols for the attached IP cores.

1.5 Project Scope

The design of the NA for DTU NoC involves the following design tasks and responsibilities:

- Determining the possible types of NA and how they differ in terms of implementation.
- Design conceptually all types of NA and determine their internal modules and what they each do.
- Defining and specifying two interfaces on the NA: the OCP interface to the IP cores and the network interface to the underlying interconnects.

- Determining all supported forms of communication in the NoC and how the NA will handle the management of these communication types.
- Defining the packet formats for different communication traffics in the network.
- Define how the NA packetizes requests from the initiating core and sends it to the packet-switched routing network.
- Determine how the NA handles congestion problem into the network.
- Define how the NA depacketizes the responses and presents it to the connected IP core while complying to the OCP protocol.
- Define how the NA sets priority for different traffic types and how to schedule them accordingly.
- Determine how different types guaranteed services will be handled in the NoC, including connection setup and teardown.
- Determine how the NA differentiates between different traffic types in the NoC and handle the payload in accordance to its packet type.

After the completion of the above design tasks, a behavioral implementation of the NA is completed in Very high speed integrated circuit Hardware Description Language (VHDL). The design and implementation are verified and tested followed by a performance and cost estimate. All of these will be described in later chapters in this thesis.

1.6 Thesis Organization

In **Chapter 2**, a brief introduction will be given about the DTU NoC. The reader will learn about the different design issues involved, the different components in the DTU NoC, the types of services the NoC provides, the types of communication the NoC handles, and how memory and addressing is defined.

Chapter 3 gives a brief overview to the OCP protocol, a discussion of the DTU OCP specification and how it is related to the full OCP protocol suite.

Chapter 4 discusses the conceptual design of the network adapter. For those readers only interested in gaining a general sense of what the network adapter is about, reading this chapter should be sufficient.

Chapter 5 examines the implementation details of the network adapter including all its components and how they are interfaced together to achieve the overall performance requirements.

Chapter 6 describes the testing of the network adapter in terms of the type of testing performed and how the testing validated the correct operation of the network adapters in accordance to the design defined in chapters 4 and 5.

Chapter 7 discusses the implementation results and gives a performance and cost estimate of the network adapter.

Chapter 8 points out directions for future work for those interested in further developing the network adapter.

Chapter 9 summarizes and concludes the thesis.

Following the chapters will be the appendices. The reader will find a copy of of the DTU OCP spec and the DTU programmer's model appended for convenience because these two documents are critical to the design and the implementation of this network adapter.

Chapter 2

Brief Overview of the DTU Network-On-Chip Architecture

Contents

Before entering into the design details of the DTU network adapter, the reader needs an overall picture of the NoC in which this network adapter is a part of. This chapter first introduces the DTU NoC by the acronym name MANGO, and describes how MANGO addresses its characteristics. Afterwards, the components of the MANGO and their functions will be discussed. Finally, this chapter presents possible types of communication that can take place in MANGO. By the end of this chapter, the reader will have gained a basic understanding of DTU NoC and how the network adapter fits into the overall NoC design.

2.1 The DTU Network-On-Chip and its characteristics

In this section, we will consider five main design aspects of the DTU NoC, also known by the acronym name **MANGO**.

Message Passing	–	By partitioning the communication into abstraction layers and using packet-switching, processing cores operating in parallel can send discrete messages to one another.
Asynchronous	–	The DTU NoC applies the globally asynchronous and locally synchronous (GALS) paradigm.
Network-On-Chip	–	The DTU NoC is realized by an on-chip network with a distributed-shared memory addressing scheme.
Guaranteed sevices	–	The DTU NoC provides differentiated services such as guaranteed service connections and best-effort connections. The configuration of these connection setups and teardowns in the network adapter is performed in a centralized manner via the Network Controller.
OCP interfaces	–	The DTU NoC implementing a core-centric communication interface by using the OCP protocol as the standardized protocol between IP cores and the network adapter.

In the following sections, we will take a more in-depth discussion of MANGO and its design implications.

2.1.1 Message Passing

As introduced in the previous chapter, a NoC is based on the idea of a packet-switched network. Processes on IP cores perform message passing in parallel using a transaction-based protocol. Although packet-switching introduces some overhead costs such as packetization, routing, and buffering, it is essential for IP reuse strategy because it enables compositional and scalable integration of the IPs. Packet-switching is based on the idea of abstraction layers and protocol stacks as each component within the network implements a stack layer and packets are created, routed, and unpacked at various layers across the protocol stack. Protocol stacks are used in networks to implement communication services and are necessary to manage the network complexity and to offer differentiated services. To better understand the communication among IPs in MANGO, we first look at the MANGO abstraction layers and its relation to the OSI reference stack.

The MANGO architecture is designed with three abstraction layers, as

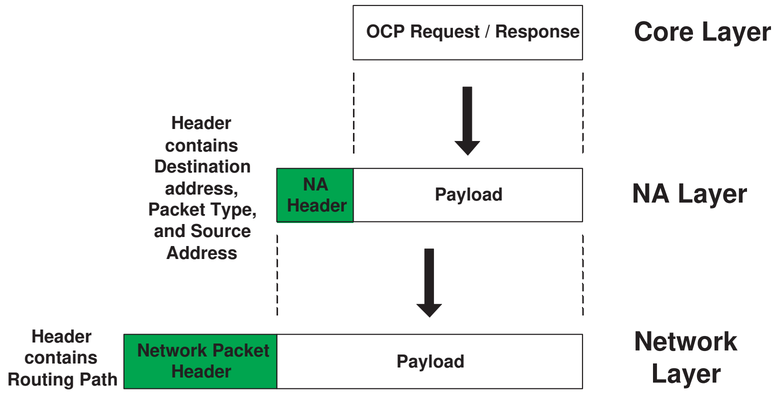

Figure 2.1: Encapsulation of OCP commands at each abstraction layer.

shown in Figure 2.1: the IP Core resides on the top Core Layer, the network adapter resides on the middle NA Layer, and the NoC resides on the bottom Network Layer. A data unit at any particular layer is always encapsulated in the layer below as payload.

Encapsulation allows each layer to fully rely on the services provided by the level below. In Chapter 4, details of the packet contents at each abstraction level will be discussed. Figure 2.2 shows the mapping of the three MANGO abstraction layers to the OSI reference stack model.

Since NoCs are small networks on a single chip, the responsibilities of each abstraction level need not be as complicated as the corresponding reference stacks in the OSI. It is not necessary to implement all of the OSI stack layers to provide high-level functionality. The OSI stack model can be modified to match the needs of the system components. Below we briefly discuss the responsibilities of each of these three MANGO abstraction layers.

Core Layer: This is where the IP core of the NoC resides. Application of the IP cores relies on the services provided by the NA layer below to communicate messages to applications running on other cores.

NA Layer: The network adapter is responsible for covering the tasks at the NA layer. The NA layer establishes and maintains end-to-end connection. It manages flow control, performs packet segmentation and reassembly, controls congestion to the network, carries out scheduling over multiple virtual channel, and hides the topology of the network and the implementation of the links from the applications at the core layer.

Network Layer: The interconnecting links and routers are at the network layer. The network layer is responsible for transferring data over the physical links, it does packet routing, and performs multiplexing and arbitration.

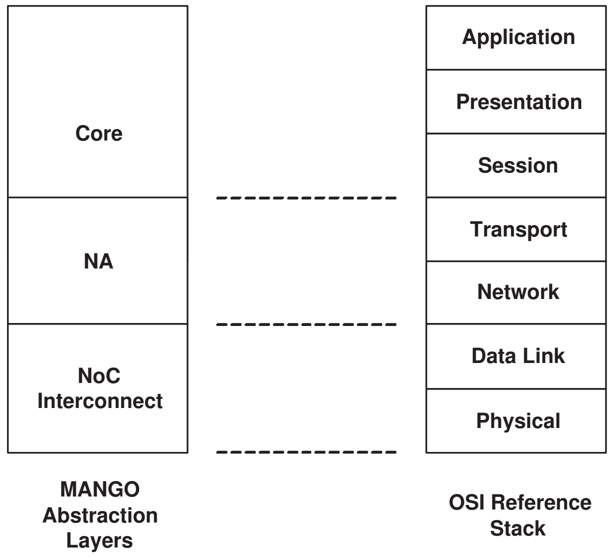

Figure 2.2: Mapping of DTU NoC protocol stack to the OSI protocol reference stack.

2.1.2 Asynchronous

Last chapter, we learned that synchronizing the various components on a SoC in a global sense can lead to difficult timing closure problems as the number of components on chip increases. Distributing the single clock source with negligible skew is extremely difficult. To eliminate the clocking problem and to modularize components in the SoC by partitioning the global problems into local ones, MANGO implements the globally asynchronous and locally synchronous (GALS) paradigm, which involves synchronous IP cores running at different speeds communicating across an asynchronous NoC interconnect. Thus, IP cores will initiate data transfers autonomously according to their needs. The asynchronous interconnect architecture provides the communication infrastructure for the IP cores.

2.1.3 Network-On-Chip with Distributed-Shared Memory

The addressing scheme on MANGO is chosen such that the address space is distributed evenly among all IP cores and the components of the NoC including the network adapter, the routing nodes, and the links. When addressing to a particular address, an IP core sees a continuous addressing space and is not aware of the underlying routing network. This addressing scheme facilitates the decoupling of computation from communication because it enables communication between IP cores to be independent of the network implementation. Any configuration of MANGO components, such as setting GS connection information in the NA and setting routing information in the routing nodes, can be done using direct read and write commands to the addresses which those components are mapped to. For readers interested in the details of memory space mapping for MANGO components, please see [8].

2.1.4 Guaranteed Services - Centralized management of connection setup and teardown

To ensure performance predictability for the SoC, MANGO offers quality-of-service to provide better service to particular flow of data that needs some amount of service guarantee. The guaranteed services can be of several possible types: throughput, latency, and jitter, etc. MANGO provides both the *guaranteed service* (GS) connections as well as the *best-effort* (BE) service. The default service is the BE service with an unspecified finite latency bound. BE packets, which use worm-hole source routing, do not require previous connection setup or connection termination after use. The routing nodes simply forwards the BE packets in the best possible manner given resource availability, and does not guarantee any QoS parameters. Guaranteed Service (GS) packets, on the other hand, can provide guarantees in terms of latency, throughput, and jitter. GS connections require reliable communication and therefore needs connection setups and teardowns, which are created and removed by sending special purpose BE request packets to the Network Controller (NC). The NC manages all connection setups and teardowns in a centralized fashion by establishing a guaranteed path through the network subject to resource availability. Details of how BE and GS communications are implemented in MANGO are further discussed in Chapter 4.

2.1.5 OCP Protocol - Core Centric Communication

As another effort to realize decoupling of computation and communication and allowing high portability of SoC cores, a core-centric communication protocol is used. This means that the cores and the interfaces to the network adapter are equipped with a general purpose interface; therefore, making the network adapter as a bridge between the communication network. The protocol chosen for MANGO is the general and standardized Open Core Protocol (OCP). It is specified in detail in [7]. OCP will also be briefly introduced later in Chapter 3.

2.2 MANGO Architecture

MANGO, depicted in Figure 2.3, is comprised of resources (IP cores) and network infrastructure. The network consists of routers, interconnecting links, network adapters (NA), synchronizer, and a network controller. It has a simple grid topology with routers located at the cross-sections of the network grid. NAs act as interfaces between the IP cores and the routers. Each IP core is connected to the network through a layer of abstraction via the NA. This concept is analogous to a microcomputer that uses a modem/ethernet card to connect to the public network. The physical network underneath is hidden from the microcomputer by the network adapter.

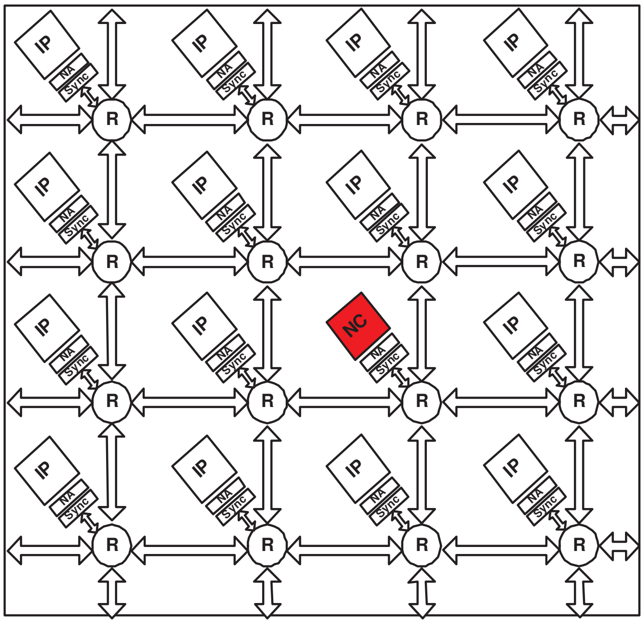

Figure 2.3: A sample 4 X 4 example of MANGO with 16 IP Cores, Network Adapters, Synchronizers and Routers. The NoC comprises of synchronous IP cores and asynchronous interconnects (GALS).

Cores communicate with each other by sending messages via the NA and network underneath. Routers are connected to four neighboring routers via interconnecting links which implement input and output virtual channels. The following sections will briefly describe the function of the components in MANGO shown in 2.3.

2.2.1 Network Controller

The Network Controller (NC) is a special core in MANGO which is responsible for setting up and tearing down guaranteed connections requested by master IP cores. The NC keeps track of all the connections in the network and manages them. The NC allows a centralized control of the connections in the NoC. Having a centralized control can possibly speed up network configuration for small NoCs. However for large networks centralized control can introduce bottleneck [5].

2.2.2 Network Adapter

The NA is designed to provide a standard interface to the IP cores such that an IP can enjoy the plug-and-play feature discussed in chapter one. The OCP protocol allows the NA to standardize such a point-to-point interface between two communicating entities - one acting as a master and the other acting as a slave. Only the master entity is capable of initiating commands. The slave entity is responsible for responding to that command. In order for one IP core to communicate with another IP core in a peer-to-peer fashion, where the network hardware architecture is transparent, there needs to be two instances of OCP interface such that one instance acts as a master on one IP and the other acts as a slave on the other IP. The IP itself is not aware of the underlying communication network. It only understands the global memory address space where it can perform read and write operations on. The NAs, which the IPs are attached to, are responsible for handling the correct packaging of the messages between the communicating IP cores such that the packets are network compatible. The NA should fully comply to the OCP specification described in [7].

2.2.3 Routing Node and Node Controller

The routing nodes are used to route packets in and out of virtual channels in all north, south, east, and west directions. The routing schemes and scheduling performed by the routing nodes are configured and controlled by the Node Controller. The Node Controller controls the nodes by offering a memory mapped view of the setup registers in each node [9]. Guaranteed service connections can be setup and torn down by simply writing to these registers.

2.2.4 Interconnect - Virtual Channels

The interconnect is implemented such that between two adjacent routing nodes, a link implements a number of virtual channels by time-multiplexing data flits from them. Virtual channels are located at each end of links leading into and out of the routing nodes. They share the same physical channel but are logically independent from each other.

2.2.5 Synchronizer

The Synchronizer component (Sync) which resides between the NA and the routing node shown in Figure 2.3 is used to synchronize between the synchronous IP packet to the asynchronous NoC interconnect. At the input and out ports to the NA, there are two virtual channel controllers in the Synchronizer, one for each port. These two controllers are used to provide handshaking mechanism with the virtual channel controllers inside the NA.

See Figure 4.11. Handshaking is needed in order to pass packet segments or flits to and from the NA and the NoC. After receiving a packet from the NA, the Synchronizer must this packet asynchronously to the NoC and vice versa. The synchronizer allows the IP cores to be running at different speeds. It alleviate the problem of having to synchronize the single clock single across all cores in the NoC.

2.3 MANGO Communication Infrastructure

Taking into account guaranteed service connections and best-effort service, there could exist four different possibilities of communication between a master (initiating) IP core and a slave (target) IP core in MANGO.

- Best-effort service in both the forward and reverse directions between a master and a slave.
- Best-effort service in the forward direction, but guaranteed service in the reverse direction.
- Guaranteed service in the forward direction, but best-effort service in the reverse direction.
- Guaranteed service in both forward and reverse directions.

The master core can request any one of the above four types of communication by sending a connection setup request best-effort packet to the NC. The NC sets up the connections accordingly and responds to the master and slave core to indicate whether the connection was set up successfully. When connection is no longer desired, the master core can send a teardown request best-effort packet to the NC, and the NC responds accordingly. For clearer explanation of how connection setup and teardown is done please read Chapter 4, Sections 4.2.3 and 4.2.4.

In this chapter, we have introduced the characteristics and the DTU NoC or MANGO architecture. We have learned about how the components of MANGO are organized and about their functions. Lastly, we have learned the different possible types of communication that can occur between two IP cores.

Chapter 3

The OCP Protocol

Contents

MANGO uses the Open Core Protocol (OCP) as the core-centric protocol that comprehensively describes the system level communication for Intellectual Property (IP) cores. Therefore, the network adapter, which is the focus of this thesis, must support the OCP protocol at its interface to its attached IP cores. However, the OCP protocol is very elaborate and flexible, and MANGO only needs a subset of the OCP protocol suite, the MANGO OCP suite. In this chapter, an overview of the general OCP protocol will be given followed by a highlight explanation of the MANGO OCP suite along with its signal meanings for handling GS service in regards to the network adapter. Understanding this chapter is important in understanding the network adapter core interface.

3.1 OCP Overview

This section gives a very general introduction to the Open Core Protocol^{TM}(OCP). The description is very brief and most of the details of the complete OCP suite are left out. For a complete description of OCP, please refer to the OCP Specification [7].

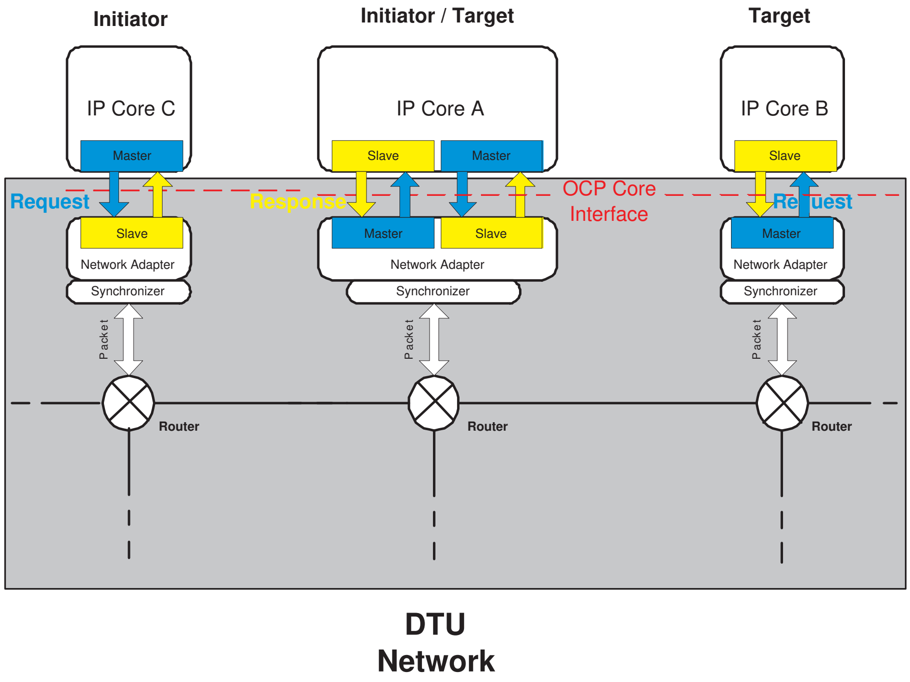

Figure 3.1: System On Chip communication showing Network Adapter and OCP instances

The OCP defines a high-performance, bus-independent interface between IP cores that reduces design time, design risk, and manufacturing cost for SoC designs [7]. Figure 3.1 shows an example of a simple system containing three NAs and three cores communicating with each other using OCP. Figure 3.1 shows that OCP is based on master/slave communication. A core can either be an initiator, containing an instance of an OCP master, or a target, containing an instance of an OCP slave, or both, depending on the core's characteristics and functions. Since the NA acts as a standard interface to the IP cores, it must provide complementary OCP instances to the connected cores as can be seen on Figure 3.1. However, from the core's perspective for example, a master core only sees the slave slave core entity while communicating. The NA's master/slave OCP instances are transparent to the core's master/slave OCP instances. This means that IP interfaces should able to function as if one IP master was directly connected to another IP slave with no communication architecture in between. In practice, the NoC architecture adds some delay to the communication interface. We will look at some performance parameters for the NA such as latency in Chapter 6.

A typical communication interaction between initiator IP core A and target IP core B performs as follows:

1. Core A master presents a command, control, and possibly data (de-

pending on whether the command is read or write) to its complementary slave OCP instance on ched NA.

2. The NA slave converts this OCP request into a NoC packet and sends this packet to the receiving NA master of the target core B. The NA for core A is responsible for preparing the header of the packet which also includes the routing path across the NoC. We opt for source based routing to reduce the complexity of routers in the NoC.

3. The receiving NA master converts the received packet to valid OCP command and presents this command to Core B's OCP slave.

4. Core B's OCP slave receives the command and performs the requested action and possibly returns a response to Core A to acknowledge transaction execution, depending on the type of service Core A requests.

3.1.1 Signals

OCP interface signals fall into three categories: dataflow, sideband, and test signals. The dataflow signals are divided into basic signal, simple extensions, burst extensions, and thread extensions.

The OCP is a synchronous protocol with a single clock signal. All OCP signals are driven and sampled at the rising edge of the OCP clk. OCP signals are point-to-point and unidirectional with exception of the clock signal. Except for a few mandatory signals in the basic signal group, all other OCP signals are optional and can be configured to support additional core communication.

The classification of the OCP signal group is shown in Figure 3.2. We will review briefly the usage of each of these signal groups in this section.

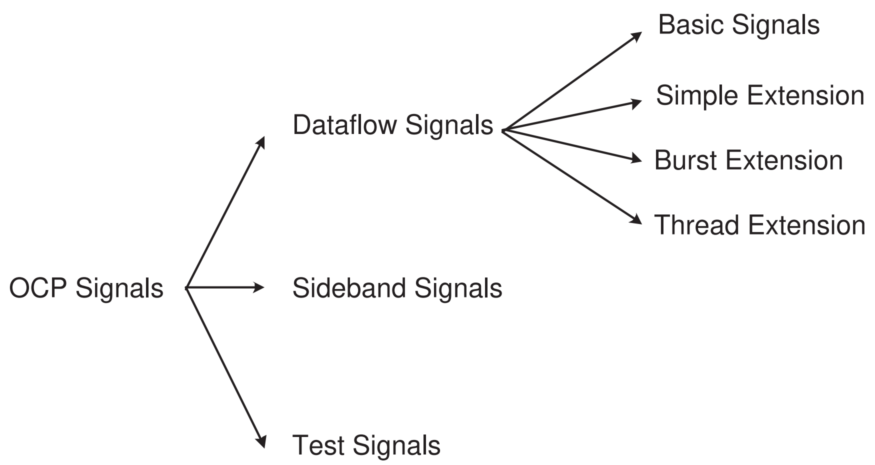

Figure 3.2: OCP signal classification

Dataflow Signals

Basic Signals are the most fundamental set of OCP signals. Table 3.1 lists all the signals in the basic signal set.

Signal Name	Description
Clk	Clock.
MCmd	Command signal from Master to Slave.
MAddr	Address signal from Master to Slave.
MData	Data from Master to Slave.
MDataValid	Indicates that the data on the MData field is valid.
MRespAccept	Ackowledge signal from Master to Slave indicating that the slave response has been accepted.
SCmdAccept	Ackowledge signal from Slave to Master indicating that the master command has been accepted.
SResp	Response signal from Slave to Master
SData	Data from Slave to Master.
SDataAccept	Acknowledge signal from Slave to Master indicating that the slave has accepted the pipelined write data from master.

Table 3.1: Basic OCP Signals.

Of the ten signals in this group, only Clk and MCmd signals are mandatory. All others are optional. The Master can issue 8 different commands using `MCmd`:

- (**IDLE**) - Idle.
- (**WR**) - Write.
- (**RD**) - Read.
- (**RDEX**) - ReadEX
- (**RDL**) - ReadLinked
- (**WRNP**) - WriteNonPost
- (**WRC**) - WriteConditional
- (**BCST**) - Broadcast

The detailed meaning for these commands can be found in [7]. The Slave can post one of the following 4 responds to `MCmd`:

- **NULL** - No Response.
- **DVA** - Data Valid/Accept.
- **FAIL** - Request Failed.
- **ERR** - Response Error.

Simple Extension provides additional support for master and slave cores with more complex communication requirements. Simple extension includes byte enable, multiple address spaces , and the addition of in-band socket-specific information to any of the three OCP phases. See Section 3.1.2 for more information on the protocol phases.

Burst Extension provides support for linking multiple otherwise independent OCP transfers in a chain. It optimizes data transfers by cutting down overlaps in control overheads such that related data transfers can be done all at once instead of individually. OCP offers different sequence types for bursting. Here we mention the main ones:

- INCR : increments the address by the specified OCP word size every clock cycle for regular memory type accesses.
- STRM : streams data to or from a target where address stays constant during a burst.
- DFLT1 : User-specified address sequence where maximum packing is required.
- DFLT2 : User-specified address sequence where packing is not allow.
- WRAP : Like INCR, but the address wraps at OCP word size indicated by the signal MBurstLength.

Thread Extension enables communication concurrency. Without it, strict transfer phase ordering must be enforced: each transfer request must be completed by the slave in the same order as they are presented by the master. The transfers in each thread must remain in-order with respect to each other, but the order between threads can be altered between request and response.

Sideband Signals

The sideband signals are outside of the formal transaction phases as are the dataflow signals. System designers can choose to use sideband signals to add a few more control features to the OCP interface such as global reset, interrupts, errors, and flags.

Test Signals

Designers can also add testing features to the interface. These signals are used for scanning of the interface to an IP core, clock control during scan operation, and JTAG (IEEE 1149.1) debug and test interface for the OCP.

3.1.2 Protocol Phases and Ordering

The semantics of OCP define signal groups, phases, transfers, and transactions. Signals are assigned to signal groups which are activated in accordance to protocol phases. There are three signal groups and phases: request, response, and datahandshake. Table 3.2 shows the three OCP protocol phases indicating the beginning and end of each phase. Request phase always

	begins when	**ends when**
Request Phase	Master drives $MCmd \neq IDLE$	Master samples $SCmdAccept = true$
Datahandshake Phase	Master drives $MDataValid = true$	Master samples $SDataAccept = true$
Response Phase	Slave drives $SResp \neq NULL$	Slave samples $MRespAccept = true$

Table 3.2: OCP protocol phases

precedes the response phase. This means that if the slave has not responded to the master by raising SCmdAccept, then the master must not post a new request until the previous request has been accepted by the slave. This is an effective way to avoid adding registers for temporary storage, and can especially be useful for the implementation of the NA. We shall see in Chapter 5, Implementation of the Network Adapter, how this feature of the OCP is used to save area on the NA. If datahandshake phase implemented, it should always be between the request phase and the response phase. For more precise definition of phase ordering within a transfer, please refer to Chapter 4 of [7].

While communicating, signals from the same signal group must all be valid and held steady at the same time from the beginning of the protocol phase until the end of that phase. Every transfer has a request phase. Read-type requests always have a response phase. For write-type requests, the OCP can be configured with or without responses or datahandshake. A write-type request without a response completes the communication transfer upon completion of the request phase.

3.2 DTU OCP Specification

MANGO has its special set of OCP signals and definitions. The DTU OCP signals is a small subset of the full OCP signals suite and is configured for the purpose of supporting MANGO components described in Chapter 2. All IP cores on MANGO should comply to the DTU NoC Specification [10], which is implemented by the NA. This section discusses some of the main aspects of the DTU OCP Specification focusing on the parts which are of special interest to the design of the NA.

3.2.1 Signal meaning for GS traffic

One of the main purposes of having the DTU OCP is to specify a set of OCP signals such that they can communicate special meanings for handling guaranteed service setup and teardown in MANGO. The DTU OCP subset assigns its own interpretation for some of the standard OCP signals. The NA must interpret these signals and handle the different types of requests and responses accordingly both at the OCP interface to the core and the network interface to MANGO.

Packet Type Recognition Signals

MANGO uses four signals to specify the type and the amount of GS service to the NA. (The meaning of the type and amount is clarified in [10].) These signals are: MCmd, MReqInfo, MFlag, and MData. Using MCmd and MReqInfo, the NA can determine the type of request the master core is presenting, either GS setup, GS use, GS teardown, or BE use, depending on the value of the two signals. For the signal encoding of MCmd and MReqInfo, please see the DTU GS-OCP Specification [10]. For connection setup, MFlag and MData encodes the service type and the type amount for request and response paths respectively.

3.2.2 Signal subset

The basic DTU OCP signal subset includes the signals shown in Table 3.3. Some of these signals are not assigned a signal bit-width in the DTU OCP Spececification [10]. In order to use these signals for implementation purpose, the signal bit-widths are assigned as shown in the table.

The reason that MDataValid and SDataAccept are eliminated in the NA implementation is because these two signals are datahandshake phase signals. Datahandshake phase of the OCP protocol is not necessary for MANGO at this point in time because it is typically only useful for master and slave devices that require the throughput advantage available through transfer pipelining. Transfer pipelining is not supported at this moment, but may be considered for future MANGO development. We can completely eliminate the datahandshake phase because the datahandshake phase is completely independent of the request and response phases. It allows the decoupling of a write address from write data which is not of interest at this moment.

Signal Name	Bit-Width	Driver	Meaning
clk	1	varies	OCP clock signal.
MCmd	3	Master	Transfer command.
MAddr	32	Master	Transfer address.
MData	32	Master	Write data.
MFlag	32	Master	Transfer connection identifier. Specifies GS type and amount in the forward direction during GS setup phase.
MReqInfo	2	Master	Transfer GS service command.
MRespAccept	1	Master	Master accepts response.
MDataValid	1	Master	Write data valid. (Not implemented)
MDataInfo	8	Master	Transfer master core's address to slave core. (Added to the signal subset)
SCmdAccept	1	Slave	Acknowledge signal for MCmd.
SData	32	Slave	Read data and GS connection identifier (connection ID).
SResp	2	Slave	Transfer response signal.
SDataAccept	1	Slave	Slave accepts write data. (Not implemented)
SDataInfo	32	Slave	Identifying Slave core for connection setup. (Added to the signal subset)

Table 3.3: DTU NoC OCP Signal Set

Signals added to the DTU OCP Signal Set

Two extra signals, MDataInfo and SDataInfo, are added to the DTU OCP interface. MDataInfo, 8 bits wide, driven by the NA OCP master, is used to indicate the source address of the packet to the slave core. For further clarification on its usage please see Chapter 4 Section 4.4. SDataInfo, 32 bits wide, driven by the slave, is used to indicate the address of the slave core to identify a connection along with the connection ID to the master core when connection set up was successful.

3.2.3 Protocol Transaction and timing

The DTU OCP follows the phasing order and timing defined in the OCP Specification [7]. The following timing diagram describes a valid OCP transfer. It shows a typical request and response phase transaction at the OCP interface between the a master OCP entity on the core and a slave OCP entity on the NA.

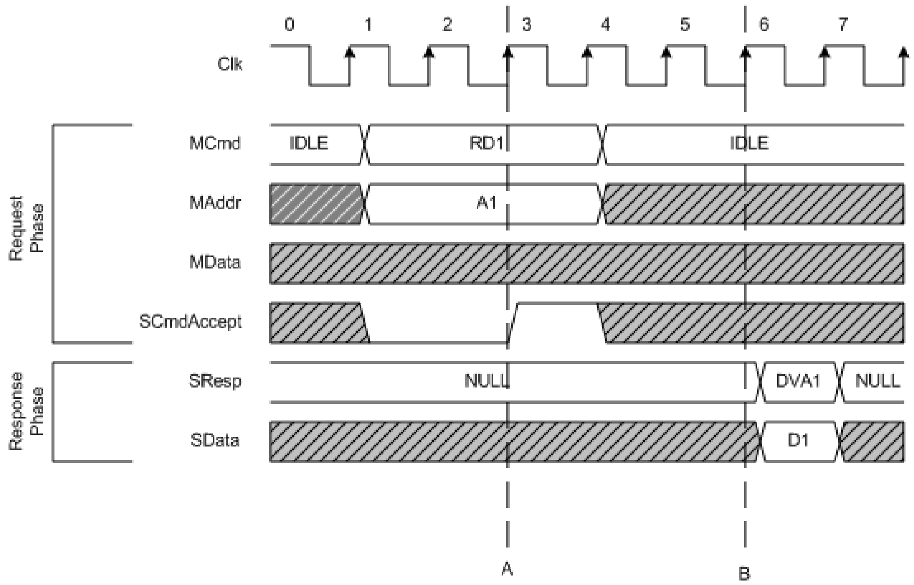

Figure 3.3: Timing Diagram for an OCP read request handshake and separate response.

At point A, the NA slave raises SCmdAccept, which means it has packetized the master's request. If the there is a congestion to send the packet at the network interface end, the NA will not raise SCmdAccept until the congestion is eliminated. At point B, a response to the read request was presented to the master. The number of clock cycles from the request to the command accept is referred to as *request-accept-latency*. The number of clock cycles from the master posting a request to the slave responding to that request is referred to as *request-to-response-latency*. For the transaction in Figure 3.3, the request-accept-latency is 2 and the request-to-response-latency is 5.

In this Chapter, the OCP protocol was briefly introduced. In particular, the DTU OCP signal subset was presented and the signal meanings were described. These signals will be referred and used in the following chapters.

Chapter 4

Network Adapter Conceptual Design and Methodologies

Contents

This chapter is intended to give the reader a conceptual understanding of the design issues involved in the making of the DTU NoC network adapter.

The chapter introduces the reader to the main design challenges at a conceptual level without diving into the details. However, it is strongly advised that the reader should read the DTU GS-OCP Specification (Appendix Aand the Programmer's Model for DTU NoC (Appendix B) before reading this chapter in order to understand the terms referenced. We begin by first talking about the possible types of the network adapter, what their differences are, and how they operate in the NoC. Secondly, we discuss how the network adapter handles end-to-end flow control in terms of connection setup and connection teardown. Thirdly, we briefly describe all the components of the network adapter in a schematic diagram and give a general view of their workings. Fourthly, we present the design of the packet format for all traffic types in the DTU NoC. Finally, we look into some design issues involving the network interface to the NoC in terms of scheduling over multiple virtual channels and congestion avoidance.

4.1 Types of Network Adapter

Referring to Figure 3.1 on page 28, it was shown that there can be three different instantiations of the NA. An instantiation can have a slave OCP module to correspond to a master core, a master OCP module to correspond to a slave core, and a combined master/slave module to correspond to a core with both master and slave functionalities.

4.1.1 Slave Network Adapter

The OCP interface of a slave NA can only receive OCP master signals and respond with OCP slave signals. (See Table 3.3 for signal reference.) The slave NA contains an OCP slave control unit which controls the signal timing and communication phase ordering at the OCP interface. A slave NA only recognizes response type packets and output request type packets at its network interface.

4.1.2 Master Network Adapter

Likewise, for a master NA, the OCP interface can only receive OCP slave signals and present OCP master signals. A master OCP control unit manages the OCP signals at the OCP interface to the slave core. The master NA outputs response type packets and recognizes request type packets at its network interface.

4.1.3 Duplex Network Adapter

The duplex NA has both a master and a slave OCP module. It handles all type of network traffic for both the input and output directions, and can present both master and slave OCP signals at its OCP interface to the attached core.

This thesis work includes the design for all three types of NAs. However, only the slave NA was implemented and tested due to timing limitations. For readers interested in further developing the NA and implementing the other NA types, please see Chapter 8, Future Work, for design ideas and directions.

An important job for the NA is to manage guaranteed service connection setup and teardown services and end-to-end flow control. The following sections discuss these topics and present a solution for this design.

4.2 End-to-End Flow Control and Managing Guaranteed Service Connections

From Chapter 2 we have discussed that the communication in MANGO is partitioned to abstraction layers. The IP core in the NoC resides on the Core Layer (Figure 2.2 on page 22), relies on the end-to-end communication service provided by the NA Layer beneath. This means that IP cores can communicate on a source-to-destination basis without having to know how the actually communication is carried across to its destination. Therefore, one of the major task for the NA is to provide this end-to-end service to the IP cores. In the following sections we will discuss how the NA manage this task of isolating the IP cores from the details of the underlying communication hardware.

4.2.1 Message segmentation and reassembly

The first thing the NA does when presented with a request or response by the IP core is to recognize what type of packet this message will be packetized into. It does this by looking at the encoding of the master signals: MCmd and MReqInfo, as mentioned in Section 3.2.1. The different packet types will be discussed later in Section 4.4 of this chapter. Once the packet type has been determined, the NA can segment the message into smaller units or flits to be transported to the network. At the virtual channel input port of the NA, flits are collected and assembled to form a complete message. The NA is then responsible for recognizing the type of packet, and manage the content of the packet accordingly. It does so by either presenting the payload to the core if this is a normal communication packet, or write to certain tables or delete certain entries in the NA if the packet is a connection setup or teardown packet.

4.2.2 Multiple Connection Management

Some IP cores are multi-programmed and will need to address multiple cores at the same time. This implies that multiple (GS) connections will be entering and leaving each IP core. There needs to be some way to tell which message belongs to which connection. This requires some kind of naming mechanism, so that a process on the IP core has a way of specifying with whom it wishes to converse. The NA does this by providing a connection ID to each GS communication path the master core requests and it keeps track of these connections by using a connection ID table which maps each connection ID to the actual virtual channel which this connection was set up to.

In addition to handling connections, there must also be a mechanism for regulating the flow of information, so that a fast process on the IP core cannot overrun a slow one. The NA handles this problem by not asserting the SCmdAccept until the current request message is packetized. Not asserting SCmdAccept forces the master core to hold its request phase signals constant until SCmdAccept is asserted. This avoids the problem of accepting a new command before the first command has been taken care of.

The following two sections discuss how a GS connection is setup and torn down and what the NA does in each of these situations.

4.2.3 Connection Setup

To establish a GS connection to a slave core, the master core first presents the connection setup request to the slave NA. The NA recognizes the request and packetizes a connection setup request packet to be sent to the Network Controller (NC). As mentioned in Section 2.2.1, the NC is a special core in the NoC that is responsible for all connection setups and teardowns. When the NC receives the setup request, it tries to carry out the command and to establish the connection. The NC will then respond to the master core by sending a GS connection setup response packet to the master core's NA. The NA depacketizes it and if the connection setup was successful, it will store the virtual channel number (also called the *injection ID*) of this connection in the Connection ID table. There are several outgoing and incoming virtual channels at the Network Interface of a NA. Some of these virtual channels are used only for best-effort traffic while others are reserved for different GS connections. After the new entry is recorded in the connection ID table, the NA will respond to the master core by presenting a connection ID for the newly setup communication path as well as the address of the destination core to which this connection was setup to. This is done so that the master core can distinguish between different connections if it had requested for several connections to be setup to multiple cores within the NoC.

4.2.4 Connection Teardown

To delete a GS connection to a slave core, the master core presents a connection teardown request to its attached slave NA. The NA then packetizes this request and send a teardown request packet to the NC. After receiving this request, the NC will teardown the requested connection path and send a GS connection teardown response packet to the master core. The NA receives the teardown response packet. If the connection teardown was sucessful, the NA will delete the connection entry of this communication path in its connection ID table and indicate to the master core that the connection was successfully deleted. If the connection teardown was unsuccessful, the NA will not delete the connection ID table entry and an error message will be sent to the master core.

4.3 Network Adapter Component Overview

In Section 4.1, we introduced three types of the network adapter: a slave NA, a master NA, and a duplex NA. In this section, we will present the slave NA block diagram and give an introduction to the workings of the modules within the slave NA. The implementation details of the slave NA will be given in Chapter 5. For readers interested in developing the master NA and the duplex NA please see Chapter 8 for design proposals.

The slave NA consists of 7 components. They are shown in Figure 4.1.

ReqE2E, also known as the Request End-to-End Flow Control Unit, is mainly responsible for the following tasks:

- receiving requests from the master core and responding to them in compliance to the OCP protocol.
- recognizing the request type and determining which signals to pass on to the Encap unit for packaging.
- receiving responses from the Decap unit and presenting them to the master core in accordance to the OCP protocol.
- setting the input signals to both the Connection ID table and the Route Lookup Table.

Connection ID Table is memory mapped to the global memory address space specified in [8]. It is used to store injection channel IDs or virtual channel numbers for incoming and outgoing ports when a guaranteed connection is set up. The Network Controller can write to the Connection ID Tables using the GS Response Setup/Teardown packets, which will be described in Section 4.4.

Route Lookup Table contains all the routing information from this particular NA to all other cores on the DTU NoC. The route paths, which

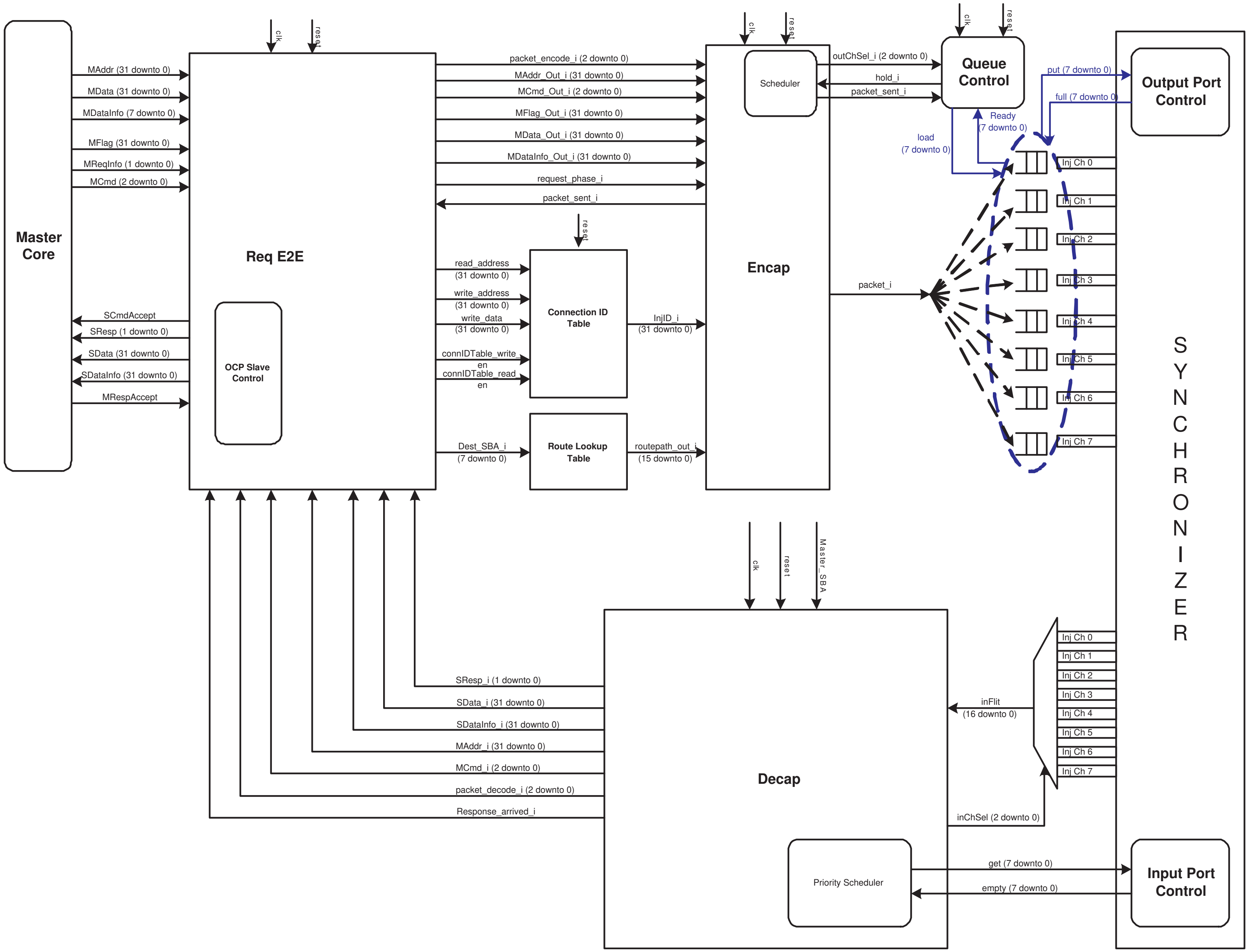

Figure 4.1: Block diagram for the slave network adapter.

are stored in this table are configured into the table at NA instantiation time. The route path will be a part of the packet header for best-effort packets.

Encap, called the Encapsulation Unit, is responsible for receiving messages from ReqE2E, Connection ID Table, and Route Lookup table, segmenting the input message and packetizing them into packet flits to be sent to the network. The Encapsulation Unit also communicates with the Queue Control because it injects packets into the output queues. If the queue is full, the Encapsulation Unit will be informed so that it will not send the packet until the output queue is available again. When this happens, the Encapsulation Unit also send a signal back to the ReqE2E Unit so as to stop it from receiving new requests from the master IP core.

Queue Control monitors the status of all eight output queues and asserts the enable signal for the output queues if the Encapsulation Unit decide to inject a packet to a queue and the queue is ready to receive a new packet. If the queue is not ready to receive a new packet, the Queue Control will inform the Encapsulation Unit to stop from receiving new commands.

Output Queues are used as storage space for each outgoing virtual channels so that if one virtual channel is blocked, the Encapsulation Unit can choose to inject into another available virtual channel.

Decap, also called the Decapsulation Unit, is responsible for choosing packets among all 8 input virtual channels according to their priorities and order, depacketize the packet, and presenting the contents to ReqE2E Unit when the packet decoding is done. The Decapsulation Unit contains the priority scheduler which communicates with the input port control unit on the Sync component.

4.4 Packet Format

An important and crucial part for the design of the NA is to decide on the different types of traffic in the NoC, to define what needs to be in the headers of each abstraction layer, and to decide on a fixed packet format for all packet types. These tasks are important because it is the responsibility of the NA to make packets to be send to the network and it is its responsibility to recognize all traffic types in order to handle the packet contents properly.

This section describes the packet format for MANGO. The names of the fields in the packet formats are signals defined in the DTU GS-OCP Specification [10].

4.4.1 Defining the Packet Types and GS Connection Types

There are six types of Best Effort packets.

1. GS Setup Request
2. GS Setup Response
3. GS Teardown Request
4. GS Teardown Response
5. BE Request
6. BE Response

The first four packets are used for the setting up and tearing down of guaranteed service connections, and are used to configure the NA accordingly. The last two are simply general use best effort packets. All best effort packets have the same headers.
There are two types of Guaranteed Service packets.

1. GS Request
2. GS Response

Both of these two GS packet types are simply for message passing between IP cores and are not used for any network configuration purposes. The GS packets do not require any routing information because these packets are sent through a guaranteed connection channel across the network. The guaranteed connection channels can be seen as a pipe linking one IP to another IP. Whatever goes in at one end of the pipe will show up in order at the other end. The guaranteed service connection has previously been setup by the Network Controller after receiveing a connection setup request from the master core via a best effort GS Setup Request Packet. After the guaranteed connection is setup, the Network Controller responds with a best effort GS Setup Response Packet to either the master core or the slave core or to both, depending on the type of guaranteed service connection that was requested.
There are three types of GS connections:

1. Forward GS, return GS (Requires both outgoing and incoming injection ID, therefore a GS Setup Response packet is sent to both the master and the slave.)
2. Forward GS, return BE (Requires outgoing injection ID, therefore a GS Setup Response packet is sent to the master)
3. Forward BE, return GS (Requires incoming injection ID, therefore a GS Setup Response packet is sent to the slave)

The guaranteed service connection is torn down when the master core sends a best effort GS Teardown Request packet. After the Network Controller tears down the connection, it responds to the master and/or slave core using a GS Teardown Response packet, depending on which one of the above GS connection types was setup.

In the following sections, we will describe each of eight packet types mentioned above in detail beginning with the best-effort packets and followed by the guaranteed service packets. Packets are composed of a number of flits. Each flit in MANGO is 17 bits: 1 bit for control and 16 bits for content. The flit control bit is 1 if the flit is the last flit of the packet, otherwise it is 0.

4.4.2 Best Effort Packet Format

Packet Control Bit	Flit Content			
16	15	12 11	3	0
0	Route Path			
0	Destination Address (1 of 2) (bytes 1 and 0)			
0	MCmd	T/S Response Type	Source Core	Reserved
0	Destination Address (2 of 2) (bytes 3 and 2)			

Figure 4.2: Best Effort packet header

The BE packet header consists of four flits. The format of the header is shown in Figure 4.2 and payload flits will follow. The packet is transmitted in the flit order shown from top to bottom.

The description of each field in the basic header is as follows.

Route Path: The route path contains the route directions to the destination core. 16 bits in the route path field allow a maximum of 8 hops, as each hop occupies 2 bits of direction encoding. This limits the size of the network to maximum 5 by 5 mesh grid.

The direction encoding is as follows:

- N : 00
- E : 01
- S : 10
- W : 11

Destination Address: Destination Address specifies the memory location

that this packet is addressing. As mentioned in [8], byte 1 of the address field distinguishes the different components within one destination core: `00` to `FC` for IP core addressing, `FD` for NA control, `FE` for Routing Node control, and `FF` is reserved. The specific meaning of Destination Address for different BE packets will be detailed in later sections.
MCmd: MCmd has different meanings depending on whether this packet is a BE packet used for GS setup and teardown or a BE packet for general use. For all response BE packets, MCmd will always be `000`. Therefore, MCmd is used to determine whether a packet is a request or a response packet. The details of the different MCmd encoding for different BE packets will be further clarified in later sections and is described in [10].
T/S Response Type: The T/S Response Type field is only relevant when the packet is a teardown or setup response packet with MCmd equals to `000` and byte 1 of MAddr is `FD`. This field differentiates between a GS teardown response BE packet and a GS setup response BE packet: `1` for setup and `0` for teardown.
Source Core Address: The Source Core Address contains the system base address (SBA) of the source core. For more about SBA, see [8]. The specific meanings of Source Core Address for different BE packets will be detailed in later sections.

GS Setup Request Packet

A GS Setup Request packet is sent by a master core to the Network Controller when the master core wishes to request a guaranteed service connection to a slave core. The payload of a GS Setup Request packet is shown in Figure 4.3.

Packet Control Bit 16	Flit Content 15 … 0
0	MData (bytes 1 and 0)
0	MData (bytes 3 and 2)
0	MFlag (bytes 1 and 0)
1	MFlag (bytes 3 and 2)

Figure 4.3: GS Setup Request packet payload.

Destination Address: Destination Address in the header field is mapped to master core's MAddr signal, which tells the Network Controller which slave core to establish the guaranteed connection to. MAddr should be assigned as shown in Table 4.1.

3rd byte	SBA of the slave core
2nd byte	FF
1st byte	FD
0th byte	Don't care

Table 4.1: MAddr assignment for GS setup and teardown requests

MCmd: MCmd in the header field will be assigned to RD.
Source Core Address: Source Core Address in the header field contains the SBA of the master core. The Network Controller needs the master core's SBA to set up a guaranteed service connection between the master core and the slave core, and to know which IP core's Connection ID table to write to after the connection is set up. The NA will map the Source Core Address to MDataInfo as a part of the master signal group to the Network Controller's OCP slave.
MData: MData contains the type of GS connection and the amount of service required in the return direction, from the slave to the master.
MFlag: MFlag contains the type of GS connection and the amount of service required in the forward direction, from the master to the slave.

GS Setup Response Packet

A GS Setup Response packet is sent by the Network Controller to the master core or the slave core or both after a guaranteed connection is set up. For GS connection setup for both the forward and return directions, the Network Controller will assign the same connection ID to both the master core and the slave core, meaning that both cores receive a GS Setup Response packet. In order to devise an easy and unique way to generate connection IDs for the IP cores, it was decided that the lower 3 bytes of the addresses mapped for Connection ID Tables can be used as Connection IDs. The 4th byte or the most significant byte of the connection ID Table address is the SBA of the IP core which is attached to that particular NA. Therefore, when the same connection ID is assigned to both the master and the slave core, data will be written to the same address locations of the two Connection ID tables. The payload of a GS Setup Response packet is shown in Figure 4.4.
Destination Address: Destination Address in the header field contains a Connection ID Table address. Therefore, the Network Controller will directly write SData to the Connection Table in the NA after the connection is setup. This Connection ID Table address will then be used as the connection ID for this GS setup as mentioned earlier. This ensures that all the connection IDs in the whole network are unique, since no two addresses in the address space are the same.
MCmd: MCmd is 000.

Source Core Address: The Source Core Address contains the SBA of the Network Controller.
SResp: SResp specifies whether the GS connection requested was successful or not: 1 for success and 0 for failure.
SDataInfo: SDataInfo contains the address of the destination core which this connection was set up to. SDataInfo and the Connection ID will be sent to the master core. This allows the master core to match this GS connection response to its GS connection request.
SData: SData contains the outgoing and incoming injection channel IDs for this GS connection. Since currently the priorities of the incoming and outgoing virtual channels are statically assigned, the incoming injection ID does not have any actual effect on the operation of the NA. However, for future modifications with incoming virtual channel priorities being dynamically assigned, all incoming and outgoing channels will have its priority dynamically changing. The different priority assignments of these injection channels will affect the priority scheduler, which selects the flits from the incoming virtual channels for depacking.

GS Teardown Request Packet

A GS Teardown Request packet is sent by the master core to the Network Controller when the master core finishes communicating with the slave core and wishes to tear down the communication channel. The payload of a GS Teardown Request packet is shown in Figure 4.5.
Destination Address: Destination Address in the header field has the value of the master core's MAddr signal, which tells the Network Controller which slave core the guaranteed connection was set up to. MAddr assignment is the same as for GS Setup request and is shown in Table 4.1.
MCmd: MCmd in the header field will be assigned to WR.
Source Core Address: The Source Core Address contains the SBA of the master core.

Packet Control Bit	Flit Content	
16	15	13 … 0
0	SResp	Reserved
0	SDataInfo (bytes 1 and 0)	
0	SDataInfo (bytes 3 and 2)	
0	SData (bytes 1 and 0)	
1	SData (bytes 3 and 2)	

Figure 4.4: GS Setup Response packet payload

MFlag: MFlag contains the connection ID or the Connection ID Table address for the connection to be torn down. This will inform the Network Controller, so this connection ID can be reused for future connections.

GS Teardown Response Packet

A GS Teardown Response packet is sent by the Network Controller to the master core or the slave core or both after a guaranteed connection is torn down. The payload of a GS Teardown Response packet is shown in Figure 4.6.
Destination Address: Destination Address in the header field contains the ID of the GS connection that was torn down. Since the Connection ID is also the Connection ID Table address, the Network Controller can directly inform the receiving NA to change the data in the specified Connection ID Table entry to an injection channel with BE-low as its priority. Therefore, resetting the Connection ID Table entry. The destination address is also mapped to SData and sent to the master core, so that the core will know the which connection ID this connection teardown is associated with.
MCmd: MCmd is 000.
Source Core Address: The Source Core Address contains the SBA of the Network Controller.
SResp: SResp indicates whether the teardown was successful or not: 1 for success and 0 for fail.

BE Request Packet

A BE Request packet is sent by a master core to a slave core when guaranteed service is not required. The payload of a BE Request packet is shown in Figure 4.7.
Destination Address: Destination Address contains the IP core address of the destination core to which the master core wants to read from or write to.
MCmd: MCmd meaning is specified in [7].
Source Core Address: The Source Core Address contains the SBA of the master core. It will be used to generate the return route path from the slave

Packet Control Bit 16	Flit Content 15 … 0
0	MFlag (bytes 1 and 0)
1	MFlag (bytes 3 and 2)

Figure 4.5: GS Teardown Request packet payload

Packet Control Bit	Flit Content	
16	15 13	0
1	SResp	Reserved

Figure 4.6: GS Teardown Response packet payload

Packet Control Bit	Flit Content
16	15 … 0
0	MData (bytes 1 and 0)
1	MData (bytes 3 and 2)

Figure 4.7: BE Request packet payload

core back to the master core.
MData: This field is only present when MCmd is WR. MData contains the data to be written to the destination core specified by the address in Destination Address.

BE Response Packet

A BE Response packet can be sent by a slave core to a master core in response to a BE request packet. The payload of a BE Response packet is shown in Figure 4.8.

Packet Control Bit	Flit Content		
16	15	13	12 … 0
0	SResp	R/W	Reserved
0	SData (bytes 3 and 2)		
1	SData (bytes 3 and 2)		

Figure 4.8: BE Response packet payload

Destination Address: Destination Address contains the addressing information for the master core's NA. The field should be encoded as shown in Table 4.2.
MCmd: MCmd is 000.
Source Core Address: The Source Core Address contains the SBA of the slave core.
SResp: SResp meaning is specified in [7].

3rd byte	SBA of the master core.
2nd byte	any address from 0000 to FFFC
1st byte	
0th byte	Don't Care

Table 4.2: Destination Address assignment for BE response packet

R/W: R/W indicates whether this response packet is a response to a RD command or response to a WR command.
SData: Contains the requested data for a read request.

4.4.3 Guaranteed Service Packet Format

GS Request Packet

The GS Request packet is sent by a master core to a slave core after a guaranteed connection has been established between them. When sending GS packets, the master core must provide the connection ID using the MFlag signal. The connection ID tells the Network Adapter which injection channel has been setup for this communication path. The GS Request packet is shown in Figure 4.9.

Packet Control Bit	Flit Content		
16	15	12	4 0
0	MCmd	MFlag (byte 0)	Reserved
0	MAddr (1 of 2) (bytes 1 and 0)		
0	MAddr (1 of 2) (bytes 3 and 2)		
0	MData (1 of 2) (bytes 1 and 0)		
1	MData (1 of 2) (bytes 3 and 2)		

Figure 4.9: Guaranteed Service Request Packet

MCmd: The meaning of MCmd is specified in [7].
MFlag: MFlag contains the Connection ID Table index. Based on this field the NA for the slave core can find the correct outgoing injection channel when responding to this request.
MAddr: MAddr specifies the memory location that this packet is addressing. This address will only be in the range from `0000` to `FFFC` for bytes 1 and 2 because GS packets are used solely for communication between the master and the slave and not for connection setup or teardown as are BE packets.
MData: This field is only present when MCmd is WR. MData contains the data to be written to the destination core specified by the address in Destination Address.

GS Response Packet

The GS Response packet is sent by a slave core to a master core in response to a GS Request packet. The GS Response packet is shown in Figure 4.10.

Packet Control Bit 16	Flit Content 15	12	10	9 … 0
0	MCmd	SResp	R/W	Reserved
0	SData (bytes 1 and 0)			
1	SData (bytes 3 and 2)			

Figure 4.10: Guaranteed Service Response Packet

MCmd: MCmd is 000.
SResp: SResp meaning is specified in [7].
R/W: R/W indicates whether this response packet is a response to a RD command or response to a WR command.
SData: Contains the requested data for a read request.

4.5 Design issues at the Network Interface

The network interface (NI) is the interface between the NA and the NoC interconnect. As seen in Figure 2.3 on page 24 between the routing node and the NA a Sync component is found, which interfaces between the synchronous IP core and the asynchronous routing node. Figure 4.11 illustrates a routing node, virtual channels, links, and an interface to an attached NA. On the NA side of the Sync component the virtual channels are synchronous to the NA clock signal, while on the other side they are asynchronous. Currently, MANGO implements eight virtual channels on two NA virtual channel ports: the input virtual channel port and the output virtual channel port, which we will discuss separately in this section.

4.5.1 NA Input Port at the Network Interface

The input virtual channel port is where the NA receives packets in flits from the network interconnect. As mentioned in Section 4.4, each flit consists of 17 bits. At the rising edge of the clock, if a flit arrives at a virtual channel the input port control unit on the Sync will inform the NA by using the Empty signal. The NA will then respond by using the Get signal. This communication protocol is illustrated in Figure 4.12. When the Empty signal is high, the NA recognizes that the channel is empty, so there is no flit present and the Get signal is low. The NA is allowed to assert the Get signal as long as the Empty signal is low.

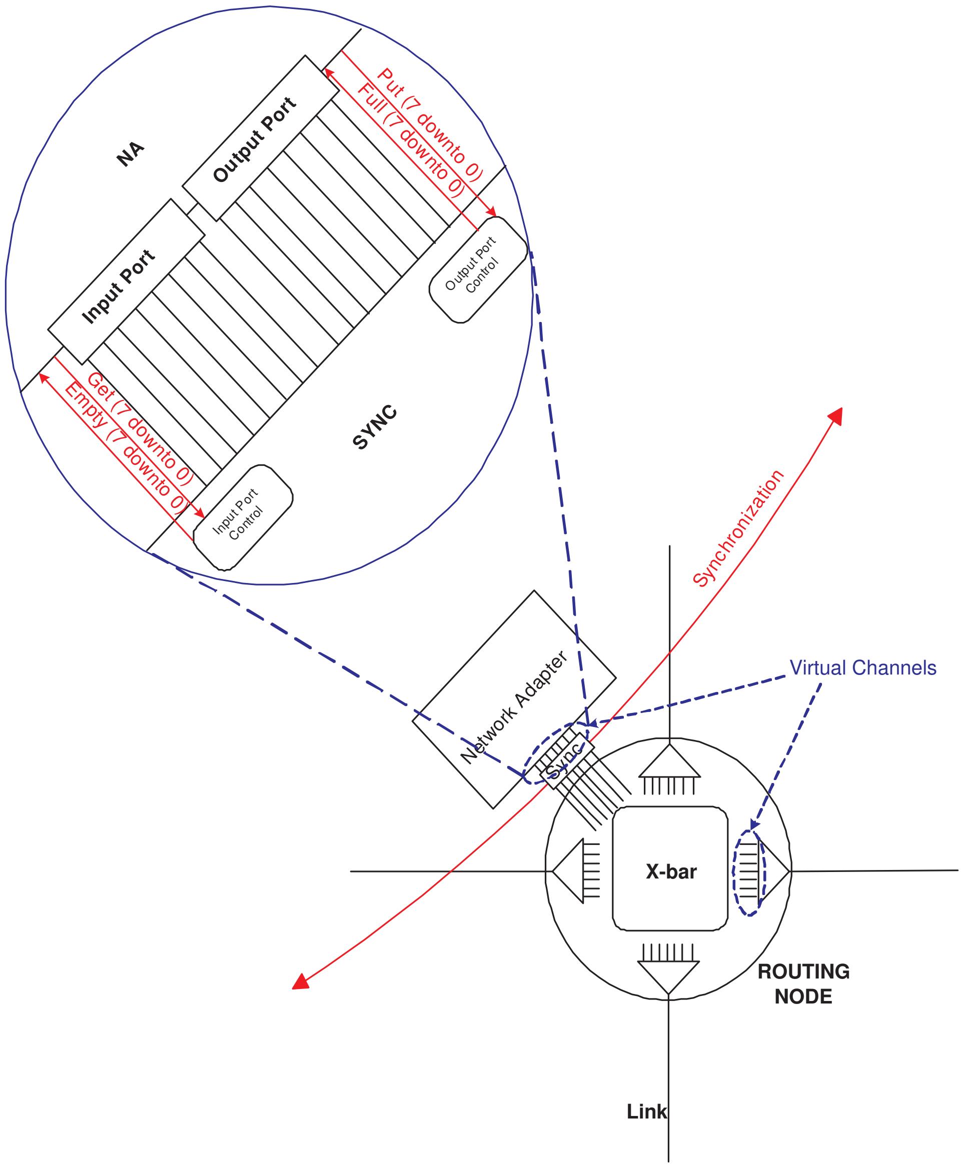

Figure 4.11: Network interface between the routing node and the network adapter. Virtual channels going through the Sync component are for both ingoing and outgoing directions.

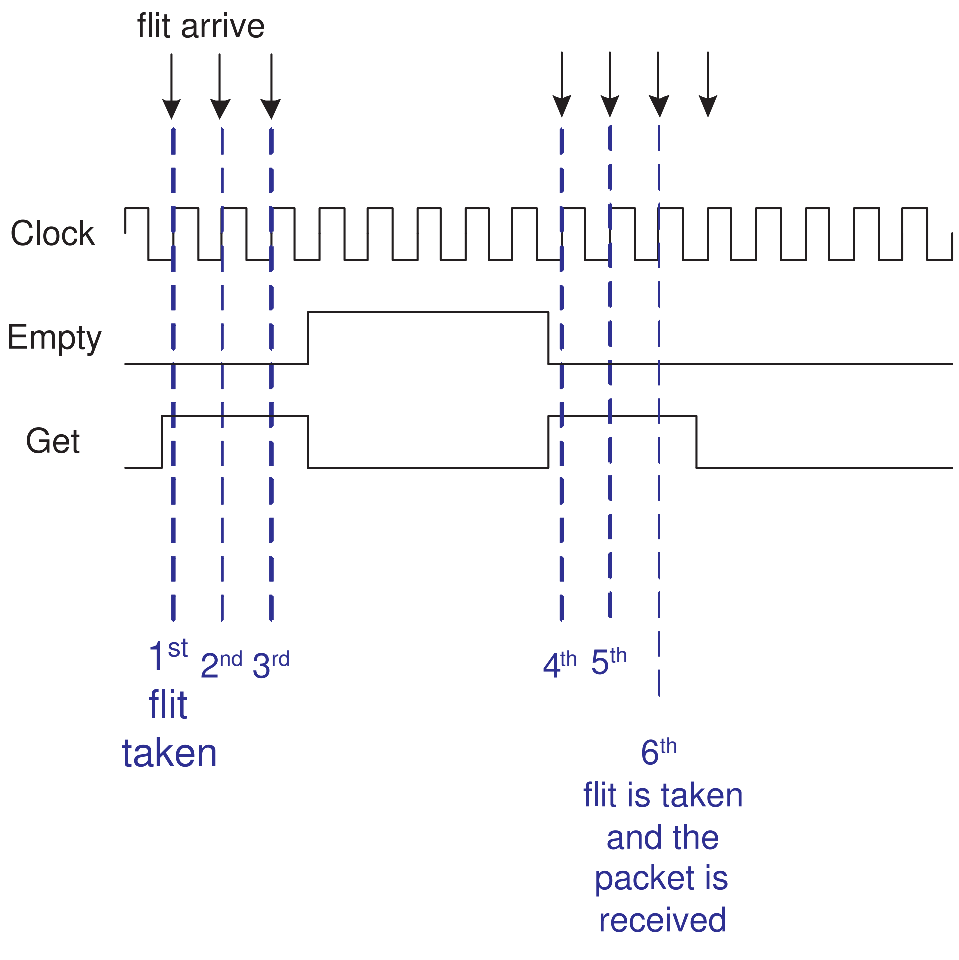

Figure 4.12: Communication protocol for one virtual channel at the network interface input port.

The above protocol holds for all 8 input virtual channels. Therefore both the Empty and Get has 8 bits.

Static vs. Dynamic Virtual Channel Priority Assignment

As mentioned previously, each virtual channel is assigned a particular priority. Currently, the NA implements four priority levels. The priority assignment for each priority level is as follows:

- Priority Level 3 - Guaranteed throughput (highest priority)
- Priority Level 2 - Guaranteed power
- Priority Level 1 - Guaranteed jitter
- Priority Level 0 - Best-effort low (lowest priority)

For implementation purposes, the actual names for each priority level are irrelevant and maybe subject for change.

The priority levels of the virtual channels are important for the NA when it comes to selecting which virtual channel to send a packet on and deciding which virtual channel to select next when receiving a packet. Priority level assignment for different traffic classes directly affects how a certain class of traffic can maintain a certain level of guaranteed service. For example, if a virtual channel has been set up to provide guaranteed throughput, the designer must ensure that when traffic arrives at that virtual channel, it will take priority over other less urgent traffic in order to assume a specified amount of guaranteed throughput is achieved. Therefore, the NA priority scheduler should choose the higher priority virtual channel for packet processing over lower priority channels. Once a channel is chosen for packet processing the NA priority scheduler will keep on choosing that channel until all the flits of that packet have been received in full. This mechanism ensures that one packet is processed at a time.

There are two possible ways of assigning priorities to virtual channels: static and dynamic. Static assignment means that the priority is set and fixed at NA instantiation time. The NA scheduler will always pick the virtual channels in a fixed manner. In this case, when a GS connection is setup by the network controller, there need not be any information to write to the scheduler. Traffic can directly arrive at the specified virtual channel and the NA will choose the channels according to the fixed priority assignment. Dynamic assignment, on the other hand, is done such that when GS connection is set up, the network controller must use a GS response packet to configure the NA scheduler such that the virtual channel to which the GS connection was set up to can be assigned a new priority and the scheduler will pick the channels accordingly. When a GS connection is deleted, the NA priority scheduler will need to reset its entry for that connection back to the lowest priority state.

Currently, the priority assignment for each input virtual channel to the NA is static. This was determined by the DTU SoC group so as to reduce the complexity of the scheduling interface at this beginning stage of DTU NoC development.

4.5.2 NA Output Port at the Network Interface

The output virtual channel port is the place the NA injects flits into the network. The communication protocol is described in Figure 4.13. At the rising edge of each clock the NA output scheduler checks if the virtual channel to which it wishes to inject a flit into is full or not. If the Full signal is high, that means the channel is blocked and no flits should be injected. The NA will then withhold the flit until the channel becomes unblocked again. This ensures that the network will not be congested and packets being blocked up. This is also an effective way to reduce buffer space and therefore area and power. The output port control unit on the Sync provides the Full signal to the NA and the NA responds with the Put signal. If the Full signal is asserted at the rising edge of the clock, the Put signal will not be asserted.

The NA has a certain amount of output queue buffer space for holding out going flits. If a channel becomes blocked the flits will be stored up in the queue. If however, the channel becomes blocked for a long time and there is no room in the output queues, the NA will stop accepting new commands from the IP core by not asserting the SCmdAccept OCP signal.

For more information about the network interface, please refer to The Network Interface Specification [9].

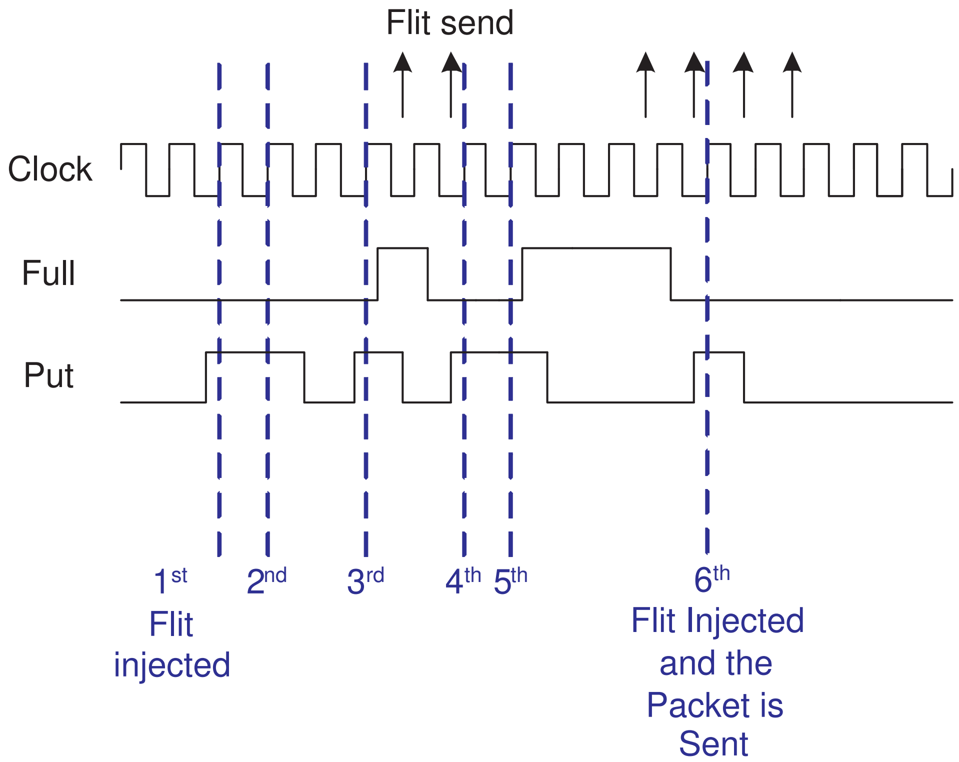

Figure 4.13: Communication protocol for one virtual channel at the network interface output port.

Chapter 5

Implementation of Network Adapter

Contents

In the last chapter, we introduced the network adapter in a conceptual perspective by examining some of the main tasks that the network adapter must perform. We also presented the packet format for the DTU NoC and gave a general introduction to each of the components on the slave network adapter. This chapter, we continue with a more in-depth analysis of the slave network adapter and discuss issues of its implementation according to the packet format presented in Section 4.4.

5.1 Architecture

Computer architecture design entails several levels of abstraction. These abstraction levels can be categorized from high level to low level in the following order [11].

Algorithm level is the highest level of abstraction that deals with things such as data structures, sorting algorithms etc.

Architecture level implements the algorithms by means of building blocks such as CPUs, memories and communication infrastructure at the system level and multipliers, muxes and controllers at the register-transfer level (RTL).

Synthesis level translates the architectural level design into netlists of logic gates.

Circuit level describes individual logic gates created by connecting transistors.

Technology level consists of basic building blocks such as transistors (NMOS and PMOS) and wires.

We can see that the lower the level, the closer we get to the physical implementation of the electronic device. The NA is designed at both the algorithm level and architectural level and is implemented in **VHDL** (Very high-speed integrated circuit Hardware Description Language). VHDL gives freedom to describe an electronic design at various levels of abstraction. However, it is designed to describe hardware, hence it supports many lower level coding styles. There are two main style of coding in VHDL. They are behavioral modelling and structural modelling. Behavioral modelling is adopted when presenting an abstract description of a system's function. Such an architectural body includes only process statements, which are collection of actions to be executed in sequence. Structural modelling, on the other hand, is expressed in terms of subsystems interconnected by signals (wires). Each subsystems may in turn be composed of an interconnection of sub-subsystems and so on until the most primitive component is reached where processes can describe its behavior. This recursive or hierarchical structure of modelling is very useful when the designer wishes to describe the design in a more concrete and specific way.

Structural modelling of a design, in most cases, is more suitable for synthesizing to an actual netlist of logic gates because the style of coding is typically less abstract and ambiguous. The slave NA is coded in the behavioral modelling style because it is more important at this point to model the behavior of the NA and solidify its specifications and design than it is to realize the design in hardware. However, some performance estimates such as the NA's area, power consumption, and latency will be given in Chapter 7, as a general guideline for future developments of the NA.

5.2 Module Design and Programming

In this section, we will begin a detailed analysis of the design of each of the components within the slave NA. We will start with the network interface input port and work our way around towards the network interface output port. Please refer to Figure 4.1 on page 42 for details of interconnections between the NA components.

5.2.1 Decapsulation Unit

We have already introduced the challenges of the NA input port at the network interface in Chapter 4 Section 4.5.1. In addition to addressing the flow of traffic at the input port by performing priority scheduling and data handshaking, the Decapsulation Unit is also responsible for reassembling the incoming packet from its segments, recognizing which type of packet it is, extracting the contents and presenting the contents to either the ReqE2E unit for response packet, or the RespE2E unit for request packet. We will address each of these responsibilities in the following sections in the order presented.

Figure 5.1 details the design of the Decapsulation Unit (Decap). On the righthand side of Decap is the interface to the NA input port. We have 3 control signals, empty, get, and inChSel and 1 data signal inFlit. Empty and get are handshake control signals for the 8 incoming virtual channels. The communication protocol was presented in Figure 4.12. InChSel is the control signal for the 8-input multiplexer at the input port. See Figure 4.1 on page 42. InFlit carries the actual flit coming in from the incoming virtual channels. On the lefthand side of Decap is the interface to other NA components: ReqE2E and RespE2E. Two signals of special importance are: response_arrived and request_arrived. These two signals are strobe signals to ReqE2E and RespE2E units respectively. They are used to inform the other two components that valid response signals has arrived. Although the NA implemented in this project is just a slave NA that can only receive response packets, the Decapsulation Unit implemented however was originally intended for a Duplex NA that can handle all packet types. Due to a limited time in this project, the Duplex NA was not feasible to finish and the Decapsulation Unit was left as it is for handling all packet types. For Duplex NA design suggestions, please see Chapter 8, Future Work.

Priority Scheduler

The most complex part of the design of the Decapsulation Unit is the Priority Scheduler. The purpose of the Priority Scheduler is to select among the 8 incoming virtual channels one that is going to be chosen to receive a packet next. The choice of channel selection is based on the priority of that virtual

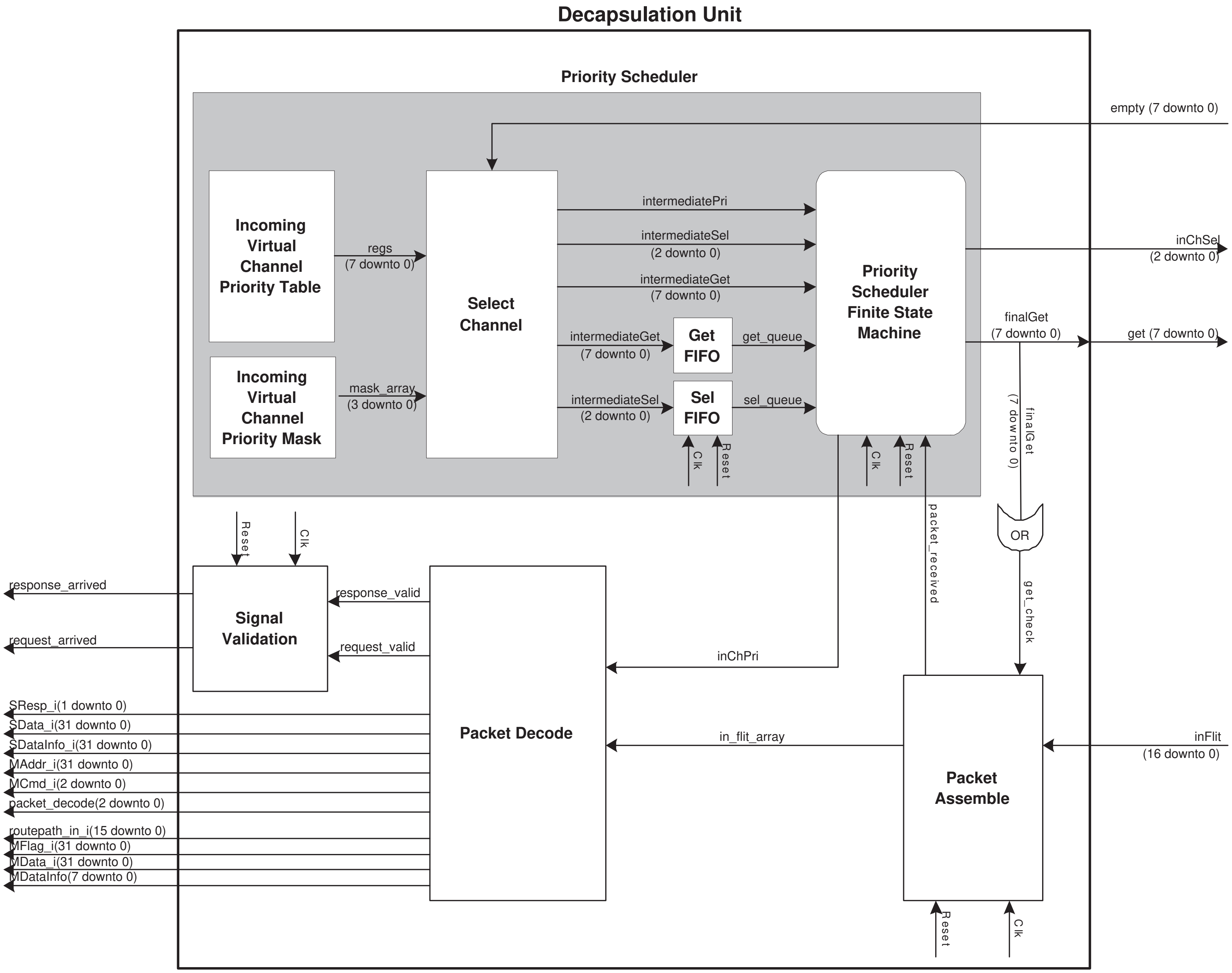

Figure 5.1: Block diagram of processes and signals within the Decapsulation Unit.

channel, which is provided by the Priority Table, and on whether there is a flit of the packet waiting at that channel, which is provided by the empty signal. Each bit of the empty signal represents a virtual channel. Once the decision to choose a virtual channel is made, the Priority Scheduler asserts the get signal for the selected channel and picks that channel using the inChSel signal. The Priority Scheduler can be viewed in three separate groups.

Group One includes the Priority Table, the Priority Mask and the Select Channel processes. This group is responsible for selecting the channels at the time the bits of the empty signal changes, meaning that flits from different channels arrive at random times and the decision of which channel to choose next is dependant on the time order in which the flits arrive and on the priority of the channels.

Group Two includes the Get FIFO and the Sel FIFO. These FIFOs store the get and inChSel signals from group one. Since the packets arrive at the incoming virtual channels at unpredictable times, the priority scheduler must remember the order the packet arrive and selects the channels accordingly. The FIFOs preserve the order of the channel selections and keep these values until they are used.

Group Three is just the Finite State Machine (FSM). It is there to assert the get and the inChSel signals. The FSM makes sure that when a channel is selected, that channel will be kept on selected for consecutive clock cycles until all flits of the packet from that channel have been received. We know from Section 4.4 that different packet types have different lengths and hence consist of different number of flits. When the FSM is ready to choose the next virtual channel, it gets the next decision from the front end of the FIFOs which stores the decisions in order.

Now we discuss each of these three groups in more detail.

Priority Table, Priority Mask, and Select Channel

The Select Channel process uses the Priority Table and Priority Mask to make incoming channel selection decisions. The Select Channel process is quite complex due to the following reasons.

1. It must always choose channels of higher priority over channels of lower priority.
2. Since there can be more than one channel of the same priority, it must for each priority level note all the channels that have a packet waiting to be received and select one of those channels.

3. To be fair in choosing channels of the same priority, if there are flits waiting in two channels of the same priority and one channel has just been chosen in the previous turn and the other has not, then the Select Channel process will choose the channel that has not been serviced yet. It does this by marking channels that have been serviced before and choosing the ones that have not been marked.

4. Since there is a possibility of having more than 1 channel of the same priority that has not been chosen before, the Select Channel process must keep track of those channels so they may not be missed out.

To perform the above tasks, the Select Channel process uses the Priority Table, which tells it the priority assignment of each incoming virtual channel, and the Priority Mask, which tells it the positions of the channels that are of the same priority. The Priority Mask is used when the Select Channel process needs to clear old tag values for marking selected channels. The masks are there so that old tags of the same priority can be cleared without touching tags for other priority levels. To explain the Select Channel process better we look at a flow chart diagram of the process in Figure 5.2. The task boxes have been numbered for easier reference later.

All four tasks and internal variables apply one priority level at a time. We will first define those variable names for clarification.

channel_selected : Keeps track of the number of channels of the same priority that have a flit waiting for processing.

tag : For fair channel selection, tag is used to mark channels that have been serviced. If after servicing a channel and another packet arrives at the same channel, this channel will not be pick for processing over another channel of the same priority that has not been serviced yet.

temp_untagged : Keeps track of all the channels of the same priority that have not been selected for service before. As the channels are being serviced one by one the marked bit in temp_untagged will be cleared correspondingly.

contain_flit : Indicates which channel currently has flits waiting to be received.

mask_array : This array contains the priority masks for each priority level. Mask_array(3) for priority level 3, mask_array(2) for priority level 2 and so on.

intermediateGet : This is the get value that has not yet been used and will enter the Get FIFO. The FSM will eventually assign intermediateGet to get when it is ready to take them out of the FIFO.

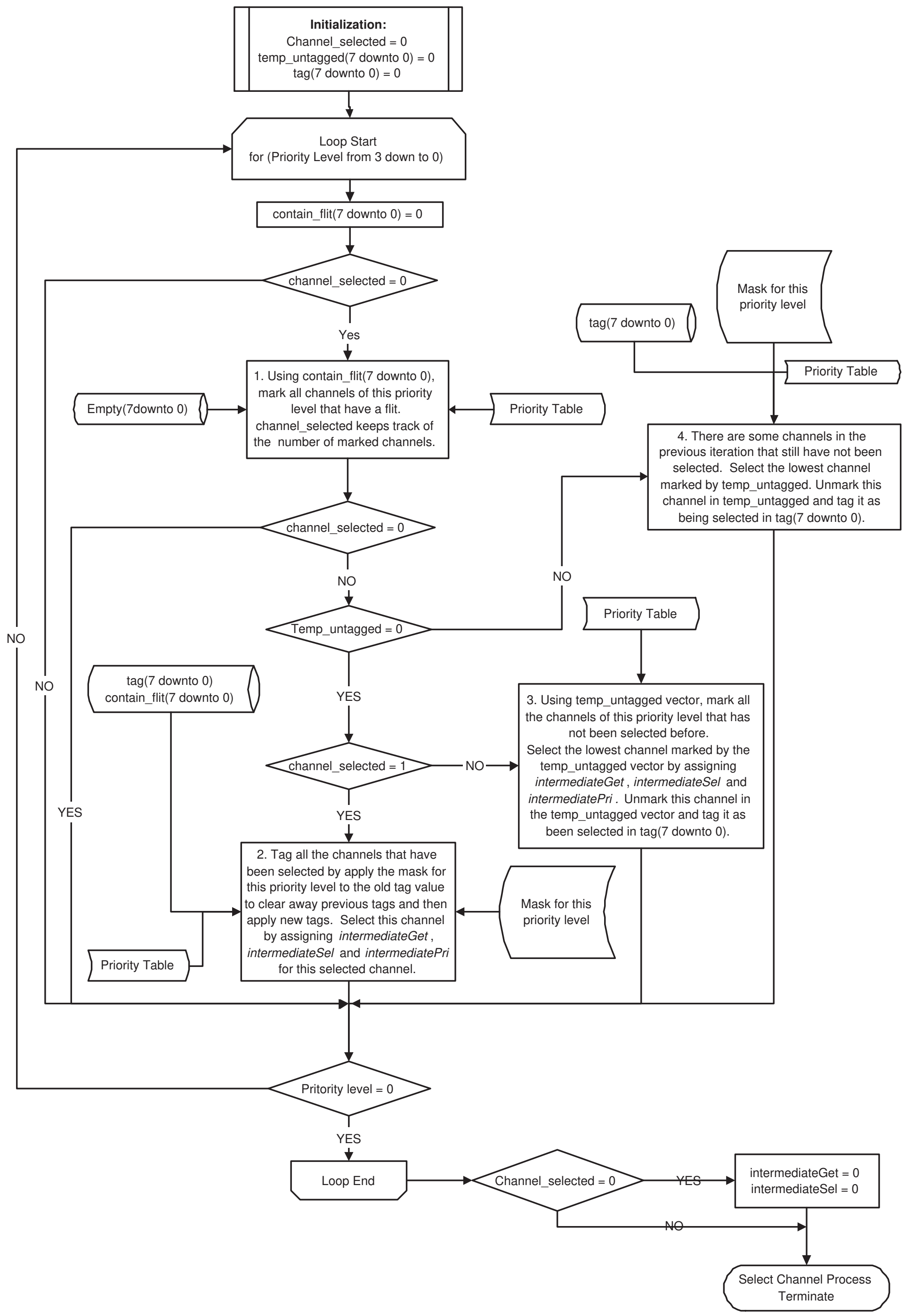

Figure 5.2: Flow chart diagram for Select Channel Process shown in Figure 5.1

intermediateSel : This is the corresponding inChSel signal and will enter the Sel FIFO.

intermediatePri : This is the associated priority of the channel indicated by

intermediateSel. This signal will eventually be used for packet decoding in the Packet Decode process shown in Figure 5.1.

We start the process by first initializing channel_selected, tag, and temp_untagged. For each priority level, we perform the series of tasks as illustrated in Figure 5.2. We will now simply clarify the numbered task boxes in the flow chart.

1. For this priority level, given the empty signal and the priority table, first mark in contain_flit all channels that have a flit, and at the end use channel_selected to keep track of the number of channels marked.

2. If there is one channel that has been marked in this priority level, we simply select this channel by assigning intermediateGet, intermediate-Sel and intermediatePri, and tag it as being selected. However, before tagging the channel, we must clear old tags of this priority and not touch tags for other priority levels. To do so, we apply the mask for this priority level to the old tag vector by logically ANDing the old tag with the negate of mask_array(pri), where pri is the current priority level. We then tag the channel currently selected by logically ORing the previous ANDing result with contain_flit, which only has the one bit set since there is only one channel selected.

3. If there are more than one channel that have been marked in this priority level, we must first make sure that we select a marked channel that contains data but has not been serviced before. To do so we logically AND contain_flit with the negate of tag and store the result in temp_untagged as shown in Table 5.1. Then, we choose a channel marked in temp_untagged from the lower number end and select it as the next channel to service. We also have to remember to unmark this channel in temp_untagged and tag it as serviced using the same procedure as described above with the priority mask.

4. It is possible that there are more channels marked in temp_untagged that has not been serviced. Therefore, we choose another channel marked in the temp_untagged vector from the lower end, unmark it, and tag it as serviced the same way as described above.

Get FIFO and Sel FIFO

The Get FIFO,get_queue, stores the get signals and the Sel FIFO, sel_queue, stores the corresponding inChSel signals. The FIFOs are implemented in a circular fashion. When a new decision arrives, it is stored in the lower end of

contain_flit	tag	$\overline{tag}$	temp_untagged
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

Table 5.1: Only when the channel contains a flit (contain_flit = 1) and has not been tagged before ($\overline{tag} = 1$) do we want to get flit from this channel. Temp_untagged marks all the channels of the same priority that has not been selected before.

the queue and work its way upwards, as shown in Figure5.3. Since there are only 8 virtual channels, there will never be more than 8 different channels to pick and hence the depth of these queues need only to be 8. When the insert counter has reached the end of the queue, it will wrap around and start inserting in the beginning of the queue. However, it will only override the old channel in the beginning of the queue when the channels has been taking out of the queue and serviced, which is indicated by `take_tag`.

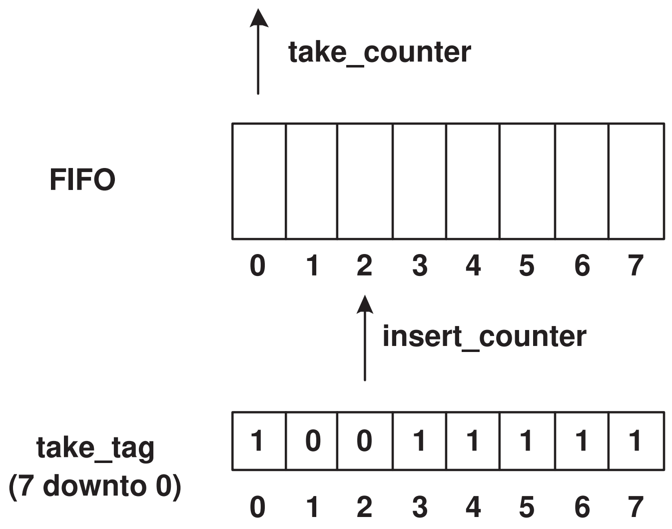

Figure 5.3: First In First Out Queues. Take_tag marks all the channels that have already been taken out of the queue by the Finite State Machine. Only when a channel has been taken out of the queue can it be overwritten with a new channel.

Priority Scheduler Finite State Machine

The Finite State Machine is to control the selecting of the channels such that a complete packet can be received before another channel is picked. Figure 5.4 shows the state diagram for the Priority Scheduler Finite State Machine.

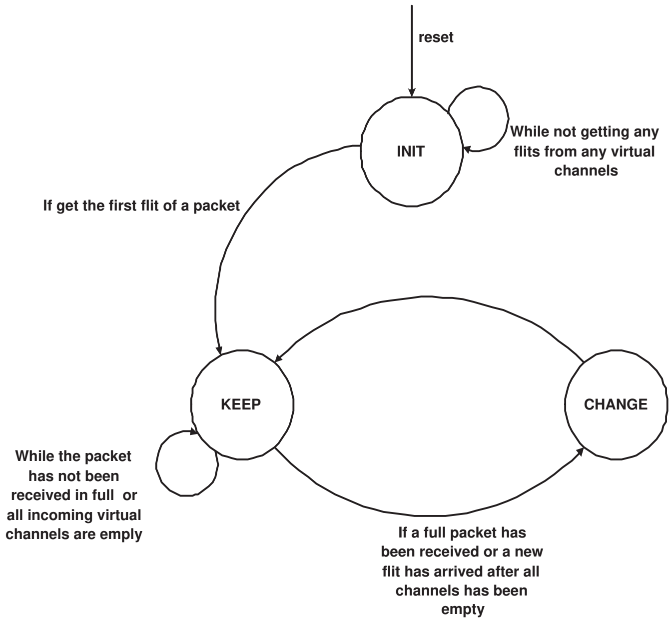

Figure 5.4: Priority scheduler finite state machine for incoming virtual channels.

At reset, the Priority Scheduler FSM enters the INIT state. If no packet has arrived yet, intermediateGet will be zero. If this is the case, we remain at INIT state and we don't assert the get signal. If intermediateGet is not zero, then we have got our first channel chosen after reset. Since this is the first channel to choose after reset, there is no need to put this no need to insert this channel into the FIFO, we simply select this channel directly. After this channel has been selected, we enter into the KEEP state in the next clock cycle.

At the KEEP state, we want to keep selecting the same channel until the full packet has been received. Therefore, while packet has not been received, packet_received equals 0, we remain in the KEEP state where we do not touch the previously assigned get and inChSel signals. Packet_received is asserted by the Packet Assemble process discussed later in section 5.2.1. If packet_received equals 1, we wish to select a new channel and we enter the CHANGE state in the next clock cycle. If when during operation no channel has a flit, we will remain in the KEEP state. When a flit arrives at a channel, we will then enter the CHANGE state.

At the CHANGE state, we take the an interediateGet signal from the front of the Get FIFO and an intermediateSel signal from the front of the Sel FIFO and assign them to get and inChSel respectively. When the take counter reaches the end of the FIFO, we wrap around to the beginning of the FIFO again. After the get and inChSel signals has been assigned, we return

	Packet Type	Bit Encoding
1	Setup Request	000
2	Setup Response	001
3	Teardown Request	010
4	Teardown Response	011
5	BE Request	100
6	BE Response	101
7	GS Request	110
8	GS Response	111

Table 5.2: Packet Type Encoding for DTU NoC

to the KEEP state in the next clock cycle.

Flit Collection and Packet Reassembly

Before decoding the packet, we must first receive all the flits of the packet and reassemble them into the original message. The Packet Assemble process does this job. Every clock cycle, it stores the inFlit into a packet array and checks if it is the last flit. InFlit(16) is a control bit- 1 indicates the last flit of the packet and 0 indicates all other in between flits. When the last flit has been received, Packet Assemble process will assert packet_received signal to let the FSM know that it is allowed to switch to another channel.

Recognizing Packet Types and Content Extraction

After a full packet has been assembled the Packet Decode process, given the priority of the channel that this packet came from, knows whether this packet is a BE packet or GS packet. It uses this information to determine which type of BE packet or GS packet it is by checking MCmd, and various bits in the packet header. The exact information has been presented in Section 4.4. The Packet Decode process encodes this packet according to the packet type encoding presented in Table 5.2 and assigns the packet type to packet_decode. Aside from recognizing and encoding the packet type, the Packet Decode process also assigns the correct values for all other OCP signals to be sent to the ReqE2E unit or RespE2E unit.

Strobing signals to ReqE2E or RespE2E

The Signal Validation process is responsible for sending the strobe signals response_arrived and request_arrived to the RespE2E unit and the RespE2E unit when a response packet or a request packet arrives respectively. It only asserts these strobe signals for one clock cycle as it is required for the OCP slave control in the end-to-end flow control units.

5.2.2 Request End-to-End Flow Control Unit

The Request End-to-End Flow Control Unit (ReqE2E) contains an OCP slave controller, which is comprised of two finite state machines: one for receiving requests from the master core and the other for responding to the requests. The ReqE2E is also responsible for recognizing the request type, sending out the correct set of signals to the Encapsulation Unit corresponding to each packet type, and asserting the control signals for both the Connection ID Table and the Route Lookup Table. A block diagram of the ReqE2E Unit is shown in Figure 5.5. We will begin our discussion of the ReqE2E Unit by starting with the OCP slave control and then packet recognition and encoding.

OCP Slave Control Unit

The OCP Slave Control Unit is implemented using two separate finite state machines: a request OCP controller for receiving requests and a response OCP controller for receiving responses to the requests. The reason this is so is that the latency of the network is unknown and the master core cannot be required to wait for the response packet to return before it can issue a second command. If done so, it would severely inhibit the speed at which the core can perform its task.

Having two separate controllers allows the master core to continue issuing commands as soon as the commands are packetized and sent to the output queues. The responses do not necessarily arrive in the order that the requests are sent. It is for the master core to rearrange the responses after they have been received. The NA will provide the master core with the Slave core's address so that the responses can be paired to its origin. An example of OCP communication timing diagram implemented by the slave NA is shown in Figure 3.3 on page 35. At point A the request OCP controller raises the SCmdAccept signal because at that point the request has been packetized and sent to the output queue. At point B, a few clock cycles later, the response packet has arrived and the response OCP controller presents the response to the OCP interface.

Request OCP Controller

Figure 5.6 shows the state diagram for the request OCP controller.

On reset, the request OCP FSM enters the IDLE state, at which both the SCmdAccept signal and the request_phase signal are de-asserted. Asserting the SCmdAccept signal tells the master core that the request has been accepted and allows the master core to issue a new request. Asserting the request_phase signal tells the Encapsulation Unit that a valid request message is ready to be packetized.

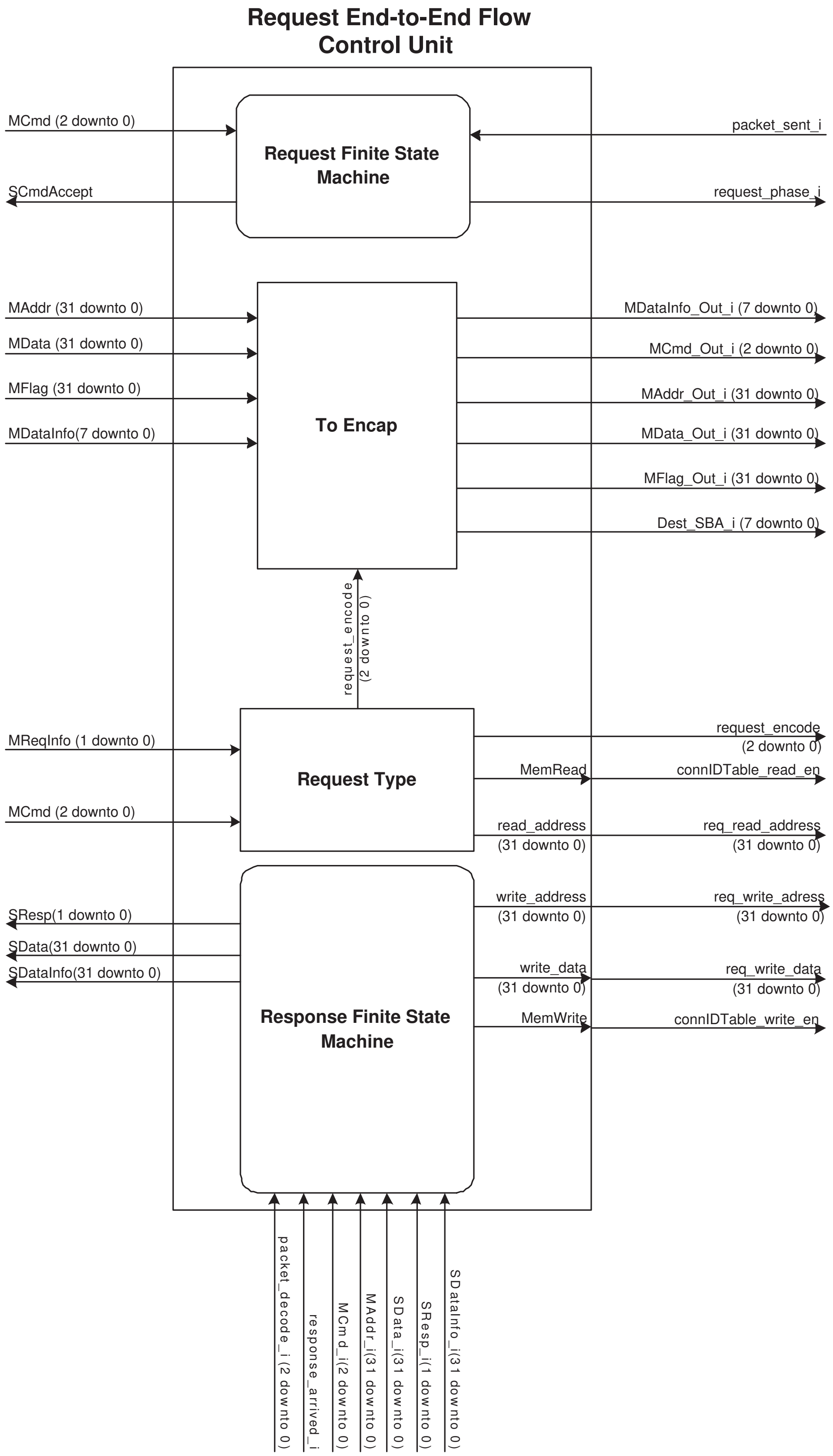

Figure 5.5: Block diagram of processes and signals within the ReqE2E Unit.

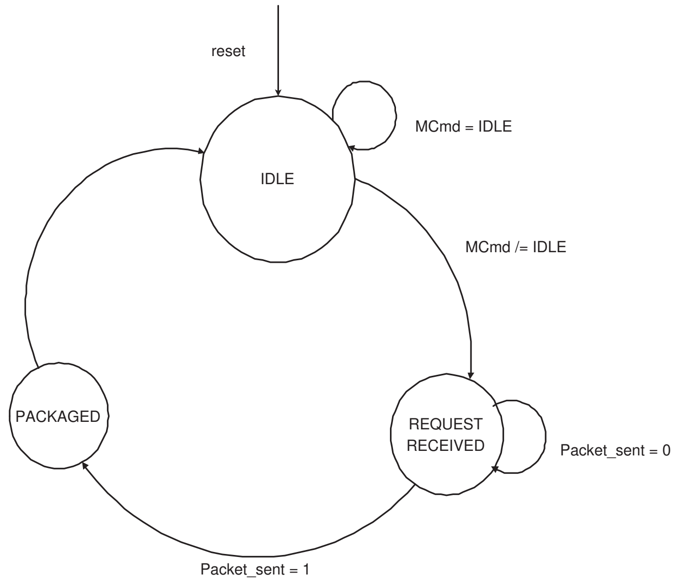

Figure 5.6: Finite State Machine for receiving requests

In the IDLE state, if MCmd is any thing other than *IDLE* or *"000"* then a request has arrived and OCP FSM enters the REQUEST RECEIVED state. Otherwise, a request has not been received and OCP FSM remains in the IDLE state.

It the REQUEST RECEIVED state, the request_phase signal is asserted, telling the Encapsulation Unit to packetize the request. The SCmdAccept signal is kept de-asserted. The OCP FSM stays at the REQUEST RECEIVED state until an acknowledge signal, packet_sent_i, from the Encapsulation Unit has been received, indicating that the request has been packetized and sent off to one of the output queues. When packet_sent_i has been detected to be asserted, OCP FSM enters the PACKAGED state in the next clock cycle.

At the PACKAGED state, the SCmdAccept signal is asserted, allowing the master core to issue a new request. The request_phase signal is then de-asserted to acknowledge to the Encapsulation Unit that a new request is no longer valid. At the rising edge of the next clock cycle the OCP FSM returns to the IDLE state.

Response OCP Controller

Figure 5.7 shows the state diagram for the response OCP controller.

On reset, the response OCP controller enters the IDLE state, where all the slave signals to the master core SResp, SData, and SDataInfo are invalid. If the response_arrived_i signal is asserted, then the RESPONSE RECEIVED state is entered in the next clock cycle. Otherwise, the OCP controller remains in the IDLE state. The response_arrived_i signal is the

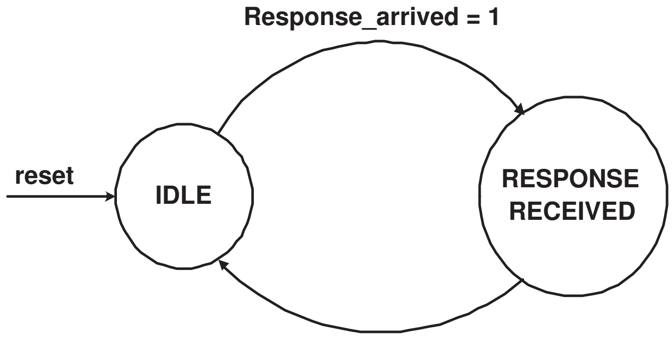

Figure 5.7: Finite State Machine for receiving responses

strobe signal from the Decapsulation Unit, informing the response OCP controller that a valid response packet has just arrived.

In the RESPONSE RECEIVED state, the OCP controller decides what to do with the received response depending on the type of response it is, which is given by the packet_decode_i signal from the Decapsulation Unit. The encoding of the response packets is shown in Table 5.2.

- If the response is a Setup Response packet and the setup was successful, the injection channel IDs for outgoing and incoming virtual channels is written to the Connection ID Table at the address given by MAddr_In, and the core know is made know of the response result by assigning SResp_i to SResp and giving the master core a connection ID using the SData signal which is assigned to MAddr_In. If the setup has failed, nothing is written to the Connection ID Table and the core is informed of the setup failure through SResp. The core can distinguish this setup response to other responses by the value of SDataInfo, which contains the address of the slave core which this connection was set up to.

- If the response is a Teardown Response packet and the teardown was successful, then the Connection ID Table entry given by MAddr_In is reset, and the core will know the successful teardown using SResp and the connection ID that was torn down using SData. If the teardown was unsuccessful, nothing will change in the Connection ID Table and the core will know the teardown failure using SResp and SData.

- If the response is a BE response, the result will be passed to the master core using SResp and SData if the response is to a read and just using SResp if the response is to a write.

- If the response is a GS response, the same is done as for a BE response.

At the rising edge of the next clock cycle the OCP controller returns back to the IDLE state. Therefore, the response phase is always one clock cycle.

Packet Recognition and Encoding

The Request Type process on Figure 5.5 is used for determine which type of request the master is presenting by using the MReqInfo signal and the MCmd signal. The encoding is shown in Table 5.3.

		MCmd	
		010	001
MReqInfo	00	BE request	BE request
	01	GS setup request	Not Used
	10	GS request	GS request
	11	Not Used	GS teardown request

Table 5.3: MReqInfo and MCmd Signal Encoding for different request types

Using the MReqInfo signal and the MCmd signal, the Request Type process assigns the packet_encode_i signal according to the Packet Type encoding shown in Table 5.2.

The To_Encap process, given the packet_encode_i signal, sends according the type of request packet the correct values for MCmd_Out_i, MAddr_Out_i, MData_Out_i, MFlag_Out_i, MDataInfo_Out_i and Dest_SBA_i to the Encapsulation Unit for packaging.

5.2.3 Connection ID Table

The Connection ID Table is simply a local memory in the NA which is globally mapped to the DTU NoC memory space as specified in [8]. It is used to store incoming and outgoing injection IDs for guaranteed service setup. When the network controller responds to a connection setup, it sends a GS setup response packet to the master and/or slave core's NA and directly writes the incoming and outgoing virtual channels that has been setup for this particular guaranteed connection into the Connection ID Table. When the connection has been torn down, the network controller will then send a GS teardown response packet to the NA and directly resets the Connection ID Table for that particular connection.

An advantage of having the Connection ID Table globally memory mapped is that the network controller can directly control the contents of the table and the configuration of GS connections is made writable globally. This makes the management of GS connections much more flexible and manageable centrally by the network controller.

The design of the Connection ID Table is very straight forward. Since a GS response packet from the network controller can arrive at the same time as GS request from the master core, the Connection ID table needs to be able to handle both read and write at the same time. At reset, all data in the memory is cleared to zero. At the rising edge of the clock, if connIDTable_read_en is high, the Connection ID table will take the lower 4 bits of the req_read_address as the addressing index to the table. The stored data, InjID_i, will be then read out. The data length of the memory is 32 bits: bits 4 to 7 indicates the outgoing virtual channel, and bits 0 to 3 indicates the incoming virtual channel. The reason that only the lower 4 bits of the address are used is that the Connection ID Table has only 16 entries in total. If connIDTable_write_en is high at the rising edge of the clock, then the lower 4 bits of the req_write_address will be used as addressing index and req_write_data will be written to the table.

5.2.4 Route Lookup Table

The Route Lookup Table is another local memory in the NA. It stores the route paths to other cores in the NoC. This route path is needed as part of the BE packet header, since all packets are source-base routed. This means that all the routing information is stored in the routepath which indicates to the routing nodes where to route the packet at each hop. The Route Lookup Table is not globally memory mapped and can not be addressed by other cores. The table is configured and the entries are set at NA instantiation time.

[8] does not specify that the Route Lookup Table should be globally memory mapped. However, it would be beneficial to do so because the network traffic congestion is dynamic. When a particular link or path is overly congested, it may be beneficial to route paths to other paths. This requires that the route path be changed. Not having the Route Lookup Table globally memory mapped removes the possibility of changing the route path from one core to another. Thus, makes the traffic routes static. On the other hand, since MANGO implements virtual channels, the congestion problem may be alleviated and it would not serve too beneficial to have the route path assignment dynamic.

The Route Lookup Table simply outputs the routepath given an IP core System Base Address (SBA).

5.2.5 Encapsulation Unit

The responsibilities of the Encapsulation Unit are as follows.

- Receive requests from the Request End-to-End Flow Control Unit (ReqE2E), as well as inputs from the Connection ID Table and Route Lookup Table.
- Given the packet type through the packet_encode_i signal, packetize the request according to the packet format introduced in Section 4.4 and separate the packet into flits, each of 17 bits in length.

- Decide which one of the output virtual channel queues to send the packetized data. If the packet type was a GS request or response packet, the output virtual channel number will be given from the Connection ID Table. If the packet type is a BE packet, the Encapsulation Unit must select one of the output virtual channels assigned to BE priority.
- Send the packet to the selected channel queue by communicating with the Queue Control Unit.

Figure 5.8 shows the state diagram for the operation of the Encapsulation Unit.

Ot reset, the FSM enters the INIT state, where outChSel signal is initialized to "000", packet_sent_i signal is de-asserted, and the outgoing packet is empty. At the rising edge of the next clock cycle the FSM directly enters the IDLE state. INIT state initializes all control signals and internal signals and will not be returned unless reset is asserted.

In IDLE state, the packet_sent_i signal is de-asserted because a new request message has not arrived. The FSM checks if a request has arrived from the Request End-to-End Flow Control Unit (ReqE2E) by seeing if request_phase signal is asserted or not. If it is asserted, a new request has arrived and the FSM changes to the PACKETIZE state at the rising edge of the next clock cycle. Otherwise, a new request message has not been received and the FSM remains at the IDLE state.

It PACKETIZE state, the packet_sent_i signal is kept de-asserted. The FSM checks which type of request packet this message is via the packet_encode_i signal in order to decide how to packetize and segment this request message. There are four different possible types of request messages: GS setup request, GS teardown request, BE request, and GS request. The FSM packetizes the message and arrange its contents exactly as it is presented in Section 4.4. After the request message is packetized, the FSM determines which one of the output queues to send the packet. If the packet is a GS request packet, the FSM will be given the output queue number from the Connection ID Table via the InjID signal. Otherwise, the packet is a BE packet and the Encapsulation Unit must select from one of the available BE virtual channels by checking and marking all the BE channels output queues with its hold signal de-asserted. Like the Decapsulation Unit, there is also a priority table for the outgoing virtual channels. This table is also statically assigned, and from it the Encapsulation Unit knows the priority of the outgoing virtual channels. An BE priority outgoing virtual channel will be chosen to send the BE packet based on whether the channel output queue is available or not. This is done by first marking all the BE channels that are available, then selecting from the marked channels one channel starting from lower number virtual channel to higher number. For both GS packet and BE packet, if the hold signal for the selected channel is asserted, the FSM enter the AWAIT state

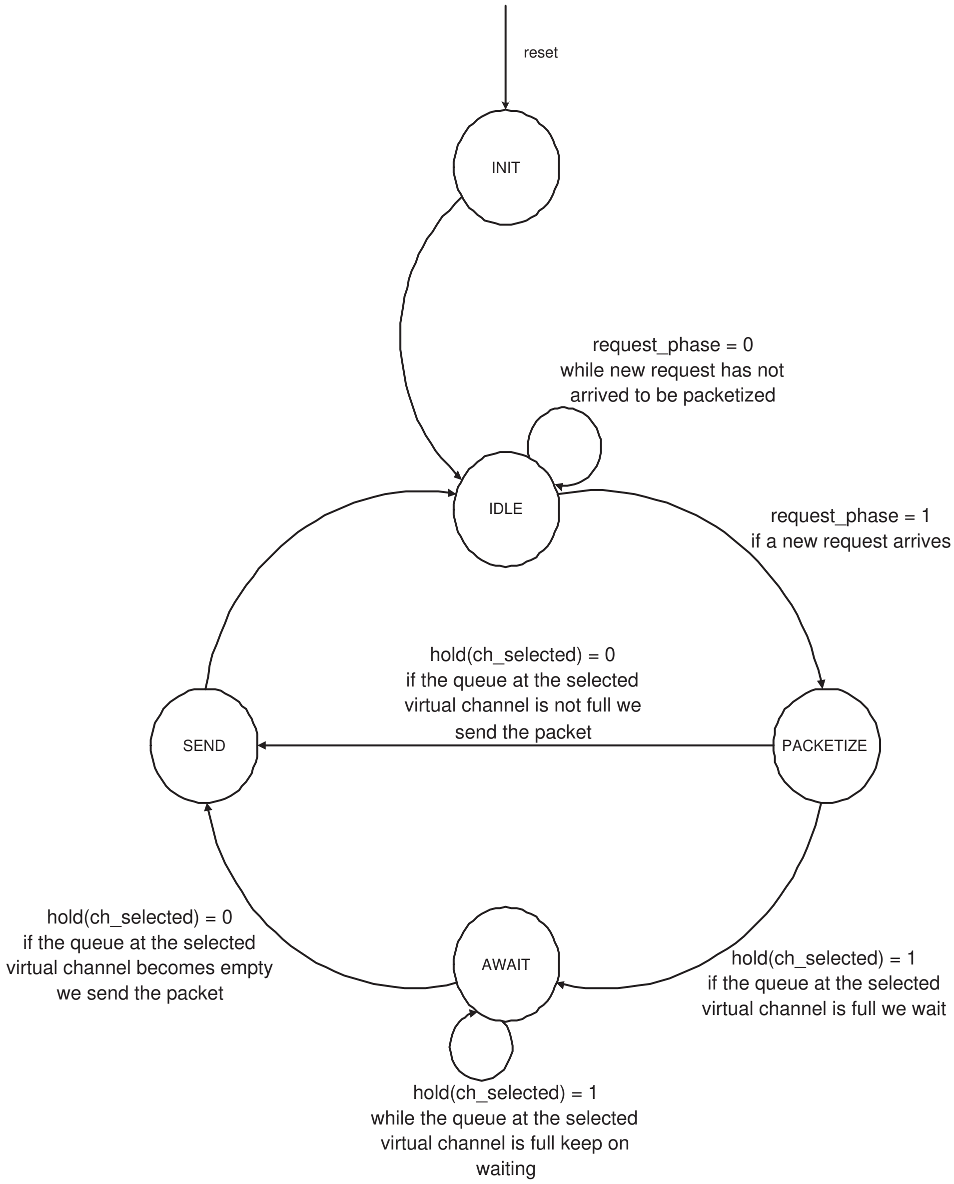

Figure 5.8: Encapsulation Unit Finite State Machine for sending out packets to the output queues.

in the next clock cycle. Otherwise, the selected output queue is available, and the FSM enters the SEND state in the next clock cycle.

In SEND state, the FSM sets outChSel signal from the chosen channel in the PACKETIZE state, asserts packet_sent_i signal, and sends the packetized data via packet_i signal. At the rising edge of next clock cycle, the FSM returns back to the IDLE state.

In AWAIT state, the FSM continuously checks if the hold signal for the selected channel is de-asserted or not. If it is, the channel output queue has become available and the FSM can enter the SEND state in the next clock cycle. Otherwise, the selected channel output queue is still busy and the FSM remains in the AWAIT state. While at the AWAIT state the packet_sent_i signal is kept de-asserted.

5.2.6 Output Queue Control

The Queue Control is intended to interface between the 8 output queues and the Encapsulation Unit. For example, the Encapsulation Unit sets outChSel signal to bits *"101"* telling the Queue Control that it wants to send the outgoing packet to output queue number 5. The Queue Control checks whether output queue number 5 is ready to receive a new packet or not by checking the signal ready at position 5. If ready(5) is not asserted or not ready to receive a new packet, then the Queue Control will inform the Encapsulation Unit to hold the new packet by asserting the hold(5) to *1*. Otherwise, if queue number 5 is ready to receive a new packet, the Queue Control will assert load(5) to *1*; hence enabling the queue and storing the new packet into queue number 5. Figure 5.9 shows the state diagram for the Queue Control.

On reset, the FSM enters the INIT state, where hold signal for all 8 channels and load signal for all 8 queues are de-asserted. The INIT state is there to initialize the output signals at reset. At the rising edge of the next clock cycle, the FSM enters the IDLE state.

In the IDLE state, the packet_sent_i request signal from the Encapsulation Unit is checked for assertion. The load signal for all 8 queues are de-asserted. However, if any one of the ready signal from the 8 output queues are asserted, meaning that the queue has become ready to receive a new packet again, then the hold signal for that queue will be de-asserted. If packet_sent_i is asserted and the ready signal at the position given by outChSel is also asserted, then the queue which the Encapsulation Unit wants to send the new packet to is available, and the FSM enters the LOAD QUEUE state at the rising edge of the next clock cycle. Otherwise, if packet_sent_i is asserted but the ready signal at the position given by outChSel is not asserted, then there is a new packet that needs to be send to a queue which is not yet available. If this is so, the FSM enters the AWAIT state at the rising edge of the next clock cycle. If neither of these cases are true, then packet_sent is not asserted, which indicates no new packet is ready to be

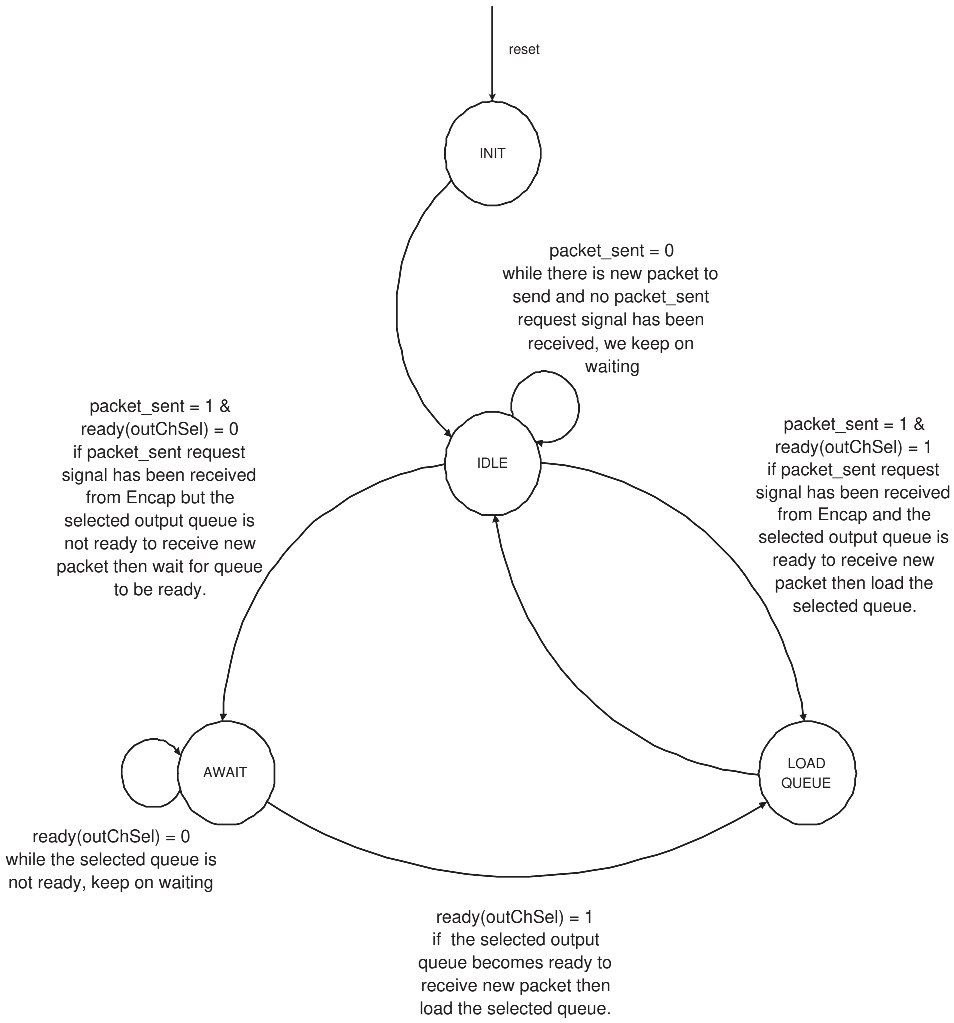

Figure 5.9: Output Queue Control Unit Finite State Machine for loading packets to the output queues.

send. In this case, the FSM stays at the IDLE state and keep on waiting for the assertion of the packet_sent_i request signal.

In the LOAD QUEUE state, the FSM enable the output queue indicated by outChSel by asserting the load signal for that queue. At the same time it sets the hold signal for the queue that just have been loaded because once it's loaded it will not become available again for a new packet until the current packet has been sent. The FSM also checks if any one of the 8 queues has become available again and de-asserts the hold signals for those queues if so. At the rising edge of the next clock, the FSM returns to the IDLE state.

In the AWAIT state, the FSM continues to check if the ready signal for the selected queue is asserted or not. If so, it go to the LOAD QUEUE state. If not, it stay at the AWAIT state. Here it also checks if any one of the 8 queues has become available again and de-asserts the hold signals for those queues if so.

5.2.7 Output Queues

There are 8 instantiations of the output queue: one for each outgoing virtual channel. The output queues are used to hold packets which are intended to be sent to the queue's corresponding virtual channel. The queues communicate with the Output Port control Unit on the SYNC component (see Fig 4.11) complying to the communication protocol show in Figure 4.13 until all the flits of the packet have been sent. There is an output queue for each outgoing virtual channel because in order to guarantee the full use of all virtual channels, a packet intended for one channel should not block a packet intended for another channel. Having a queue for each virtual channel allow the full potential of utilizing all 8 virtual channels at once.

For each output queue, there are three input: the load bit, the queue enable bit, the flit_array signal for the incoming packet, and a full bit coming from the SYNC component's Output Port Control Unit. The queue outputs an outgoing flit, outFlit, to the outgoing virtual channel, a ready bit to the Queue Control Unit, and a put bit to the SYNC component's Output Port Control Unit.

At the rising edge of each clock if the load bit is asserted, the queue are loaded. Otherwise, the queues have already been loaded and the queue checks if its corresponding virtual channel is full or not. If not full, the queue checks if the flit to be sent is the last flit of the packet. It sends off the flit and asserts the put bit. When the last flit of the packet has been sent, it asserts the ready bit, indicating that the queue is available again to receive a new packet. Otherwise if the channel is full, it waits by de-asserting the put bit and the ready bit.

5.3 Concluding Remarks

We have now concluded our discussion of the implementation details of the network adapter. There is much more that could be done to add more functionality to the NA. We will present some of these additions in Chapter 8 and discuss some of their design issues.

Chapter 6

Test of the Slave Network Adapter

Contents

Testing of the slave NA's implementation is essential for validating its correctness. The NA should comply to the OCP specification [7], the DTU OCP Specification [10], and the Network Interface Specification [9]. The tests are designed to show that the behavior of the slave NA follows the description in Chapter 4 and that it is able to perform all of its required functions. This chapter first discusses how the testing of the of slave NA was conducted. Second, the chapter will list the major test cases and what they test. Last, the test results will be discussed.

6.1 Test Method

There are altogether seven components on the slave NA and some components consists of several modules. A bottom-up testing approach was used to debug and verify each component. Figure 6.1 shows a flow chart diagram of the testing procedure that was used for each component. After designing the component and its interfaces, the component was implemented in VHDL.

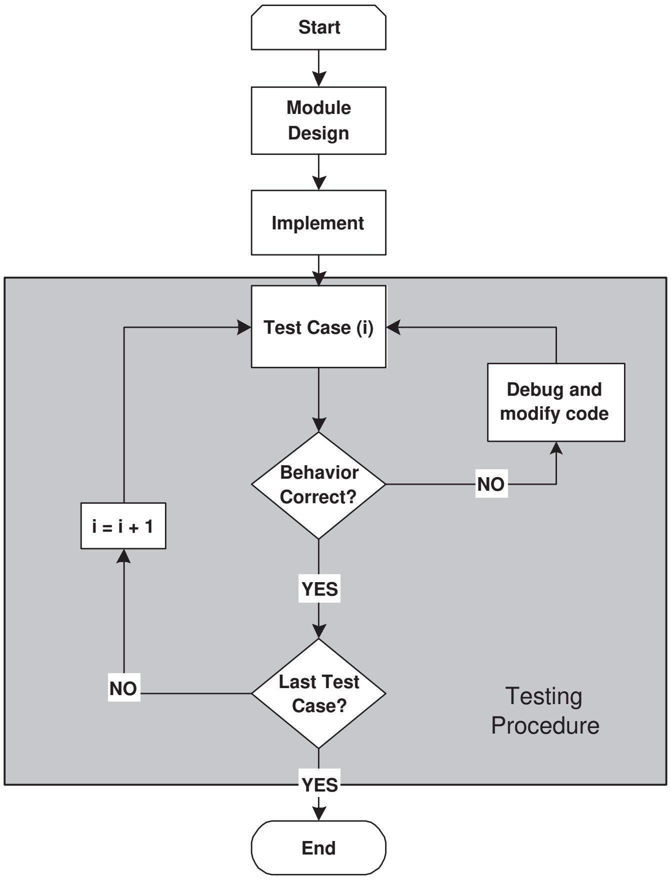

Figure 6.1: Testing procedure for Slave NA design.

Test cases were designed for the component of interest. The test began by applying the first test case to the test bench. If the behavior of the component was as expected, the next test case was applied and added to the test bench. If the behavior of the component was not as expected, there existed some bug and debugging commenced until the component passed this test case. This iteration of testing continued until all test cases were used and the component's behavior satisfied all cases, at that point the testing was complete.

After testing each component individually, components are grouped together for testing as shown in Figure 6.2. The figures shows the components in hierarchical levels. Those of the same level are in the same colors. There are 6 test groups in total. They are circled and labelled in the figure. Each group test goes through the same test procedure as the tests for individual components, as shown in figure 6.1. The final slave NA, or test group 6, incorporates all components and verifies the proper operation of the complete slave network adapter.

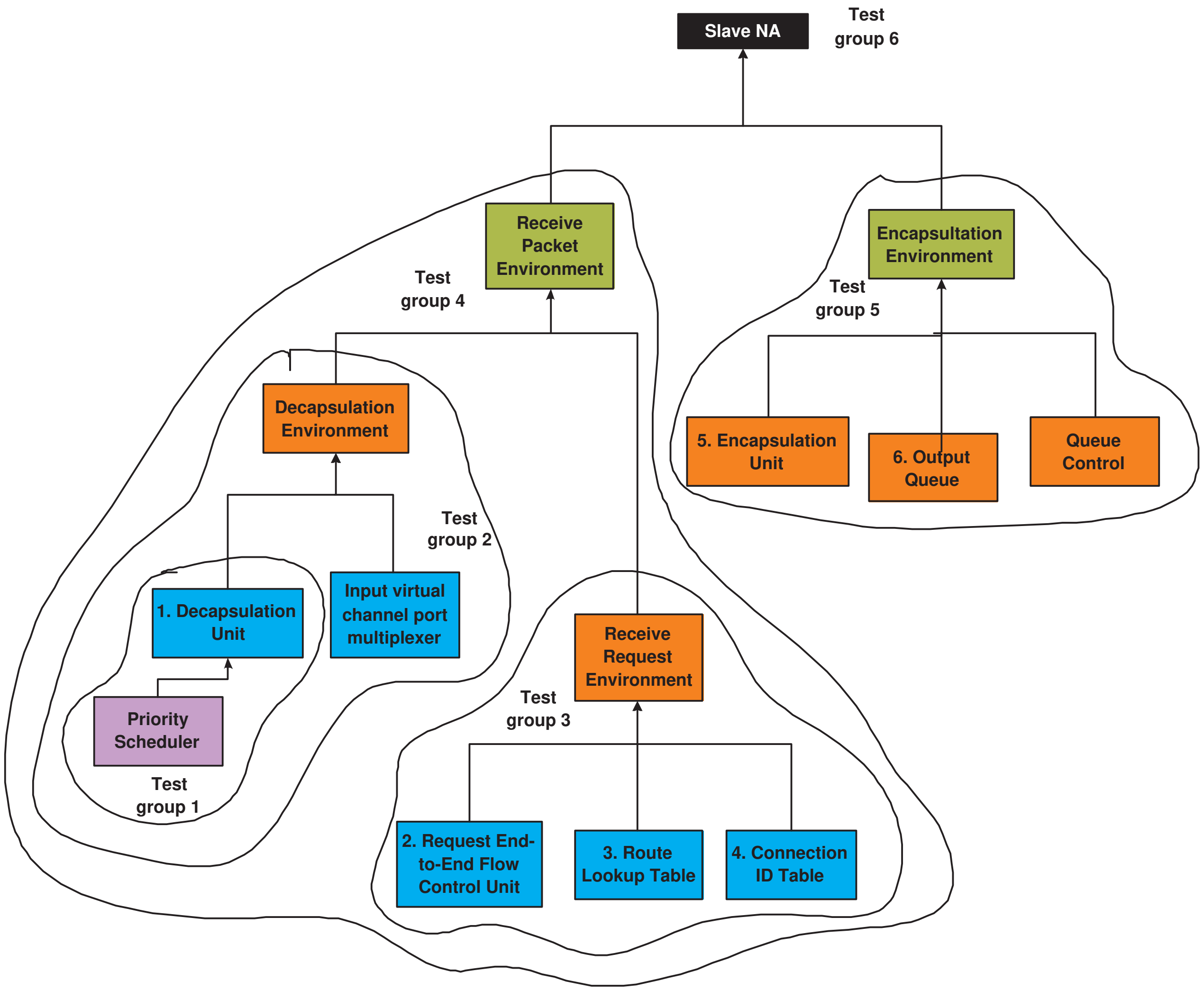

Figure 6.2: Testing hierarchy.

In the next section, we will examine four major test cases and see what each test case test.

6.2 Test Cases

Figure 6.2 lists 6 test groups and 8 individual component tests, totalling 14 test benches. It is not interesting to go into details about all these 14 test benches. Therefore, we will select four important cases for closer examination.

6.2.1 Priority Scheduler Testing

The priority scheduler test checks for the following cases:

- When two packet arrive at the same time, one packet has a higher priority than the other, the higher priority packet will be selected first.
- When a lower priority channel packet arrives before a higher priority packet, the lower priority channel will be selected first.
- If a high priority packet arrives while a low priority channel is in the process of transmitting flits, the scheduler will wait until the low priority channel finishes transmitting before selects the high priority channel.
- When two packet of the same priority arrive at the same time and one of the channel has just been selected, then the packet on the other unselected channel will be chosen.

6.2.2 Decapsulation Testing

The decapsulation testing (test group 2) tests the following cases:

- After the priority scheduler has selected a channel, the flits from that channel are stored and assembled correctly.
- The received packet is decoded correctly, and all 8 types of packets in the NoC can be recognized.
- The content of the packets is extracted and assigned to the correct internal signals on the other end of the Decapsulation Unit interface to other NA internal components.

6.2.3 Receive Request Testing

The receive request testing (test group 3) tests the following cases:

- All types of requests are interpreted and encoded correctly and the correct payload for each type of request is sent to the Encapsulation Unit.
- When a GS response packet arrives, the Connection ID Table is enabled and the correct address and data value is sent.
- The request and response phases comply to the OCP specification.

6.2.4 Encapsulation Testing

The Encapsulation Testing (test group 5) tests the following cases:

- The request message from the ReqE2E unit is packetized correctly depending on the packet type.
- For GS request packets, the packet is sent to the virtual channel specified in the Connection ID Table.
- For BE packets, the packet is sent to one of the available BE output queues. If no output queue is available, the *packet_ sent* signal to the ReqE2E unit will be de-asserted.
- If one the output virtual channel is congested the *hold* signal for that channel will be high and no more packets will be injected into that channel until the congestion is resolved.

6.3 Test Results

The 14 test benches exercised all specified functionalities of the slave network adapter. The result of the testing showed that the current implementation of the slave NA has passed all tests in each of the 14 test benches. The slave NA is performing as expected.

Chapter 7

Results Discussion and Performance Estimate

Contents

How the network adapter performs in term of its clock frequency, area, power usage, throughput, and latency is very important to the overall design of MANGO and to the feasibility of using this network adapter. The performance of the network adapter is evaluated in terms of its clock frequency, throughput, and latency. The cost is evaluated in terms of its area usage and power consumption. We begin this chapter by first discussing the results achieved in this project in terms of its performance and cost estimate. Then some suggestions will be provided for possible cost saving strategies for future NA designs.

7.1 Performance

In order to provide a good and useful performance and cost estimate for the slave NA, the behavioral VHDL model of the slave NA was slightly modified to allow synthesis and was synthesized in Synopsys Design compiler using a 0.18 μm cell library [12]. The resulting synthesized VHDL code is not a working VHDL description of the slave NA because the modifications that was made for synthesis have introduced some errors in the NA's behavior. Nevertheless, it is a reliable source for performance analysis and cost estimate.

Speed

The synthesis was done optimizing for high speed. The result achieved was that the slave NA can run at 741 MHz with a clock period of 1.35 ns. The area estimate generated in Section 7.2.1 is based on the high speed synthesis result.

Throughput

The NA is a bridge between the core and the NoC. Therefore, throughput for the NA can be in two directions: the forward direction, from the core to the NoC, and the reverse direction, from the Noc to the core. Throughput for the slave NA is defined to be:

$$\text{throughput} = \frac{n}{c}$$

where n is the number of requests that the slave NA receives from the master core and c is the number of cycles. The VHDL behavioral simulation of the slave NA shows that the throughput for the current implementation in the forward direction is:

$$\text{throughput\_fw} = \frac{1}{4} = 0.25 \quad \text{requests/cycle}$$

Therefore, if no congestion occurs at the output virtual channels, the slave NA can receives 1 request every 4 cycles. The throughput in the reverse direction is directly related to the length of the packet being received. Since the NA can (in the absence of congestion) receive packets as fast as they arrive, the only thing that limits the throughput is the link speed. Therefore the throughput in the reverse direction can be characterized as:

$$\text{throughput\_rv} = \frac{1}{m}$$

where m is the number of flits within the packet.

The purpose of the output queues is to increase the throughput of the NA. When a packet is sent to an output queue, the slave NA is free to receive the next request from the master core. Therefore, if the queue depth increases, the throughput would increase. However, this is done at the expense in area. Naturally, the bigger the storage space in the output queues, the more space it will use up in the NA.

The network adapter receives and sends one flit per cycle at 741MHz, and each flit is 17 bits. Therefore, the bandwidth that the network adapter supports is 1.57 GBytes/s.

Latency

Latency in the forward direction is defined as the time from when the request is presented at the OCP interface to the time when the last flit of the packet is set out of the NA. Latency in the reverse direction is defined as the time when the first flit of the packet enters the NA until the result is presented to the core at the OCP interface. The simulation results shows that in the absence of network congestion the latency for the forward is:

$$\text{latency\_fw} = m + 2 \quad \text{cycles}$$

and the latency for the reverse direction is:

$$\text{latency\_rv} = m + 1 \quad \text{cycles}$$

where m is the number of flits within the packet.

From the latency equation above, the latency is linearly dependent on the length of the packet. Without network congestion the slave NA is able to packetize a request and send the packet to the network adding an additional 2 cycles. It adds only 1 cycle in the reverse direction.

7.2 Cost Estimate

7.2.1 Area Usage

The size of the NA on a chip is important in analyzing how much overhead space the NA will occupy in comparison to both the size of the IP cores and in comparison to the complete NoC. If the size of the NA is too big compared to the IP cores, then it may be too expensive to implement all the functions of the NA, and may be more reasonable to cut down its features in order to decrease its area. Table 7.1 presents an area estimate of the slave NA implemented in this project using the μm technology. In future networks, smaller cell libraries will be used for synthesis. National Technology Roadmap for Semiconductors estimates that 0.05 μm technology will be attainable for future SoCs [13].

The area estimate shows that the top three modules that occupy the greatest amount of space are: the output queues, followed by the Connection ID Table, followed by the Decapsulation Unit.

The number of output queues is dependent upon the number of virtual channels supported. The current implementation implements eight output and input virtual channels. Each output queue occupies approximately 5.8% of the total area. Therefore a tradeoff exists between the size of the NA and the number of channels it supports.

The Connection ID Table implemented complies to the MANGO memory mapping specified in [8], which maps 16 entries to the table. However, since

Component Name	Computation (μm^2)	Storage (μm^2)	Total Area (μm^2)	Cell Count	% of area
8 Output Queues	68226	87520	155752	3965	46.31%
Conn ID Table	40755	31457	72212	1916	21.47%
Decap	43335	26075	69410	2507	20.64%
Encap	8544	11767	20312	821	6.04%
Queue Control	2310	2138	4448	161	1.32%
ReqE2E	6008	2433	8441	403	2.51%
RouteLookupTable	839	0	839	53	0.25%
InputPortMux	4902	1	4903	291	1.46%
Slave NA	174919	161391	**336314**	**10117**	100%

Table 7.1: Estimate of an area breakdown of the slave network adapter using 0.18 μm cell library.

there are only 8 output virtual channels and hence only 8 possible connection setups, we only need to use half of the storage space in the Connection ID Table. Therefore, the size of the Connection ID Table shown here could be cut down by half.

The Decapsulation Unit takes up 20.64% of the overall area. Computation area is about twice the storage area. This is due to the complicated computation work done by the priority scheduler within the Decapsulation Unit. The storage area is mainly used for storing the incoming flits for packet reassembly and for storing control information used by the Priority Scheduler. The written code is not optimal for synthesis. Changing this may possibly result in area savings.

The Encapsulation Unit occupies about 6.04% in area. Most of its area usage is used for packet construction and segmentation, which is its main task. Therefore, there is not much room here for area reduction.

The total area of the slave NA is approximately 10K gates. According to [14], future designs should be composed of cores ranging approximately from 50K-100K gates. With modules of these sizes wire delays will still be manageable. In comparison to such core sizes, the slave NA adds about 10% to 20% of overhead in term of area. MIPS32® 4K$^{\text{TM}}$ Family processing cores synthesized using 0.18 μm process occupies area ranging between 1400000 μm^2 to 2500000 μm^2 [15]. So the slave NA occupies about 13% to 24% of the MIPS core.

This may seem quite large, but whether or not this area overhead is affordable depends on the situation at hand.

7.2.2 Power Consumption

In order to generate some power consumption figures using Synopsys, a switching activity estimate for all the input and output signals was conducted. Normally, using a VHDL simulator and an appropriate test bench, a detailed switching activity summary can be generated for all signals throughout the design. Hence, a good estimate of power consumption can be obtained. However, due to limited time, the switching activity estimate was only approximated by hand for the signals at the interfaces of the slave NA. Therefore the results shown in this section can only be used as a rough guideline. Table 7.2 shows a power breakdown for the slave NA using the method described when estimating the switching activity for a READ command. Power results for other commands consume approximate the same amounts of power, therefore will not be included.

Component Name	**Total Dynamic Power (mW)**	**Cell Leakage Power (μW)**	**mW / MHz**
8 Output Queues	145.40	7.20	0.19
Conn ID Table	51.87	3.39	0.07
Decap	32.63	2.94	0.04
Encap	5.86	0.70	0.01
Queue Control	1.01	0.17	0.00
ReqE2E	7.33	0.37	0.01
RouteLookupTable	0.33	3.54 e-02	0.00
Slave NA	525.96	14.98	0.71

Table 7.2: Power Estimate for Slave NA when performing READ operation.

The ordering of the power consumption from the most power hungry module to the least is the same as for area usage. The output queues again are at the top of the list and consume the most power. As discussed in the previous section, the number of output virtual channels determines the number of output queues implemented. Hence power can be reduced if we reduce the number of output virtual channels. Nonetheless, doing so will mean that a smaller number of guaranteed connections will be supported and the functionality of the NA will be less desirable. The MIPS32® 4K™ Family MIPS cores running at clock frequency of 90 to 167 MHz consume about 1.3 to 2.2 mW/MHz of power. The slave NA consumes approximately 46% to 68% less power than the MIPS core. This seems quite high for slave NA power consumption. However, the MIPS cores are optimized for power consumption and the power estimate presented here is not precise.

The columns of table 7.2 do not add up. The reason is dependent upon where among the components of the NA Synopsys adds the power consumption calculation for the clock signal. The clock signal, which is distributed across all components in the network adapter, consumes much of the overall power. Synopsys adds the clock signal power consumption only to the top level of the slave NA, not to the sub-components. Therefore, table 7.1 shows a dramatic increase in total dynamic power at the Slave NA level. The results shown here suggest that the clock signal consumes a fair amount of the overall power. One obvious way to reduce power consumption is to apply *clock gating* to components that are idle. It would be beneficial to add a controller at the top level of the NA that would monitor the interface activities of the modules within the NA. This can be done because the NA is modularized and can be partitioned fairly easily. If no event occurs at the module interfaces, the module would be idle and wasting power. In cases as such, it would save power to stop the clock to that module until an event occurs at the modules interface and the clock is re-enabled to that module.

Therefore, optimizing the slave NA VHDL code for synthesis and applying clock gating will improve the power performance of the slave NA. A better estimate of the power consumption can be achieved using a more accurate switching activity estimate method.

7.3 Suggestions for optimization

As suggested earlier, the throughput can be increased by increasing the depth of the output queues. However, doing so would increase both the area and power consumption of the NA. It is desirable to find an optimal number of virtual channels to implement such that the cost in terms of area and power consumption can be kept reasonably low while still maintaining an acceptable level of service.

From Tables 7.1 and 7.2, the output queues and the Connection ID Table occupies the most space and consumes the most power. Therefore, it would be beneficial to find alternative ways to implement them for synthesis. The Decapsulation Unit also is at the top of both lists. The difficult part for optimization lies in the implementation of the Priority Scheduler. Figure 7.1 also shows that the critical path lies in the Priority Scheduler within the Decapsulation Unit. It starts from when the `empty` signal changes to when the input channel is selected and marked as being selected using the `tag` signal. Therefore, to reduce area, power and speed, efforts should be made in the NA design to improve the efficiency of the Priority Scheduler.

```
Point                                        Incr       Path
-----------------------------------------------------------
input external delay                         0.00       0.00 r
empty[6] (in)                                0.00       0.00 r
Decapsulation/empty[6] (Decap)               0.00       0.00 r
Decapsulation/U12560/Z (M_ND3HSX3)           0.05       0.05 f
Decapsulation/U10867/Z (ND2HSX4)             0.07       0.12 r
Decapsulation/U12603/Z (F_IVHSX8)            0.05       0.17 f
Decapsulation/U10864/Z (ND2HSX4)             0.04       0.20 r
Decapsulation/U12604/Z (IVHSX8)              0.05       0.25 f
Decapsulation/U12598/Z (ND2HSX3)             0.05       0.30 r
Decapsulation/U12625/Z (AO7HSX4)             0.08       0.38 f
Decapsulation/U12718/Z (AO7CHSX4)            0.08       0.46 r
Decapsulation/U10862/Z (ND2HSX4)             0.07       0.53 f
Decapsulation/U10835/Z (F_IVHSX8)            0.04       0.58 r
Decapsulation/U12722/Z (F_ND2HSX6)           0.04       0.62 f
Decapsulation/U12610/Z (IVHSX8)              0.03       0.65 r
Decapsulation/U12597/Z (F_ND2HSX6)           0.04       0.69 f
Decapsulation/U12687/Z (F_NR3AHSP)           0.08       0.76 r
Decapsulation/U12642/Z (F_AO7HSP)            0.07       0.83 f
Decapsulation/U12666/Z (ND3HSX3)             0.07       0.90 r
Decapsulation/U12686/Z (F_IVHSX8)            0.06       0.96 f
Decapsulation/U12685/Z (NR2HSX3)             0.06       1.01 r
Decapsulation/U12613/Z (ND2HSX4)             0.06       1.07 f
Decapsulation/U12628/Z (F_ND2AHSX3)          0.05       1.12 r
Decapsulation/U10859/Z (F_IVHSX8)            0.04       1.17 f
Decapsulation/U12699/Z (F_ND2AHSX3)          0.03       1.20 r
Decapsulation/U12698/Z (AO3HSP)              0.05       1.26 f
Decapsulation/tag_reg[0]/D (FD2HS)           0.00       1.26 f
data arrival time                                       1.26

max_delay                                    1.35       1.35
library setup time                          -0.09       1.26
data required time                                      1.26
-----------------------------------------------------------
data required time                                      1.26
data arrival time                                      -1.26
-----------------------------------------------------------
slack (MET)                                             0.00
```

Figure 7.1: Slave NA critical path generated by Synopsys.

Chapter 8

Future Work

Contents

The work presented in this project lays the ground work for a NoC network adapter. Many additional features can be built on top of this foundation. For readers interested in further developing the DTU network adapter, this chapter will present two possible designs for the master NA and the duplex NA introduced earlier in sections 4.1.2 and 4.1.3 respectively. However, the designs introduced in this chapter are preliminary drafts, are incomplete and need further development. In addition to this, we will also present some suggestions for changes in the NA when support for bursts is provided.

8.1 Master NA Design

The master NA compliments a slave IP core. This means that it is only fit to receive slave OCP signals and to present master OCP signals. At the network interface end, except for GS setup and teardown response packets which are used to configure the master NA for GS connection setups and teardowns, the master NA can only recognize request packets and send response packets. In this section, we intend to give a general idea of the major internal components in a master NA. The design is very similar to that of the slave NA implemented in this project but with some slight variations. First we look at a block diagram of the master NA in Figure 8.1.

From Figure 8.1, we can see that the only major difference between the master NA and the slave NA is the replacement of the Request End-to-End Flow Control Unit (ReqE2E) with the Response End-to-End Flow Control Unit (RespE2E), which we will examine in more detail in Section 8.1.1. The implementation of the Connection ID Table, the Route Lookup Table, the Queue Control, and the Output Queues remains the same. However, the Encapsulation Unit should, instead of packetizing request packets, packetize response packets; and the Decapsulation Unit, instead of recognizing and depacketizing response packets, should recognize and depacketize request packets. In order to do so, both the Encapsulation Unit and the Decapsulation Unit need to use the packet format presented in Section 4.4.

8.1.1 Response End-to-End Flow Control Unit

The RespE2E unit communicates with the slave NA. Therefore, it needs an OCP Master Control unit which will control the communication at the OCP interface according to the OCP timing specification discussed in Chapter 3. When the RespE2E unit presents the master OCP signals to the slave core, the slave core can choose to either respond to the request in the same clock cycle as it accepts the request, or respond to the request several clock cycles later. The slave core accepts the request by raising the SCmdAccept signal. However, if SCmdAccept is not raised, the RespE2E unit must hold all its master request signals until SCmdAccept is asserted. Hence, while the request is not accepted, the RespE2E unit must not accept any new request packet from the network. The hold_i signal is used to tell the Decapsulation unit to refrain from accepting more packet from the network until the request has been accepted. Figure 8.2 shows a possible implementation of the OCP Master Control finite state machine (FSM).

At reset, the FSM enters the IDLE state, where all master OCP signals to the slave core are invalid and the hold_i signal is de-asserted. While a new request message has not arrived, the FSM remains at the IDLE state. If a new request message arrives, resquest_arrived_i is raised, the PRESENT REQUEST state is entered at the rising edge of the next clock cycle.

At the PRESENT REQUEST state, the controller first checks the type of packet that has been received. If it is a BS or GS request packet, the FSM presents the master OCP signals for this particular request message and hold these signals while SCmdAccept is not asserted. If it is a GS setup or teardown packet, the master NA will enable write to the Connection ID table for NA configuration and also do a write to the slave core presenting it the connection ID for the reverse direction GS connection setup or teardown. The FSM also asserts the hold_i signal while at this state. It remains in PRESENT REQUEST state until SCmdAccept is asserted. If SCmdAccept

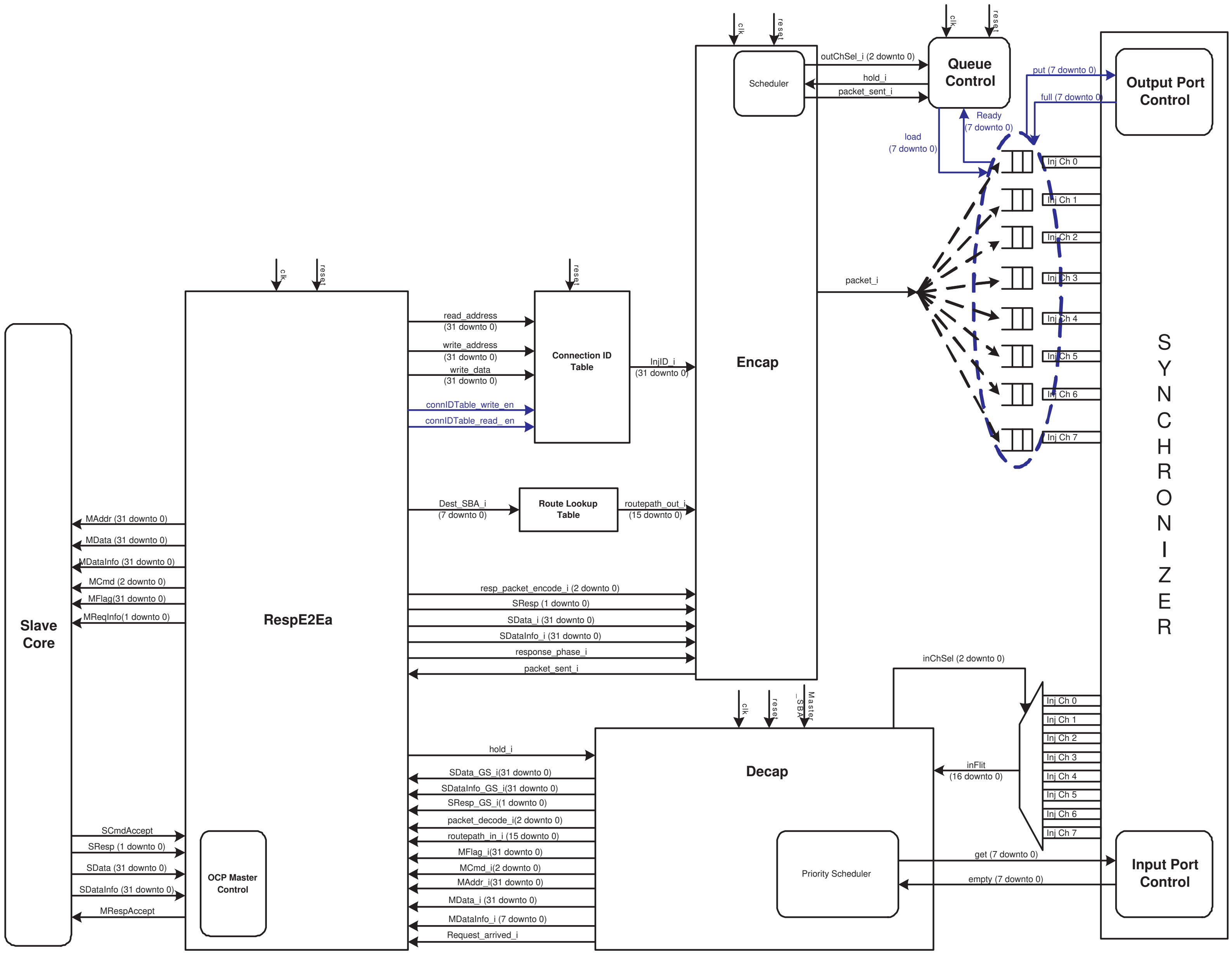

Figure 8.1: Block Diagram of a master network adapter.

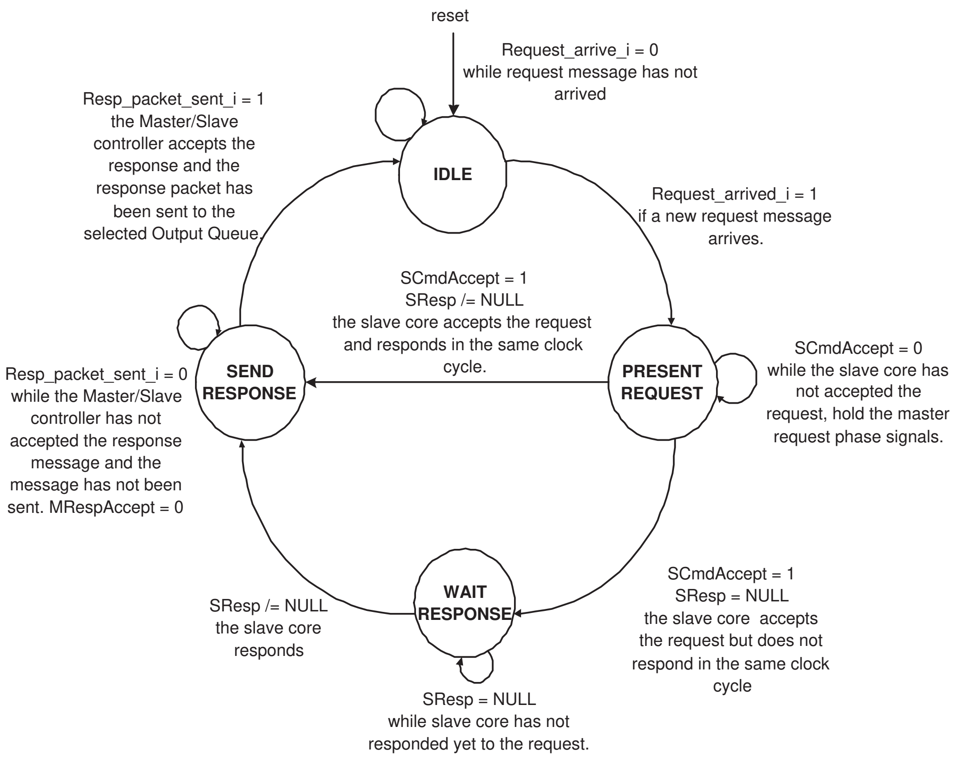

Figure 8.2: An example state diagram for Response End-to-End Flow Control Unit.

is asserted and SResp is NULL then the FSM waits for response and enter the WAIT RESPONSE state in the next clock cycle. If SCmdAccept is high and SResp is not NULL then SEND RESPONSE state is entered in the next clock cycle.

At the SEND RESPONSE state, the FSM determines the type of response this is by using the packet_decode signal and encode the response packet using resp_packet_encode_i correspondingly. For example if the request message is a BE request, then the response packet will be a BE response packet. MDataInfo_i is mapped to the sources address of the received packet 4.4. Therefore, MDataInfo_i can be used as input to the Route Lookup Table, DEST_SBA_i, to generate the route path back to the master core. The OCP response signals are passed on to the Encapsulation Unit for packaging and the hold_i signal to the Decapsulation unit is de-asserted. If the packet_sent_i signal is not asserted then all response signals is held constant and the FSM remains in SEND RESPONSE state. When packet_sent_i signal is detected asserted, the FSM returns to the IDLE state in the next clock cycle.

At the WAIT RESPONSE state, we continue to assert the hold_i signal and remain at this state until SResp is not NULL. If SResp is not NULL we enter the SEND RESPONSE state at the next clock cycle.

8.2 Duplex NA Design

The duplex NA combines the master and slave NA in one. The Encapsulation Unit and the Decapsulation Unit packetizes and depacketizes both request and response packets. The Connection ID Table and the Route Loookup Table are shared among the master and slave End-to-End Flow Control Units. Therefore, a Master/Slave Controller is needed to coordinate the master and the slave for accesses to the tables as well as the Encapsulation Unit, since it can only packetize one packet at a time. Figure 8.3 presents a block diagram for the duplex NA, which combined both the slave NA and the master NA with an additional Master/Slave controller. In the following section, we will discuss the additions to and changes to the network adapter in the duplex version.

8.2.1 Component Added - Master/Slave Controller

The Master/Slave controller needs to arbitrate between sending a request packet from the master OCP instance and sending a response packet from the slave OCP instance. It controls the enable signals to the Connection ID Table and selects which unit, either ReqE2E or RespE2E, is allowed to access the Connection ID Table and Route Lookup Table. Furthermore, it tells the Encapsulation Unit which type of message to packetize, either

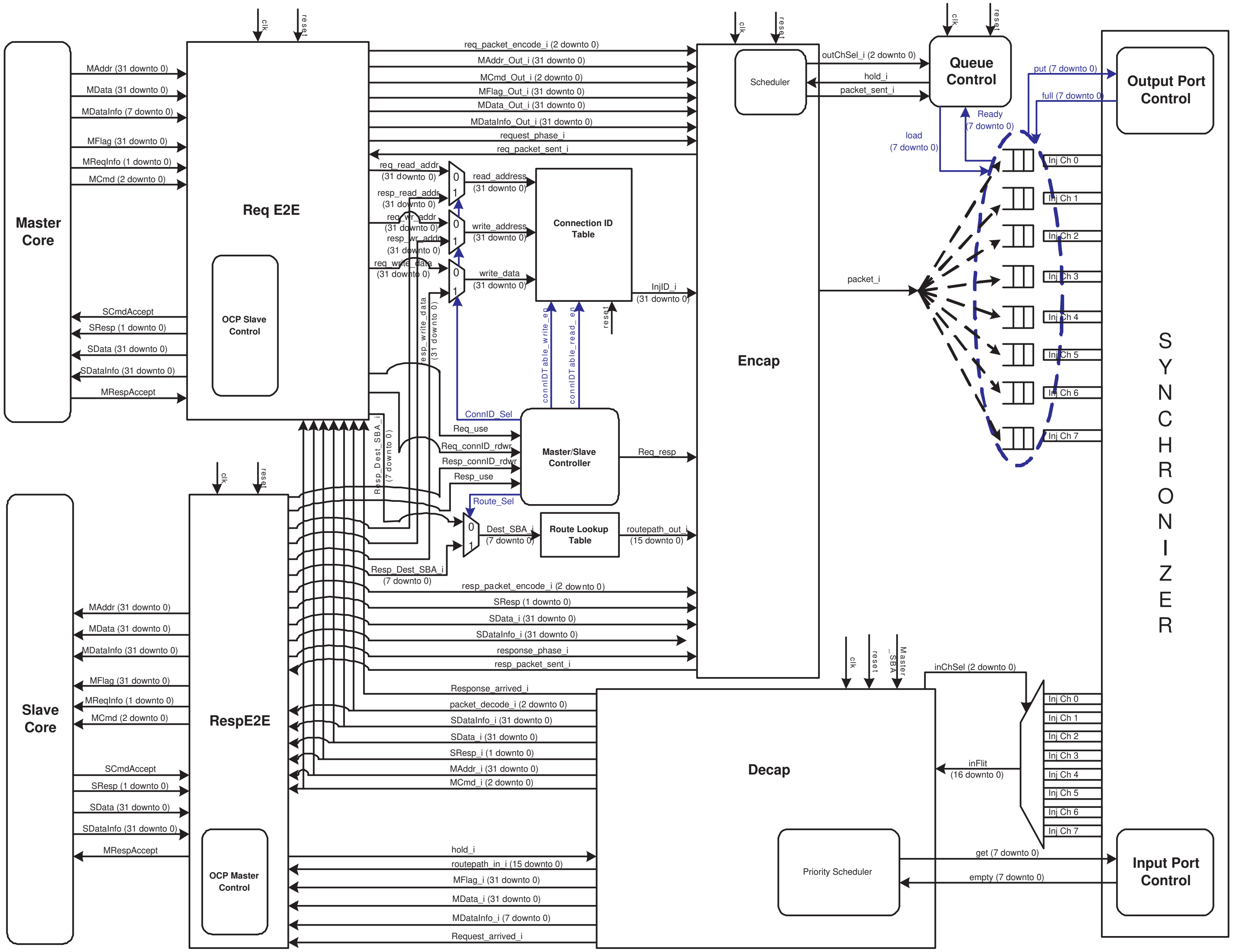

Figure 8.3: Block Diagram of a duplex network adapter.

a request message from the ReqE2E unit or a response message from the RespE2E unit.

The Master/Slave Controller takes in four inputs and five outputs. Inputs Req_connID_rdwr and Req_use are asserted when the ReqE2E unit needs to either use the Connection ID Table or send a request packet; and inputs Resp_connID_rdwr and Resp_use are asserted when the ResqE2E unit wishes to access the Connection ID Table or send a response packet. It is possible that both the Req_connID_rdwr and the Req_use signals are asserted the same time, and likewise for the Resp_connID_rdwr and the Resp_use signals. This means that the Connection ID Table should be able to accept a read and a write the same time. When the Master/Slave controller chooses to service the ReqE2E unit when it requests a read or write to the Connection ID Table, the Master/Slave controller will assert ConnID_Sel to choose inputs from the ReqE2E unit. Likewise, if RespE2E unit is chosen, the ConnID_Sel will be de-asserted. When the Master/Slave controller chooses to service a request packet to be packetized, it will assert Req_resp signal. If it chooses to service a response packet it will de-assert the Req_resp signal.

The Master/Slave unit needs to be further developed, as its arbitration algorithm has not been specified, only its behavior at its interfaces.

8.2.2 Components Modified

Response End-to-End Flow Control Unit

There needs to be two additional signals asserted in the PACKET SENT state of the RespE2E FSM. If the packet received is a GS setup or teardown response packet, the RespE2E must configure the NA for GS connection in the reverse direction. Hence it needs to write to the Connection ID Table. The Resp_connID_rdwr signal is used to inform the Master/Slave Controller in such cases. The Resp_connID_rdwr signal is asserted when writing to the Connection ID Table and de-asserted when reading from the Connection ID table. The second signal added is the Resp_use signal, which is asserted every time the RespE2E unit wants to send a packet to be packetized.

Request End-to-End Flow Control Unit

Likewise, two additional signals must also be added to the ReqE2E unit. A Req_connID_rdwr signal is asserted for reading and writing to the Connection ID Table for GS connection usage in the RESPONSE RECEIVED state of the Response OCP controller, which is presented in Section 5.2.2, Figure 5.7. A Req_use signal is used when the ReqE2E unit wishes to send a request packet to be packetized. This signal is asserted in the REQUEST RECEIVED state of the Request OCP controller presented in Figure 5.6.

Encapsulation Unit

The Encapsulation Unit will now receive a Req_resp signal from the Master/Slave Controller. When Req_resp is 1, the Encapsulation Unit will receive a request message from the ReqE2E unit to packetize. When Req_resp is 0, it will packetize a response packet from the RespE2E unit. The Encapsulation Unit will now be able packetize all six types of the DTU NoC packets according to the packet format in Section 4.4.

Decapsulation Unit

The Decapsulation Unit for the duplex NA has already been implemented. However, there needs to be one more mechanism added, which is the hold_i signal from the RespE2E unit. The Decapsulation Unit needs to stop selecting another channel when hold_i is asserted.

8.3 Burst Extension

Providing support for burst extensions on the NA means that the following issues must be addressed:

- A burst packet must be added to the packet type encoding in Figure 5.2.
- The packet format must then include new formats for carrying OCP burst extension signals.
- The RespE2E unit after receiving the base address for the burst packet and the burst length must keep track of the burst address increments and present the next consecutive burst read request to the slave core. After the slave core accepts each burst read request, it will respond with the burst data. The RespE2E unit then must send the correct response packet information for the Encapsulation Unit.

In this chapter, some design issue for developing a master NA and a duplex NA was discussed. The design details made here should only be used a guide for future master and duplex NA development.

Chapter 9

Conclusion

As mentioned in the introduction, the goal of this thesis project was to investigate the design of an OCP compliant network adapter for DTU Network-On-Chip (MANGO). Specifications at both the OCP interface and the network interface were provided by other members of the MANGO group to set the bounds for this design work.

During this project, several lessons were learned.

Firstly, the functionality and the design of the network adapter are closely tied to the design of the packet formats for the NoC. The more elaborate the packet formats is, the more work the network adapter must do to provide the services specified in the packet format.

Secondly, there is a definite tradeoff between the functionality of the network adapter and the cost of implementing it. The more functionality required, the more complex the network adapter, and therefore the more costly it is in terms of power and area. For example, in order to provide differentiated services, the network adapter must implement priority scheduling and output queues for the virtual channels. However, implementing priority scheduling is quite complex and therefore the computation complexity of the network adapter increases. The virtual channels provide means for differentiated services. An output queue is required for each virtual channel. The higher the number of virtual channels, the more queues and therefore the more area the queues will occupy.

Finally, in order for the network adapter to work properly and to be made more compatible to the rest of the NoC, the network adapter designer must have a clear and complete understanding of how the overall NoC works. The specifications at the OCP interface and the network interface need to be more clearly detailed.

The behavioral simulation results of the slave network adapter implementation showed throughput of 1 request every 4 cycles and a latency of the length of the packet plus 2.

An estimate based on a 0.18 μm cell library shows that the slave network adapter can approximately run at 741MHz and occupies roughly 336,000 μm^2 in area. A rough power estimate showed that the network adapter uses around 525 mW of power for every request it services. The performance estimates demonstrated that the area of slave NA implemented in comparison to future NoC core sizes adds about 10% to 20% of area overhead. The power consumption of the slave NA in comparison to MIPS32®4K™Family processing cores is that the slave NA consumes about 46% to 68% less in power.

The network adapter developed in this thesis project has paved the foundation for future network adapter developments. Several areas of the design still need to be further explored for functionality and efficiency. The crucial part is finding the correct balance between the amount of functionality that the network adapter supports and the cost of implementing them. This thesis has provided understanding for many design issues involved in the network adapter development as well as useful performance and cost estimates which will provide valuable insight to future designs.

Appendix A

DTU GS-OCP Specification

The GS-OCP Configuration
For
Networks-on-Chip

System-on-Chip Group, CSE-IMM
Technical University of Denmark
Lyngby 2800, Denmark

Version 1.1, April 23, 2004

Preface

This document provides information on the OCP interface and its configuration supported for Network-on-Chip (NoC) with guaranteed-services (GS). The reader is assumed to be familiar with the System-on-Chip concepts and multihop NoC concepts. Additionally the reader is assumed to be well versed in OCP, based on OCP-IP release 2.0 [1].

Contact Information: Shankar Mahadevan
Richard Petersens Plads
DTU-Building 322
DK-2800 Kgs. Lyngby
Denmark
Email: sm@imm.dtu.dk
Tel.:+45 4525 3754

Contents

1 Introduction

A system-on-chip (SoC) is a multiprocessor environment within a chip. The Network-on-Chip (NoC) is an multiport multihop communication media within the SoC. As seen in Figure 1, many different types of IP cores can be connected to the NoC. The layer between the IP core and the network hardware is defined as the Network Adapter (NA). It can implemented as a hardware module or hardware-dependent-software (HdS). The NoC design space encompass the NA as well as the network hardware. In this document we focus on a hardware NA.

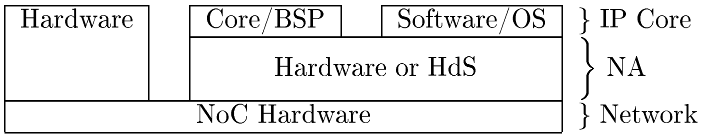

Figure 1: Overview of network interface types.

The boundary between the IP core and the NA is called the core interface (CI). The Open Core Protocol (OCP) [1] defines the CI. Our interest in using OCP is to provide seamless communication capability while preserving the plug'n'play characteristics between the IP core and the NoC. It is also been adopted by the industry and is becoming de facto standard. Regardless of the underlying architecture (topology and protocol) or the implementation (synchronous or asynchronous), the NoC, provides the same boundary i.e. the OCP through all its ports. The OCP itself can be configured based on the required local IP core and NA interaction at that particular port.

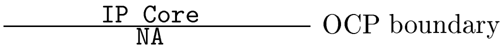

Figure 2: The OCP boundary between the IP core and the NA.

1.1 Objective

In this document we provide the specification for the OCP-compliant CI. The specifications are targeted specifically for guaranteed-services (GS) usage. This is thereafter called the GS-OCP boundary.

1.2 Motivation for the GS-OCP

Before highlighting the details of the GS-OCP, we present our motivation to provide an alternate, yet OCP-IP compatible signal set. Any NoC, in the very least, is expected to forward datum in a best-effort (BE) fashion. This is the default service. When issuing the OCP-IP commands on the `MCmd` field, if no additional specifics of how this command should be serviced is prescribed, then the command is serviced in BE fashion by the NoC. Thus the OCP-IP specification given in [1] is sufficient.

It is expected, from current NoC research, that in addition to the BE service, NoCs need to provide more advanced guaranteed-services (GS) to the core.

The GS may be classified in terms of guarantees of latency, through-put, jitter, power, etc. Thus an easy means to access and retain these services is required via the CI. Following section explains the service negotiation and use by the IP core.

Service Negotiation and Use

Special commands are expected to be used by the IP core or related software to ask and retain GS. These commands are transmitted across the CI to the NA for specific actions within the network to implement these services. Following are specifics of the type of commands relating to GS services.

- `SetUp(DestCore, TypeFwd, AmountFwd, TypeRtn, AmountRtn)`: The setup command, asks to establish GS with the destination core (*DestCore*). The GS can be either one-way or for round-trip. The *TypeFwd* and *AmountFwd* correspond to the GS of the request path, while *TypeRtn* and *AmountRtn* correspond to the GS for response path. The *type* is a representation of different kinds of service such as GS-latency, GS-jitter, etc. The *amount* would represent how much of the service is requested. For example GS of 5MB through-put or 10ns latency. Thus the value 5MB or 10ns is the amount argument. Once the service is established, the NA returns a connection ID (*ConnectionID*) for use by the core to invoke this service at a later time. If the service is not established, the NA returns appropriate error message. To setup only one-way GS, tie the alternate (response or request) path to zero.

- Using the established services: The OCP provides support for commands such as Read, Write, ReadEx, etc. Each of these commands may have a service request associated with it and want to use it. Two simple examples of using the established services may be:

  - `Read(ConnectionID, LocalAddrs)`
  - `Write(ConnectionID, LocalAddrs, Data)`

  From the examples, the *ConnectionID* parameter is the variable that allows to use the pre-established service. The *LocalAddrs* is the local address at the destination core *DestCore*.

- `TearDown(ConnectionID, DestCore)`: This command is used to free-up the network resource used for GS. The *ConnectionID* and *DestCore* are the arguments used by local NA to identify the established GS it represents for tear-down. The tear-down is expected to always return success.

The GS-OCP signal field is specified explicitly to support the above itemized style of invocation, use and tear-down of NoC service. An error during negotiation of the services is returned via the GS-OCP boundary. How the error is handled by the IP core is not in the scope of this document.

2 The GS-OCP Specification

The GS-OCP boundary has been specifically designed with a network-centric view. It adheres to the OCP-IP specification by the OCP-IP consortium [1].

As mentioned earlier, this specification is not relevant for NoCs that support no more than one service type. The novelty of the GS-OCP protocol is to transfer the GS calls between the IP core and the NA.

The signal set of the OCP-IP is not changed, but the GS-OCP specification designates additional fields from the OCP-IP signal set as mandatory. Table 1 shows the required fields of the GS-OCP basic signal set. The extension of the OCP-IP with mandatory `MFlag` and `MReqInfo` field serves the purpose of easing (flexible) service requests across the CI.

Name	Bit-Width	Driver	Function
`Clk`	1	*varies*	OCP clock
`MAddr`	configurable	master	Transfer address
`MCmd`	3	master	Transfer command
`MData`	configurable	master	Write data and also to specify response GS `TypeRtn` and `AmountRtn` during Setup phase
`MDataValid`	1	master	Write data valid
`MRespAccept`	1	master	Master accepts response
`SCmdAccept`	1	slave	Slave accepts transfer
`SData`	configurable	slave	Read data and GS connection identifier
`SDataAccept`	1	slave	Slave accepts write data
`SResp`	2	slave	Transfer response
`MFlag`	configurable	master	Transfer connection identifier and also to specify request GS `TypeFwd` and `AmountFwd` during Setup phase
`MReqInfo`	2	master	Transfer GS service command

Table 1: The GS-OCP basic mandatory signal set.

In the very least, an IP core must always provide the basic GS-OCP signals, to negotiate GS. All ports of the NoC will implement the basic GS-OCP set for accessing the network. The BE service can be invoked via the basic OCP-IP signal set (where the additional GS-OCP signals are tied to zero in the NA). The burst and thread extensions of the GS-OCP follow the OCP-IP conventions [1]. In the current version of the GS-OCP, the simple, sideband (except `MFlag` and `MReqInfo` respectively) and test signals are the not considered.

2.1 `MFlag`, `MCmd`, `MData` and `MReqInfo`

In this section, the meaning of `MFlag`, `MCmd`, `MData` and `MReqInfo` when driving these signals at the CI is explained. `MReqInfo`, in conjuncture with `MCmd` is used as GS negotiation encoding for the IP core master. Currently, `MReqInfo` is configured to 2-bits, thus representing up to four commands. Table 2 shows the encoding of `MCmd` and `MReqInfo`. `MFlag` and `MData` encode the service *type* and *amount* for the request and response paths during setup phase (see Figure fig:GSSetupIntprt). During other times `MData` has OCP-IP meaning [1], while `MFlag` carries GS identifier. Both are configurable fields. The bit-width for the type and amount may be different based on the network implementation, but equal for both the request and the response path.

		MReqInfo(2-bits)			
		0x0	0x1	0x2	0x3
MCmd (3-bits)	IDLE	Reserved	Reserved	Reserved	Reserved
	RD	BE	Setup	Use	Reserved
	WR	BE	Reserved	Use	Tear-down
	RDEX	BE	Reserved	Use	Reserved
	WRNP	BE	Reserved	Use	Reserved
	WRC	BE	Reserved	Use	Reserved
	BCST	BE	Reserved	Use	Reserved

Table 2: The `MReqInfo` and `MCmd` Signal Encoding for GS calls.

$$\underbrace{\texttt{MData}}_{configurable} \qquad \underbrace{\texttt{MFlag}}_{configurable}$$

$$\Downarrow$$

$$\underbrace{Additional\ bits}_{if\ any} + \underbrace{\mathtt{TypeRtn}\ \texttt{and}\ \mathtt{AmountRtn}}_{encoding} \qquad \underbrace{Additional\ bits}_{if\ any} + \underbrace{\mathtt{TypeFwd}\ \texttt{and}\ \mathtt{AmountFwd}}_{encoding}$$

Figure 3: `MData` and `MFlag` encoding during GS setup phase. During GS use phase the meaning is `MReqInfo` specific.

3 The GS-OCP Usage

The GS-OCP boundary has been specifically designed with a network-centric view. In this section we highlights how the GS-OCP signal set is interpreted by the NA. The section is relevant primarily for the implementation of the NA.

3.1 Addressing Interpretation

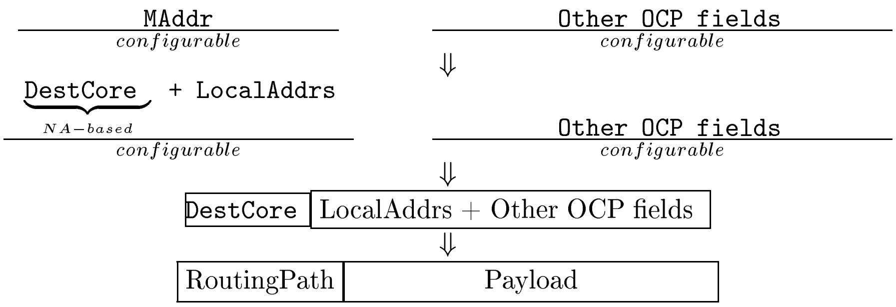

Figure 4: Address interpretation.

Figure 4 illustrates the GS-OCP NA's address interpretation via the `MAddr` field during the GS setup phase. The destination core address, provided in *DestCore*, would be a subset of the `MAddr` field and is used by the NA to uniquely identify the node in the network and draw a routing path to it.

This interpretation is also valid when using the default BE service. In that scenario, the remaining of the `MAddr` (designated as *LocalAddrs*) is the local memory address at that destination and has no useful meaning, except as payload, to the network.

3.2 Service Information Exchange

This section details how the IP core interacts with the NA to communicate the service requirements through the GS-OCP boundary. The `MFlag`, `MCmd`, `MData` and `MReqInfo` field in the GS-OCP is the means to transfer the service request from the IP core to the NA.

Best-effort Service Invocation

As mentioned before, BE is the default NoC service. It can be used directly when `MReqInfo` and `MFlag` are set to default values i.e. zeros.

Guaranteed Service Invocation

As seen in Table 1, the GS-OCP use additional signals for accessing GS via the NA. Table 3 summarizes which GS-OCP fields should be valid for invoking the GS mechanisms corresponding to discussion in Section 1.2. An interesting issue is error handling via the `SResp` field in the GS-OCP. The GS-OCP follows the response encoding of the OCP-IP (Table 4). The GS-OCP presents a "Request failed" when the asked GS cannot be provided. Additional information on the nature of failure may be embedded in the `SData` field. It is up to the core, then to take follow-up action such as attempting alternate type of service request, etc.

Request Phase					
GS-OCP Fields:	MCmd	MAddr	MData	MFlag	MReqInfo
Setup	RD	DestCore	*TypeRtn+AmountRtn*	*TypeFwd+AmountFwd*	0x1
Use	RD	`LocalAddrs`	DontCare	`ConnectionID`	0x2
	WR	`LocalAddrs`	Data	`ConnectionID`	0x2
TearDown	WR	DestCore	DontCare	`ConnectionID`	0x3

Response Phase		
GS-OCP Fields:	SData	SResp
Setup	`ConnectionID`	*when success*
	Optional error encoding	*when failure*
Use	*OCP-IP meaning*	
TearDown	*OCP-IP meaning*	Success

Table 3: The GS-OCP Service Request-Response Signal Summary.

4 Miscellaneous Concerns

The OCP-IP is an evolving standard for plug'n'play in SoC environment. Thus many cores support primarily the basic signal set within the OCP-IP. Since in

SResp(2-bits)	Response	Remark
0x0	No response	No response
0x1	Accept	Successful GS setup
0x2	Request failed	Failure of GS setup (Optional: Set SData field to the nature of failure)
0x3	Response error	Failure of GS setup (Optional: Set SData field to the nature of failure)

Table 4: The SResp signal encoding [1].

the DTU-OPC NoC, additional fields are used as part of invoking GS, a means to set this field via the basic OCP-IP signal set is needed. This issue is discussed in this section.

4.1 Guaranteed Service Request via the Basic OCP-IP Fields

IP Core
——————————— } Basic OCP-IP CI
$\boxed{\text{Address Interface}}$
——————————— } The GS-OCP CI
NA

Figure 5: The core to the NA OCP translation via Address Interface.

In order to incorporate IP cores that use an address-mapped view to invoke GS, an Address Interface (AI) module is proposed (Figure 5). The AI may be implemented to drive the appropriate GS-OCP fields via a memory mapped control register.

4.2 Concept of Connection ID

Using only destination address to identify a particular GS is not sufficient. A connection identifier (ConnectionID), associated with an established GS, is required since a single source-destination pair may have different types of GS concurrently established between them and simultaneously active. A centralized ConnectionID generator may be used to create the ConnectionID resulting in an identifier which is unique and visible globally in the network. Otherwise, ConnectionID may be generated locally by the NA; in this case uniqueness should be ensured either by additional header fields encodings related to source and destination of the communication or by means of end-to-end NA synchronization.

5 Example Transaction

In this section an example usage of the GS-OCP is presented. Consider a network which provides sixteen GS services. This is thus represented by 4-bits of the *type* encoding. The *amount* encoding is assumed to be granular up to 4-bits too. In the following table we show the transaction from source core A to destination core B, that may assist in implementing a NA. Following the GS-OCP specification, the *type* and *amount* are encoded by simple concatenating when presented to the appropriate GS-OCP signal.

Setup Phase `SetUp(coreB, 0x1, 0x5, 0x4, 0xf)`:

- Master Core A → Master NA : `MCmd` = RD, `MAddr` = coreB, `MData` = 0x4f, `MFlag` = 0x15, `MReqInfo` = 0x1. The Master NA interprets this as a round-trip GS request and response path setup and creates a network control packet with response path setup information in the payload.
- Master NA → Network → Slave NA : Network specific (path creation).
- Slave NA : The NA here recognizes the incoming packet as a network control datum. Instead of forwarding it to the attached device, it creates a table entry associating the traffic from the incoming request GS path to specific response GS path. A new network control packet is generated to establish the return path, based upon the payload information which encodes the `MData` field.
- Slave NA → Network → Master NA : Network specific.
- Master NA → Master Core A : Depending upon the round-trip network response, `SResp` is set. The `SData` field will hold ConnectionID in case of success or optional additional status information on error.

Use Phase `Read(ConnectionID, coreB+LocalOffset)`:

- Master Core A → Master NA : `MCmd` = RD, `MAddr` = coreB+LocalOffset, `MData` = DontCare, `MFlag` = ConnectionID, `MReqInfo` = 0x2. The Master NA interprets this as a GS packet and forwards it to the established channel.
- Master NA → Network → Slave NA : Network specific.
- Slave NA → Slave Device B : The NA here recognizes the incoming packet as a datum for the slave device and forwards it. The Slave device performs the read.
- Slave Device B → Slave NA : The NA receives the response data and does a table look-up to determine the response path. It then builds a packet, and forwards it.
- Slave NA → Network → Master NA : Network specific.
- Master NA → Master Core A : The `SData` field will hold read result.

Tear-down Phase `TearDown(ConnectionID, coreB)`:

- Master Core A → Master NA : `MCmd` = WR, `MAddr` = coreB, `MData` = DontCare, `MFlag` = ConnectionID, `MReqInfo` = 0x3. The Master NA interprets this as a network control packet and initiates teardown.
- Master NA → Network → Slave NA : Network specific.
- Slave NA : The NA here recognizes the incoming packet as a network control datum. Instead of forwarding it to the attached device, it deletes the table entry associated with the incoming request GS path. A new network control packet is generated to tear-down the return path.
- Slave NA → Network → Master NA : Network specific.
- Master NA → Master Core A : `SResp` is expected to assert "Accept". The NA guarantees the successful tear-down.

6 Conclusions

The aim of this document is to present the GS-OCP configuration, designed explicitly for NoCs supporting guaranteed services. It details the means to invoke, retain and tear-down these GS. The concept of threading and burst-mode transfers has not be shown in this version of the document, but follows the OCP-IP specifications [1].Testing and debugging is an on-going process and changes to the specifications may be made accordingly.

References

[1] Open Core Protocol Specification (Release 2.0). OCP International Partnership, 2001.

Appendix B

Programmer's Model for DTU NoC

Programmer's Model for DTU NoC

System-on-Chip Group, CSE-IMM
Technical University of Denmark
Lyngby 2800, Denmark

Version 1.1, June 25, 2004

Preface

This document provides the programmer's view for DTU Network-on-Chip (NoC). The reader is assumed to be familiar with the System-on-Chip concepts and multihop NoC concepts. And also GS-OCP document.

Edited by: Shankar Mahadevan
Richard Petersens Plads
DTU-Building 322
DK-2800 Kgs. Lyngby
Denmark
Email: sm@imm.dtu.dk
Tel.:+45 4525 3754

1 Introduction

The program's view describes the global address space of the DTU NoC. We make the assumption that address space is distributed equally between all IP cores, including the NoC. The NoC is composed of three components: network adapters (NA), routing nodes and links. The NA and the routing nodes are addressable. It is assumed that 4-bytes are available for addressing and 4-bytes are available for data (array of words), each 32-bit address pointing to a single 32-bit data word. In order to make efficient use of services available via the NA and nodes, a programmer's model is required. This document explains this model.

1.1 Memory Distribution

Of the 4-byte of address field, the most significant byte is used to represent the globally unique device address also called system base address (SBA), thus upto 256 IP cores can be represented. Following table shows the meaning of each byte. The bytes are shown in little endian order.

Byte	*Remark*
3rd	SBA, upto 256 devices
2nd and 1st	Combined are used to distinguish different components Byte meaning: 0000 to FFFC = Available to local IP core FFFD = for NA control FFFE = for Routing Node control FFFF = Reserved
0th	Local offset

Table 1: System Address Space Interpretation.

Since each IP core has an associated a node in the network, the NA and the local node can be addressed with the most-significant-byte, just as the IP core. Note, for nodes without attached IP cores, the addressing scheme need not be different. This may be inefficient use of address space, but useful for routing to that location. The next two byte (2nd and 1st) are used to distinguish the target component. The values in the lower 4-bits in this nibble identify the target component i.e. NA or node. The last byte of the address field is used for local operations in the specific component. Thus from the 4-byte address field, for system base address of 0x00:

- `0x00000000 to 0x00FFFCFF` are available for IP cores
- `0x00FFFD00 to 0x00FFFDFF` are available for NA control (256 words)
- `0x00FFFE00 to 0x00FFFEFF` are available for node control (256 words)
- `0x00FFFF00 to 0x00FFFFFF` are reserved

The memory space is distributed in an uneven format to optimize its usage. (The core is expected to make use of most of the available memory in any system.)

1.2 NA Address Space Distribution

Table 2 defines the meaning of local offset within the NA address space and the meaning of data value contained in it. Most significant byte 0xXX is the local SBA value.

NA Address (3rd to 1st byte)	***Local Offset (0th-byte)***	***Remark***	***Data (4-bytes)***
0xXXFFFD	00	Reserved	Reserved
0xXXFFFD	01	Status	0 = not-operational 1 = operational
0xXXFFFD	02 to 07 08 to 0F	For interrupt handling	Reserved for NA attached to master device Reserved for NA attached to slave device
0xXXFFFD	10 to 1F	Connection ID table entries	Returns first entry Returns last entry
0xXXFFFD	20 to 2F	Injection ID table entries	Returns first entry Returns last entry
0xXXFFFD	30 to FF	Reserved	To be defined

Table 2: NA Address Interpretation.

The NA interrupt types are: external interrupt (from other Slave IP), local buffer full, GS not available, transaction not allowed (burst/thread transaction without GS), etc. The interrupt handling procedure is explained later in Section 2.

1.3 Node Distribution

Table 3 defines the local offset within the routing node address space and the meaning of data value contained in it. The node interrupt types are: VC i/p to i/p not allowed, VC type mismatch (cannot reserved BE VC), VC not available, etc.

2 Interrupt Handling

Interrupt handling is unique in multi-master multi-slave environment. In this section we explain interrupt handling for NA. The `SInterrupt` (1 bit) field is available in OCP-IP for the Slave IP core to inform the Master IP core of interrupt occurance. The Slave IP core asserts SInterrupt, followed by which the attached local NA Master performance a read to get the interrrupt type, followed by which the interrupt information is packetized and sent to the Master IP core, finally raise the interrupt at the Master IP core by attached local Slave NA. Thus, interrupt handling involves three steps: (i) programming the NA of the Slave IP core to read the interrupt type in the Slave IP core and where to send it. (ii) packetize and send the interrupt to the designated Master, and (iii) raise the interrupt at the Master IP core. Refering back to Table 2, following is the memory view for the NA:

- 0xXXFFFD02 to 0xXXFFFD07 are available to program the (slave) NA attached to the designated master device to write the interrupt infor-

Routing Node Address (3rd to 1st byte)	***Local Offset (0th-byte)***	***Remark***	***Data (4-bytes)***
0xXXFFFE	00	Reserved	Reserved
0xXXFFFE	01	Status	0 = not-operational 1 = operational
0xXXFFFE	02 to 0F	For interrupt handling	Interrupt type (to be defined)
0xXXFFFE	10 to 1F	North VC table entries	Returns first entry Returns last entry
0xXXFFFE	20 to 2F	East VC table entries	Returns first entry Returns last entry
0xXXFFFE	30 to 3F	South VC table entries	Returns first entry Returns last entry
0xXXFFFE	40 to 4F	West IP VC table entries	Returns first entry Returns last entry
0xXXFFFE	50 to 5F	Local VC table entries	Returns first entry Returns last entry
0xXXFFFE	60 to FF	Reserved	To be defined

Table 3: Routing Node Address Interpretation.

mation such as who created the interrupt (SBA and component) and type of interrupt. The value in the interrupt register 0xXXFFFD02 at the (slave) NA attached to master device has following byte-location-dependent meanings:

- `3rd byte`: SBA, to know where the Interrupt originated
- `2nd byte`: Reserved
- `1st byte`: Component, which component at that SBA defined location asserted interrupt (IP core, NA or node)
- `0th byte`: Interrupt type, component depended value

The rest of the location 0xXXFFFD03 to 0xXXFFFD07 are yet to be defined.

- 0xXXFFFD08 to 0xXXFFFD0F are available to program the (master) NA attached to the slave device. It can be programmed with memory location within the slave device from where to read the interrupt type and the SBA of the designated master.

Note multiple interrupts can be sent to the same master. Since there is only one request path (via the receiving NA's Slave), it is automatically queued. It is up to the Master IP core's interrupt controller to judge the interrupt's priority and take suitable action.

3 Overview and Conclusion

Figure 1 provides an overview of the programmer's memory model. This document is subject to change.

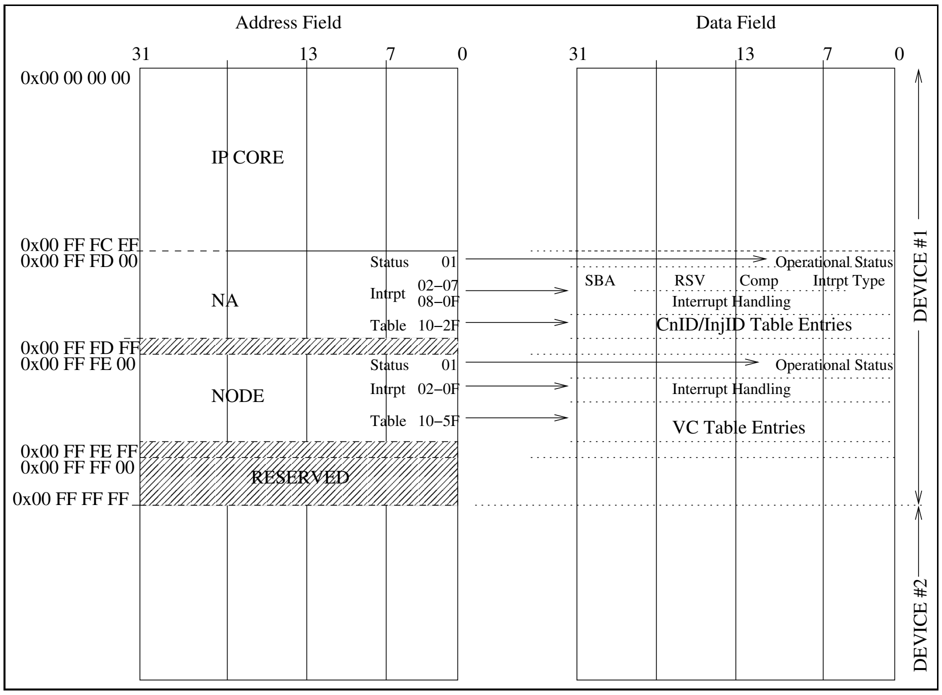

Figure 1: Memomy Map Overview

Appendix C

DTU NoC Network Interface Specification

Specification of the Network Interface (NI) (ver 0.2)
System-on-Chip Group
Informatics and Mathematical Modelling
Technical University of Denmark (DTU)
edited by Tobias Bjerregaard, tob@imm.dtu.dk

C.1 Introduction

In our NoC, The network adapter (NA) will be the module linking the OCP compliant cores with the underlying hardware structure (the actual network). The hardware will provide a set of primitives for packet routing. The concept is that the NA maps the services specified by OCP to these hardware primitives. The NA implements two interfaces: the Core Interface (CI), an OCP compliant interface [16] connecting the core to the NA, and the Network Interface (NI) which interfaces the NA to the network hardware. How OCP compliance is adapted for our NoC is specified in [17].

This document contains details of the NI and beyond. Thus its purpose is to specify how the NA communicates with the network, and also how the network is structured. It should evolve into a true specification, but at first it will be a working text, to be used by all involved as a reference to our ideas on what to build.

C.2 The Network

The network will route packets. The basic routing strategy is wormhole routing. Each network node will implement a number of in/out ports, through which packet worms can be streamed. The network can have an arbitrary architecture, but we will target grid-related topologies. The port to which the NA attaches itself is not different from the other ports. Thus the NA's task is to generate network compatible packets and stream these to the network. In the following, details of the packet format for different types of traffic will be explained.

C.2.1 NoC primitives and components

The primitive communication services provided by the network are:

BE - Best Effort routing.
GS - Guaranteed Service routing.

These services are supported by the network hardware. Its basic architecture, and the features it supports, will be explained in the following. The basic components of the system are the Network Adapter (NA), the Network Controller (NC), the Node Controller (NodeC) and the Links.

The NA handles the decoupling of communication and computation in the system, by translating high level communication requests, i.e. across OCP interfaces, to low level network primitive service requests. The NA handles end-to-end control of the global communication in the system.

The NC is a special core which controls setup of access to the GS of the network. The object of having such a core is to gain the advantages of centralized control during setup while maintaining the advantages during runtime of distributed control.

The NodeC controls the nodes, by offering a memory mapped view of the setup registers in each node. Thus GS can be setup simply by writting to these registers. In the future, it could prove interresting to investigate how the NodeC could implement more complicated tasks, such as distributed GS setup or other.

The Links provide the basic resource of the network: bandwidth. Each link will implement a number of logically independent *virtual channels* (VCs) sharing the same physical channel. These VCs will be used to provide differentiated BE traffic as well as GS traffic, as will become more clear later in the following sections.

C.2.2 BE

There are a number of ways that BE routing can be implemented. Two in particular are source routing and progressive adaptive routing, which will be impemented in our NoC:

Source Routing (SR) - the NA translates a (memory mapped) destination core address into a routing path. This is done through a table lookup. The content of the table is uploaded by the NC at start-up or during runtime. In any case, it is viewed as a static table. The routing path specifies the exact hop-by-hop path from source to destination.

Progressive Adaptive Routing (PAR) - instead of specifying a routing path, the number of hops in each direction is provided. It is then up to the node to decide which direction the packet takes. Thus the packet always progresses towards the destination, but not by a predetermined path. In order for this solution to have a simple implementation, a regular grid topology is required.

The implementation of PAR constitutes an overhead in the node, in that support for decision making, and for decrementing the hop count values is needed. The time spent by the node to make the decision will be higher because of the complexity of the decision. Also, the entire header needs to be received before the packet can be transmitted. On the other hand, in a highly loaded network, the packets will be routed around congested areas, thus evening the network load, making the most out of available resources. The trade-off to consider is whether PAR is worth the area/power. It might be better to implement a wider datapath, making a simple network faster, and thus avoiding high load in the network. Also, if some prior knowledge of the global system communication is available, the sourcerouting paths can be chosen as to avoid hotspots. These trade-offs need to be investigated for a variety of applications.

Differentiated BE

Our network will implement 2 priority levels for BE packets: high (BEhi) and low (BElo). Interrupts could be routed as high priority packets, since they often require a minimal part of the total bandwidth. Each packet type will be routed on a separate set of virtual channels. Thus they are logically independent, and low priority packets cannot stall high priority packets. Since high priority packets will take priority over low priority BE packets, and furthermore not be stalled, they will zoom through the network, independently of any other traffic.

Packet Format

In Figure C.1 the BE packet format is specified, and it is specified how the packet context is interpreted at different levels in the path between NI and CI (within the NA)

Network level: the *Header* contains the identifiers *packet class* and *routing operand*. The packet class determines which class the packet belongs to. This can be used to identify different priorities of best effort routing, system packets, or other types of packets. In our initial scheme the different packet types are routed on logically independent networks (on separate sets of virtual channels), thus the packet class identifier is not stricktly necessary. We include it however, in order to allow for future expansions. It will be 2

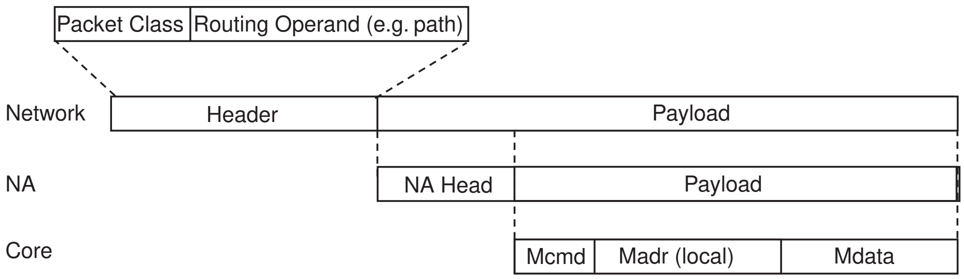

Figure C.1: The format of a network packet.

bits, and the default value will be 00. The routing operand field is used as operand to the packet class identifier.

In the SR scheme the routing operand holds the routing path, hop by hop. Note that it does not have a fixed length, as the routing path will be a non fixed number of hops.

In PAR, the operand will be further subdivided, containing three fields: direction (NE:00, SE:01, SW:10, NW:11), number of hops along one diagonal, and number of hops along the other diagonal. Example: 1010010011 means: direction SW (10), 9 hops south (1001), 3 hops west (0011).

In a fully adaptive routing scheme, the routing operand could simply identify the destination core, leaving it to the network to find the optimal path.

NA level: The *NA header* holds information that the NA needs to identify threads, response channels etc. Some of this information is needed by the destination NA to handle end-to-end control. The address provided by the (source) core is partly global (specifying the destination core) and partly local (specifying an address at the destination core). The global part is what is being translated to a routing path. The local part is just being forwarded to the destination core. Details of this and more is available in [18]

Core level: The format at the core level is the format of the CI, which in our case is determined by the OCP specification. Details of our subset of OCP is available in [17].

When streamed through the network, different points on the packet need to be identified: (i) end-of-header and (ii) end-of-payload. Start of header is simply the first that happens after (ii).

Deadlock Avoidance

The network must be implemented so as to avoid deadlocks. This is done by making sure that there are no cycles in the resource dependency graph [19]. In our BE routing schemes many packets may simultaneously share the network, thus the potential for deadlock is present. However, the fact that the SR and PAR routing are incremental means that there are simplifications that can be taken into account, making dealock avoidance easy to obtain. Since the routing is incremental, we know that packets will always move in one and only one of the following directions: NE, SE, SW, NW. Thus, by having two virtual channels along each link, 1 and 2, and allocating their use such that packets moving along particular directions make use of particular channels as shown in Table C.1, packets in each of the four directions flow along independent sets of VCs, thus deadlock is guaranteed to be avoided. Stricktly speaking, 2 VCs along each link is not needed to avoid deadlock, but it is nice and symmetric, also the fact that packets a routed along 4 logically independent paths relax interpacket dependencies, and minimize congestion problems.

Table C.1: Virtual Channel (VC) Assignment

Packet Direction	**VC Assignment**
NE	N:ch1, E:ch1
SE	S:ch1, E:ch2
SW	S:ch2, W:ch1
NW	N:ch2, W:ch2

Implementing the described scheme for VC assignment, there are no further routing restrictions necessary to avoid deadlock. An advantage of the scheme is also that less bits are needed in the header during SR, because there are only three directions to take at each hop: *straight*, *turn* or *arrive* (at local core).

C.2.3 GS

In the following, the use of GS is briefly described. A detailed explanation of its use is beyond the scope of this document. Instead, we will focus on the concept and implementation of the primitives needed to support GS communication.

GS traffic is implemented by setting up a path from source to destination. This is can be done by the NC. Upon receiving a request for a GS path from the core, the NA should forward this request to the NC. What should be forwarded is: the path (or at least the destination core - then the NC will determine a path), the type of service (an integer specifying the service type - see below) and the amount of the service (another integer). The request will be forwarded to the NC as a normal BE packet. The data payload of the packet (see Section C.2.4) is to contain the information mentioned. The NC will program the nodes, by writting directly to the registers within, through a memory mapped access interface. The node will thus, through the NodeC as explained earlier, look to the NC as a small memory (an OCP slave).

When the NC has setup the path, it will inform the NA, who will in turn conclude the request from the core and provide the core with a connection ID (a locally unique name for the service). If the path could not be guaranteed, naturally the NA will conclude the request with a denial. A specific return path may also need to be setup in the destination NA. All this is detailed in [18].

The following types of path services are envisioned, the support for which HW primitives are needed:

- **Bandwidth:** a guaranteed high bandwidth path.
- **Low Power:** a path which implements lowpower encoding on links.
- **Latency:** a path with a low latency bound.

The details of their implementation will follow in future documents.

When accessing a service which has been initialized, the connection ID is provided to the NA by the core. This connection ID is translated (by table lookup) by the NA to a channel by which the network is accessed. No routing path is needed, since the channel maps directly to the (connection-oriented) GS path, as will be described in Section C.2.3.

GS Routing

When routing GS packets, the Network Header described in Section C.2.2 is not necessary. A GS path is connection-oriented and as such it behaves like a virtual point-to-point connection. Thus, when specifying the injection channel at the source, this leads automatically to the packets ending up at the destination according to the specification of the path. This is what we define as a *virtual circuit*. Figure C.2 illustrates how a virtual circuit is instantiated, by mapping input VCs to output VCs at each node.

If no header is used, it is required that the nodes contain some sort of knowledge about which output channel any input channel which is part of a GS path maps to. This can take the shape of a routing table in each node. The information could also be carried in the header of the packet. The header could contain a sequence of 'next hop' channels, sort of like source routing, except not only the direction of the routing is specified, but also the specific VC to use. The paragraphs below describe these two different GS routing styles. Table C.2 is an overview of the pros and cons of each.

No Header Routing: Routing information stored in local node table. Pros: (i) no header in packet meaning optimal bandwidth utilization, also low packet latency, (ii) fast routing: next hop can be looked up before packet arrives, or ca be available always (in a dedicated register). Cons: (i) area

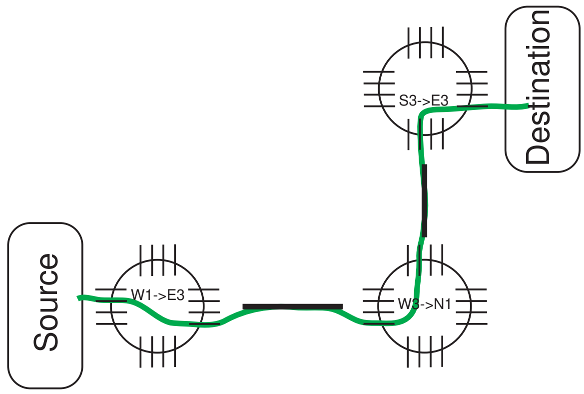

Figure C.2: A set of virtual channels create a virtual circuit, for use as a GS path. At each node, the input VC is mapped to one particular output VC. The VCs are used only by that particular GS path, and traffic along the path is thus logically independent of other traffic in the network.

overhead because nodes need local lookup table, (ii) slow setup since tables needs to be updated in all nodes along the path before path can be used.

Source Routing: Routing information is stored in packet header. Pros: (i) no local routing table, meaning no area overhead in node, also, low setup time. Cons: (i) header causes lower bandwidth utilization and higher latency. Also, if there are a large number of output channels to choose from, either a decoding scheme is needed (increasing complexity of node circuitry) or the header will be very large, (ii) central GS routing decisions a must.

Table C.2: GS Routing Schemes

Scheme	**pros**	**cons**
No header	optimal bandwidth utilization, low latency	slow setup, area overhead in nodes
Soruce Routing	no local routing table	lower bandwidth utilization, higher latency, central routing decisions a must

Deadlock Avoidance

Since each GS path (virtual circuit) is formed by a unique set of logically independent channels (VCs), deadlock will never be an issue. One thing to keep in mind is: the path can only use each VC once, i.e. loops are allowed (perhaps in a broadcast situation), but not with re-use of VCs.

C.2.4 The Network Interface (NI)

The NI is the interface by which the NA plugs into the network. It consists of two unidirectional ports, one port for each direction (in and out of the NA). There are a number of fields: At each port a number of *injection channels* (equivalent to VCs) are available by which the network can be accessed. When the NA creates a packet, it determines such an injection channel, by which the packet enters the local node. This choice depends on the type of routing (BEhi, BElo, direction (NE, SE, etc), GS). This can be considered part of the routing path. For GS transactions the injection channel uniquely identifies the entire path, as it constitutes the starting point of a virtual circuit. At each node, the particular input channel is mapped to a specific output channel, and thus, hop by hop, the entire path is determined, as decribed in Section C.2.3.

The fields which need to be created by the NA are thus:

- **Injection Channel:** the channel by which to engage the routing.
- **Header:** the header of the packet, as specified in Section C.2.2.
- **Payload:** the data payload to be carried by the packet.

The data of the fields need to be serialized for the network. Keep in mind that the header and payload are not necessarily fixed length (more on that as this specification gets more specific).

C.2.5 Network Architecture

The network consists of nodes connected by links. The basic implementation is grid approximation, but it is not restricted to this. The deadlock avoidance scheme described for BE routing, in section C.2.2, is based on a grid architecture. It can easily be expanded to an irregular structure, as long as it is a subset of a grid and incremental routing is ensured. For more complex irregular structures, further investigations are necessary.

C.2.6 Node Architecture

Each node consists of the actual node, and a NodeC. The NodeC implements a memory mapped access point to the internal registers of the node, in particular those needed to setup virtual circuits. Other ideas for features that could be setup by the NodeC include ways to specify prioritization, algorithms for new routing schemes, etc. The node controller might also be a more complicated module, used in distributed GS path allocation.

The NodeC is envisioned to share the NI of the NA, thus some of the circuitry in the NA can be reused, such as synchronization and packet decoding circuitry. The NodeC is synchronously implemented. This makes its design easy and redesign fast. The registers being setup for virtual circuits are considered static when in use. They are setup long before being used, and torn down long time after their use has ceased. Thus no conflicts concerning synchronization between events in the asynchronous domain of the network and the synchronous domain of the NodeC occur.

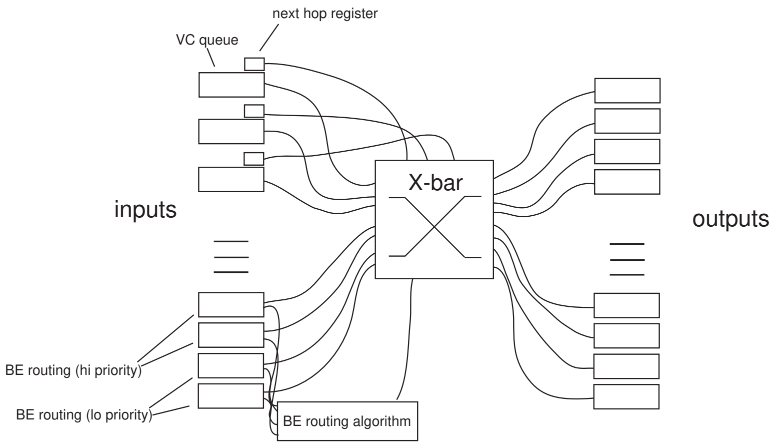

Figure C.3: Node architecture.

The node itself will implement 16 channels on each input and output port, and a X-bar routing between these. This is illustrated in Figure C.3. Four channels will be used for BE routing (BEhi and BElo) and 12 channels will be used for GS routing. We will adopt the GS routing scheme in which the routing information is stored locally in the nodes (no header), thus each of these 12 GS channels will have a register which identifies the next hop.

The exact implementation of the X-bar is not decided at present. It may either be a traditional X-bar design, or perhaps a mesh of forks and joins. Providing GS across a fully connected X-bar is a major issue. Also, a design which is scalable to many more VCs is also an issue. Perhaps a design of FLEETzero [20] type would be appropriate.

C.2.7 Link Architecture

The links will implement 16 virtual channels, to accomodate the 16 channels described in Section C.2.6. Issues to investigate are: arbitration and prioritization (dynamic or static?), pipelining, and more..

Appendix D

Slave Network Adapter Source Code

The source code for the slave network adapter is burned on CD attached to the back of this thesis.

Bibliography

[1] The international technology roadmap for semiconductors. Technical report, Semiconductor Industry Association, 2001.

[2] Praveen Bhojwani and Rabi Mahapatra. Interfacing cores with on-chip packet-switched networks. In *Proceedings of the 16th International Conference on VLSI Design*, pages 382–387, 2003.

[3] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vencentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proceedings of the 38th conference on Design automation*, pages 667–672. ACM Press, 2001.

[4] Luca Benini and Giovanni De Micheli. Networks on chips: A new SoC paradigm. *Computer*, 35(1):70–78, 2002.

[5] Andrei Rădulescu, John Dielissen, Kees Goossens, Edwin Rijpkema, and Paul Wielage. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Volume II (DATE'04)*, page 20878. IEEE Computer Society, 2004.

[6] K. Goossens, J. van Meerbergen, A.Peeters, and P. Wielage. Networks on Silicon: Combining Best-Effort and Guaranteed Services. *DATE´02*, Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition, 2002.

[7] OCP International Partnership. *Open Core Protocol Specification, Release 2.0*, 2001.

[8] Shankar Mehadevan. Programmer's Model for DTU NoC. 2004.

[9] Tobias Bjerregaard. Specification of the Network Interface. 2004.

[10] Shankar Mehadevan. The GS-OCP Configuration for Networks-On-Chip. 2004.

[11] Martin Hans. Architectural aspects of design for low static power consumption. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2004.

[12] STMicroelectronics. *CORELIB8DHS HCMOS8D 3.1 User's Manuals*, september 2001.

[13] The national technology roadmap for semiconductors. Technical report, SIA – Semiconductor Industry Association, 1997.

[14] Dennis Sylvester and Kurt Keutzer. Impact of small process geometries on microarchitectures in systems on a chip. In *Proceedings of the IEEE*, volume 89, pages 467 – 489. IEEE, 2001.

[15] MIPS Technologies. Mips32® 4k™ family. `http://www.mips.com/content/Products/Cores/32-BitCores/MIPS324KFamily/ProductCatalog/P_MIPS324KFamily/productBrief`.

[16] Open Core Protocol Specification, Release 1.0, 2001.

[17] OCP Configuration for DTU Networks-on-Chip, 2004.

[18] Juliana Zhou. Network adapter specification, 2004.

[19] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[20] William S. Coates, Jon K. Lexau, Ian W. Jones, Scott M. Fairbanks, and Ivan E. Sutherland. FLEETzero: An asynchronous switching experiment. In *Proceedings of the Seventh International Symposium on Asynchronus Circuits and Systems, 2001 (ASYNC 2001)*, pages 173–182. IEEE Computer Society, 2001.