Ayan Mandal · Sunil P. Khatri
Rabi Mahapatra

Source-Synchronous Networks-On-Chip

Circuit and Architectural Interconnect Modeling

Springer

Source-Synchronous Networks-On-Chip

Ayan Mandal • Sunil P. Khatri • Rabi Mahapatra

Source-Synchronous Networks-On-Chip

Circuit and Architectural Interconnect Modeling

Springer

Ayan Mandal
Computer Science and Engineering
Texas A&M University
College Station
USA

Rabi Mahapatra
Computer Science and Engineering
Texas A&M University
College Station
USA

Sunil P. Khatri
Electrical and Computer Engineering
Texas A&M University
College Station
USA

ISBN 978-1-4614-9404-1 ISBN 978-1-4614-9405-8 (eBook)
DOI 10.1007/978-1-4614-9405-8
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013951532

To,
Our families
—Ayan, Sunil and Rabi

Preface

This book is motivated by the challenges faced in designing a high speed low power on-chip communication framework in modern digital ICs. As VLSI fabrication technology scales, an increasing number of processing elements (cores) on a chip makes on-chip communication a new performance bottleneck. The Network-on-Chip (NoC) paradigm has emerged as an efficient and scalable infrastructure to handle the communication needs for such multi-core systems. In most existing NoCs, design decisions are made assuming that the NoC operates at the same or lower clock speed as the cores, which slows down the communication system. Since the NoC connects the cores across the entire chip, it becomes difficult to operate the NoC at a high frequency. Another major challenge in designing a high speed NoC is the difficulty of distributing a high speed, low power clock across the chip.

This book consists of three parts. In the first part, we propose several techniques to address the issue of distributing a high-speed, low power, low jitter clock across the IC. We primarily focus our attention on resonant standing wave oscillators (SWOs), which have recently emerged as a promising technique for high-speed, low power clock generation. In addition, we also present a dynamic programming based approach to synthesize a low jitter, low power buffered H-tree for clock distribution.

In the second part of this book, we use these efficient clock distribution schemes to present a novel fast NoC design that relies on source synchronous data transfer over a ring. In our source-synchronous design, the clock and data NoC are routed in parallel yielding a fast, robust design. The source synchronous NoC is clocked by SWOs, which operate significantly faster than the cores that are served by the rings. This allows us to significantly improve the cross section bandwidth and the latency of the NoC. We develop a deadlock-free routing protocol for the source-synchronous ring-based NoC. Our modified source-synchronous design allows the cores to extract a low jitter clock directly from the high speed ring clock by division, and hence the cores operate synchronously with the NoC. This is significant since it eliminates synchronizer latencies that are typically incurred in an asynchronous design. Using the above modified design, we propose a class of source-synchronous, floor-plan friendly NoC topologies which consume significantly lower area compared to a state of the art mesh. Architectural simulations on synthetic and real traffic show that our source-synchronous NoC designs can provide significantly lower latency while achieving the same or better bandwidth compared to a state of the art mesh, while consuming lower area. In addition, our routing scheme performs well under adversarial traffic as well. The fact that the our ring-based NoC runs significantly faster than the mesh contributes to these improvements. Moreover, since our proposed NoC designs are fully synchronous, they are very amenable to testing as well.

In the final part of this book, we explore an alternate scheme of achieving high-speed on-chip data transfer. Traditional pulse-based on-chip data transfer achieves a maximum data transfer rate of one bit per wire per clock cycle. Instead, we propose the use of sinusoidal signals of different frequencies as information carriers for on-chip data transfer. The key advantage of our method is the ability to superimpose such sinusoids and thereby effectively send multiple logic values along the same wire in a clock cycle. Experimental results show that for the same throughput as that of a traditional scheme, we require significantly fewer wires. The proposed sinusoidal data transfer scheme can be used for fast off-chip data transfer as well.

In summary, this book presents techniques to address the issue of chip-wide clock distribution, as well as the issue of designing a fast NoC in addition to exploring alternate schemes of achieving high-speed on-chip data transfer.

College Station, TX	Ayan Mandal
College Station, TX	Sunil P. Khatri
College Station, TX	Rabi N. Mahapatra
	September 2013

Acknowledgements

The authors would like to gratefully acknowledge Dr. Derek Chiou, Dr. Duncan M. Walker and Dr. Riccardo Bettati for their contribution to the work presented in this book. The authors would also like to acknowledge Dr. Nikhil Jayakumar and Kalyana Bollapalli who gave insightful comments on synchronous clock distribution and generation schemes.

Contents

Chapter 1
Introduction

Abstract In this chapter, we first define some terminology that we will be using throughout this book. Then, we talk about the common performance evaluation metrics for a Network-on-Chip (NoC), followed by its design aspects. Next, we motivate the need for designing a fast NoC. After this, we briefly introduce our high-speed clock distribution schemes followed by our fast source synchronous NoC design. Finally, we briefly introduce our work on sinusoidal based on-chip data transfer.

1.1 Terminology

- *VLSI:* Very Large Scale Integration (VLSI) refers to a methodology of designing integrated circuits (ICs) with a high level of functional integration.
- *CMP:* A Chip Multi-processor (CMP) consists of multiple computational cores integrated onto a single integrated circuit (IC) die.
- *PE:* A Processing Element (PE) is another name for a computational core in a CMP.
- *PTM:* A Predictive Technology Model consists of circuit level (SPICE) models for PMOS and NMOS transistors in a VLSI fabrication process, and is used for circuit-level projections of speed, power and other electrical characteristics of a proposed circuit design.
- *TWO:* A Traveling wave oscillator (TWO) is a resonant oscillator circuit. It comprises of a sufficiently long wiring ring, such that its capacitive and inductive parasitics result in a high frequency oscillatory network. Oscillations in this network are sustained by a plurality of inverter pairs spaced along the ring. The phase of the generated clock varies along the ring, and this is advantageous for Clock Data Recovery (CDR).
- *SWO:* Standing wave oscillator (SWO) is another type of resonant oscillator circuit. It comprises of a long wiring ring, and oscillations are sustained in this resonant ring by using a *single* inverter pair. By making a mobius connection at the end of the ring, the clock signal at any point in the ring is sinusoidal, and has the *same phase* at all points along the ring. An SWO is very useful in the design of synchronous integrated circuits (ICs).

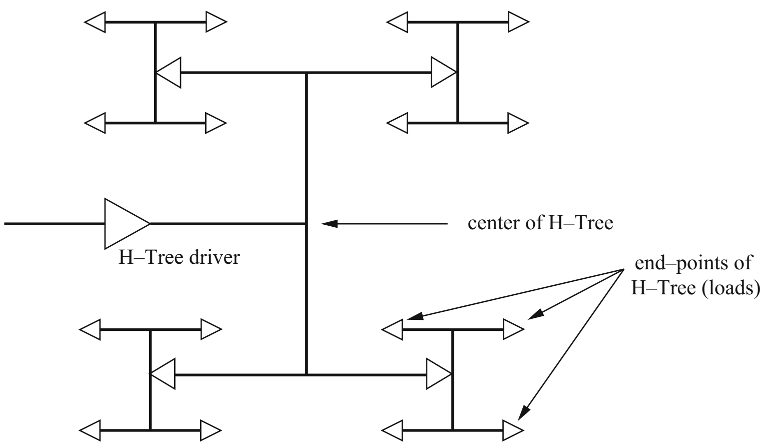


Fig. 1.1 H-Tree based clock distribution network

- *Q-factor:* An important parameter that determines the characteristics of a resonant oscillator is its quality factor (Q-factor). The Q-factor is a dimensionless quantity that determines how under-damped an oscillator is. High Q and low Q oscillators have their own advantages. High Q oscillators oscillate with a smaller frequency range, and have very high amplitudes at their resonant frequency. Such an oscillator would perform better by filtering out noise signals at frequencies other than the resonant frequency. On the other hand, the oscillation range is large for oscillators with low Q, which allows them to be useful in circuits which have a wider span of operating frequencies.
- *H-Trees:* The H-tree is a clock distribution network which is arranged in a recursive-H geometric pattern. Figure 1.1 shows a two level H-Tree. A H-tree that connect a central clock source to a multitude of clock loads (end-points), such that the electrical characteristics from the source to any load are nominally identical. Also, all the end points of the H-Tree are uniformly spaced with respect to the central clock driver. In practice, modern H-trees consist buffer circuits that are inserted symmetrically on each path from the source to the loads. This is done to ensure that the slew-rates of the clock signal at all the loads are sufficiently high. Hence all the end-points receive their clock signals after traveling through identical wire segments (in terms of length and width) and the same number of identical buffers (which are depicted as triangles in Fig. 1.1). A large buffer drives the clock signal to the center of the tree, while the length of the wires and sizes of the buffers can be progressively reduced as we traverse each path from the center of the H-tree to any end-point.
- *Skew:* Clock skew refers to the maximum difference in the clock arrival time between any two different end-points in a clock distribution network. Clock skew is graphically shown in Fig. 1.2. In this figure, $node1$ and $node2$ are two different end-points of the clock distribution network. Nominally, the clock is expected to arrive at $node1$ and $node2$ at the same time, but in practice these arrival times may

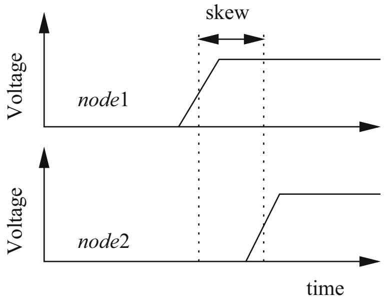


Fig. 1.2 Clock Skew

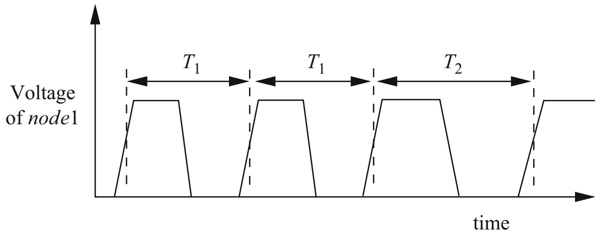


Fig. 1.3 Clock Jitter

differ. The main reasons for clock skew in a network are mismatches in device and interconnect, and temperature or voltage variations across the die. Nominally, we design a clock distribution network to have zero skew. A low value of clock skew is important for VLSI IC designs.

- *Jitter:* Jitter refers to the uncertainty in timing at a single end-point (clock load of the clock distribution network). Jitter is pictorially depicted in Fig. 1.3. In this figure, the clock period of *node*1 varies over time from a minimum value T_1 to a maximum value T_2, yielding a jitter of $T_2 - T_1$. Jitter is typically computed by measuring the difference between maximum and minimum clock period over a long duration of operation. Sources of jitter can be power supply noise or capacitive cross-talk induced noise. Another form of jitter measurement measures the difference in the clock period on a cycle-to-cycle basis.
- *PLL:* A PLL is a negative feedback control system that generates an output clock which is both phase and frequency locked to the input reference signal. A block diagram of a basic PLL is shown in Fig. 1.4. In this figure, *out* is the output clock generated by the PLL which is phase locked to *refclk* (which is the input clock). The *out* clock is frequency divided to yield a *divided_clk* signal, which is phase and frequency locked to *refclk*.

  Typical oscillators in a digital IC use some form of Voltage Controlled Oscillator (VCO) (Tasic et al. 2005; Megej et al. 2000; Hiroshi and Takaaki 1993; Thamsirianunt and Kwasniewski 1997; Hajimiri et al. 1999), and implement a PLL with the VCO in a closed-loop configuration (as depicted in Fig. 1.4). A

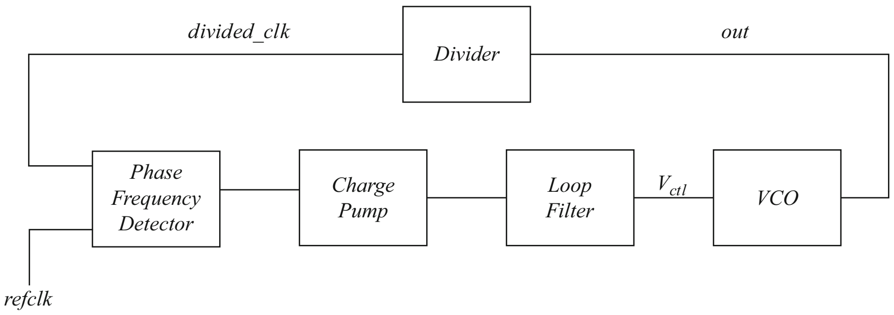
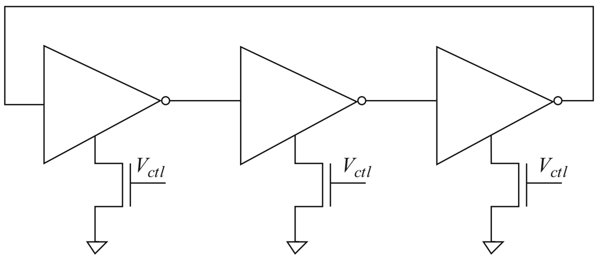


Fig. 1.4 Block diagram of a basic PLL design

Fig. 1.5 Analog voltage controlled oscillator (AVCO)



phase frequency detector (PFD) determines the phase error, and accordingly either speeds up or slows down the oscillation frequency of the VCO. VCOs are typically implemented in one of two ways

– *Analog* VCOs (ACOs) (Jovanovic and Stojcev 2006; Chen and Sheen 2002), in which a ring oscillator with a small (odd) number of inverters is typically used. This ring oscillator's frequency is modified by means of a voltage or current signal. Figure 1.5 shows an implementation of an Analog VCO. The frequency of oscillation of the ring oscillator is controlled by modulating the gate voltage (V_{ctl}) of the stacked NMOS transistors, thereby achieving a current-starved ring oscillator based ACO.
– *Digitally Controlled Oscillators* (DCOs) (Hiroshi and Takaaki 1993; Thamsirianunt and Kwasniewski 1997; Hajimiri et al. 1999), in which a large number of inverters are implemented such that inverter i drives the input to inverter $i + 1$. By closing the loop at the k^{th} inverter (where k is odd), the oscillator can be made to oscillate at variable (discrete) frequencies. The oscillator can be sped up or slowed down by decrementing or incrementing k respectively, using a control signal b_k which closes the loop at the k^{th} inverter. Figure 1.6 shows an example of a DCO implementation. The number of inverters that act as part of the ring is controlled by setting the values of $b_3, b_5, b_7, \ldots, b_k, \ldots, b_n$ in a one-hot fashion.

• *Asynchronous Communication:* A communication paradigm which has communicating islands which operate asynchronously.

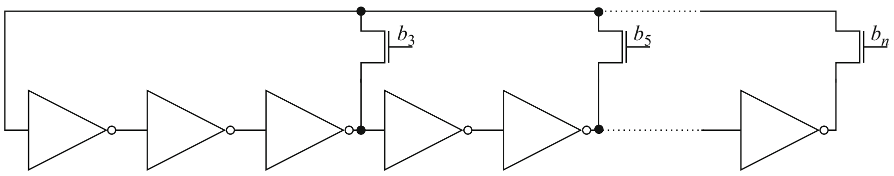


Fig. 1.6 Digitally controlled oscillator (DCO)

- *Mesochronous Communication:* A communication paradigm between two communicating islands which have the same frequency by not necessarily the same phase.
- *Multi-synchronous Communication:* A communication paradigm between two communicating islands which have different phase and frequency.
- *Synchronizer:* Synchonizers are used to reliably transfer data from one clock domain to another clock domain.
- *Packet:* A packet is the smallest unit of formatted data used to communicate between two PEs. A packet consists of two types of information: control (for example source/destination addresses) and data (for example memory read/write information).
- *Flit:* A flit is the unit of transfer supported by a NoC. A packet is typically broken into multiple flits.
- *Router:* A NoC router is a responsible for routing flits from a source to a destination. When a flit arrives at a router, it reads the address information in the flit to determine its final destination. Then, based on the routing policy, it directs the flit to the next router on its destination path.
- *Network Interface:* A network interface (NI) is a logical block which connects the PE with a NoC router. An NI is typically responsible for converting core messages to network packets and vice-versa.
- *Link:* A link is the physical interconnection between two routers and also between a router and an NI.
- *FIFO:* A First in first out (FIFO) buffer queues incoming data, and when requested, drives out the data that had arrived earliest.
- *Injection Rate:* The probability with which a PE injects a flit into the NoC every PE clock cycle.

1.2 Performance Evaluation Metrics for a Network-on-Chip

Multi-core processors require an efficient and scalable NoC infrastructure to handle the inter-processor on-chip communication needs. In this section, we discuss some common metrics that are used by the research community to evaluate NoC performance.

- **Throughput:** Throughput quantifies the rate at which data can be transferred across the NoC. It is defined as the average number of packets received per PE per clock cycle.
- **Latency:** Latency can be categorized in following two parts:
  - Network Interface latency: Latency incurred by the packet at the network interface before getting inserted into the communication fabric of the NoC.
  - Network latency: Latency incurred in the communication fabric from the point the packet is injected into the network till it reaches its destination.
- **Area:** The NoC area comprises of router (switch) and link area.
  - Router Area: The router is the most complex component of the NoC, providing routing, arbitration and flow control for network packets. Hence, the router typically accounts for a large part of the NoC area.
  - Link Area: The links are responsible for connecting routers to each other, in a manner dictated by the routing topology of the NoC.
    - Wiring area: The link width (flit width) and the physical dimensions of the wires determine the wiring area.
    - Repeater area: Long links may need multiple repeaters (buffers) along the path, thus adding to link area.
- **Power:** A significant amount of total chip power is consumed by the NoC (e.g. the NoC in the RAW (Taylor et al. 2002) system consumes as much as 36 % of the total power). Hence, power needs to be treated as a major design metric for NoC design. Power consists of 2 components:
  - Static Power: Static power consumption due to leakage currents is a significant contributor to total system power and is primarily technology dependent.
  - Dynamic Power: As a packet is transferred over links or stored in FIFOs, dynamic power is consumed as a result of a capacitive load being charged and discharged, as well as due to transient currents during transistor switching.
- **Reliability and Fault Tolerance:** The communication links connecting PEs are subject to failures. It is highly desirable to have a NoC design which can provide reliable data transfer under node or link failures.
- **Scalability:** Productivity, cost and strict time to market requirements demand a scalable NoC system design, in which performance metrics scale acceptably as the NoC size grows.
- **Quality of service:** Since NoC is a shared resource, it needs to implement support for Quality of Service (QoS). The essence of QoS is the ability to offer different levels of performance (in terms of latency or throughput) to different applications, with guarantees on performance bounds.
- **Deadlock Avoidance:** NoC routing is required to be deadlock free, otherwise it may lead to performance degradation or system failure.

1.3 Network-on-Chip Design Aspects

1.3.1 Topology

Conceptually, the simplest topology for an NoC is the crossbar, in which each PE has dedicated links to every other PE. However, the crossbar has a quadratic overhead, and as such is useful only for very small NoCs. The simplest and most ubiquitous NoC topology is the 2D mesh. Dally and Towles (2001) first proposed a 2D mesh as an NoC architecture. The topology consists of a 2D mesh of links, with switches at the intersections of horizontal and vertical links. Every switch has five ports, one connected to the local resource (PE) and the others connected to the closest neighboring switches. The torus architecture was proposed in Duato et al. (2002), with switches at the edges connected to the switches at the opposite edge through wrap-around channels. The long end-around connections can yield excessive delays which can be avoided by folding the torus Dally and Seitz (1986). The inherent disadvantage of the mesh or torus topologies is their large communication radius, resulting in large amounts of interconnect and large numbers of arbiters at the N-S-E-W crossings. These in turn leads to a large power consumption. Karim et al. (2002) proposed the Octagon architecture. The Octagon network consists of a basic *octagon unit* having eight nodes and 12 bidirectional links. It has a simpler implementation compared to the 2D mesh, with a higher throughput. Unlike the crossbar, the Octagon's implementation complexity increases linearly with the number of nodes. A 2D Flattened Butterfly was proposed in Kim et al. (2007). The 2D flattened butterfly is derived by flattening the routers in each row of a conventional butterfly topology and hence provides the connectivity of a mesh with additional links. The fat tree Leiserson (1985) connects routers in a tree manner, with sources and destinations at the leaves. The major advantage of the fat tree is the large amount of bandwidth available, with the downside of the requirement for large-radix routers toward the root of the tree. The Ring Samuelsson and Kumar (2004) topology implements concentric connected rings (similar to a ring road in city), which helps to reduce the risk of congestion in the central parts of the network.

1.3.2 Flow Control

Flow control refers to the policy of network resource allocation by a router as packets travel from source to destination. Next, we discuss three main flow control alternatives of an NoC.

- **Store-and-Forward:** Packets are fully buffered at each network node (router) they traverse. This technique, used in Sethuraman et al. (2005), incurs a high packet latency (directly proportional to packet size). Flow control is done at the packet level in this scheme.

- **Virtual Cut-Through (VCT):** VCT flow control allocates downstream buffer space and transmits a packet to the next router as soon as its header arrives, instead of waiting for the whole packet to be buffered as in store-and-forward flow control Kermani and Kleinrock (1979). In effect, VCT creates a circuit switched network. However, the buffer requirements are in terms of multiples of packet sizes for both the above approaches. In addition, the pre-allocation of downstream buffer space may cause increased delays for other traffic in the NoC. As in store-and-forward, flow control is done at the packet level in VCT.
- **Wormhole:** Dally and Seitz proposed a modification of the VCT by breaking packets up into flits, or flow control digits Dally and Seitz (1986). Flit based flow control, also known as wormhole routing, reduces the amount of storage required at each node in the network because buffers may be smaller than the size of a whole packet. In wormhole routing, only the first flit carries address information, which is used to reserve the route for the remaining flits in the data transmission. In effect, wormhole flow control is VCT at the flit level, and also effectively constructs a circuit-switched network.

  A NoC design which does not support wormhole routing is called flit-switched, where every flit of a packet is routed independently. Wormhole implementation increases the design complexity, thereby slowing down the NoC architecture. In this work, we focus our attention on a high-speed NoC design, and hence opt for flit-switched flow control. Flit-switched is not energy optimal as every flit needs to have the header information (for example source/destination addresses). However, in our NoC design, we have quantified this overhead to be minimal. Flit-switched flow control also has a negative impact on the packet latency. Since each flit is routed independently, the packet delay equals the worst delay faced by any flit. Since our NoC design runs at a much higher clock than the PEs, the overall latency incurred by the flits, as validated by our experiments, is minimal. Flit-switched flow control increases the network interface (NI) complexity. At the receiver side, flits can arrive out of order, and the corresponding NI requires complex logic to reconstruct the packet from the flits. This can be achieved by using a re-order buffer (Kwon et al. 2009) in the NI.

1.3.3 Routing

The routing mechanism determines the path taken by a packet from source to destination. The route can be determined either at the source node (Source Routing) or independently across the routers of the network (Distributed Routing). Source routing does not provide path diversity. Another orthogonal classification of routing schemes is based on adaptivity of the routes to network conditions such as congestion. Thus, routing schemes can be classified as:

- **Deterministic routing:** The route between a source and destination always stays the same, and is independent of the network state. Dimension Order Routing

(DOR) is a widely used, low complexity, deadlock-free routing mechanism. DOR chooses a specific dimension order (x followed by y, or vice versa) and routes packets using this dimension order, with at most one turn in their route. DOR has been used in many experimental and commercial interconnection networks (Taylor et al. 2002; Lenoski et al. 1992).

- **Adaptive Routing:** In adaptive routing, the paths taken will vary with network conditions such as congestion. However, routers incur additional complexity to support this mechanism. Adaptive routing is often used to reduce congestion or avoid network faults. Adaptive routing has been used in commercial systems like (Muller et al. 1998; Adiga et al. 2005).

1.3.4 Arbitration

Within a router, arbitration is required while servicing conflicting requests. When multiple input ports request a common output port, the requested output port determines the order in which requests are serviced. There are two types of arbitration schemes in use:

- **Static:** The arbitration is predetermined and does not depend on the state of the router. Such a scheme may not be able to guarantee fairness.
- **Dynamic:** The arbitration depends on the state of the router. The Round-Robin scheme is a popular fair dynamic arbitration approach. Dynamic arbitration can also provide QoS guarantees to certain classes of applications.

1.3.5 Buffering

At the intermediate nodes (routers), packets need to be temporarily stored (buffered), while waiting for channel access. Buffering is classified under following two categories:

- **Input Buffering:** Packets are buffered only at the input port. This prevents additional requests from upstream routers when the input buffer is full. Head-of-Line (HOL) blocking, in which an upstream router may not be able to send requests, despite the fact that output links are unoccupied.
- **Output Buffering:** Packets are buffered at the output port contending for the output link. With this style of buffering HOL blocking is avoided, thereby decreasing the average latency of the packets.

  An NoC design can use both input and output buffering.

1.4 The Need for a Fast Network-on-Chip Design

The number of PEs on chip multiprocessors (CMPs) continues to grow as VLSI fabrication technology scales. Hence it is becoming increasingly difficult to connect the different PEs of a CMP in a scalable and efficient way. Traditional shared bus based communication suffers from poor scalability, high arbitration complexity and low bandwidth. Adding more PEs to a shared bus also adds parasitic capacitance, degrading electrical performance as well. The arbitration delay in a shared bus also grows with number of PEs. The available bandwidth is limited and shared by all PEs attached to the shared bus.

The NoC paradigm has emerged as an alternative to the traditional shared bus by improving the bandwidth, power efficiency and scalability of on-chip communication. All links in an NoC can be simultaneously used for data transmission, which provides a high level of parallelism. The NoC also enables pipelining of data, providing a much greater aggregate bandwidth than shared buses.

A typical NoC design is modular, with basic units of network interfaces, routers and links. This reduces design complexity as the above units can be designed once and then replicated across the chip. The NoC architecture can also improve design productivity by serving as a reusable communication sub-system across chip generations.

Since NoCs connect PEs across the chip, it becomes difficult to operate the NoC at a high frequency. One reason for this is the difficulty of distributing a high speed, low power clock across the chip. Moreover, most of the existing NoC designs use complex routers which slows down the NoC clock. A router provides connectivity to neighboring routers with the help of links, and usually has multiple input and output ports. Hence the router has to manage several complex tasks such as arbitration of output links, buffering and flow control of network data. A pipelined router architecture achieves a higher frequency at the cost of higher latency, but does not address the issue of router complexity.

1.5 Clock Distribution for fast Networks-on-Chip

As mentioned in the previous section, high speed clock distribution is a major challenge in designing a high speed NoC. In this work, we propose several techniques to address the issue of distributing a high-speed, low power, clock for the NoC and the PEs. We mainly focus our attention on resonant ring-based oscillators which have recently emerged as a promising technique for high-speed, low power clock generation.

Traditionally, clock distribution networks have been optimized to minimize end-to-end delay of the distribution network. However, since most ICs have an on-chip PLL, we argue that the design goal of minimizing end-to-end jitter is more relevant. Hence, we focus on minimizing jitter for clock distribution networks.

We present a novel low-jitter phase-locked clock generation and distribution methodology which uses resonant standing wave oscillators (SWOs). The same clock signal is used to serve both the NoC and the PEs. We also present an all-digital control loop to phase-lock the SWO to an external reference clock. Experimental results demonstrate that the jitter of our approach is dramatically lower (by $\sim 4.6\times$) than existing schemes, while the power consumption is significantly lower (by $\sim 2\times$) as well.

We also present a dynamic programming based approach to synthesize a minimum cost buffered H-tree for distributing a clock signal to the PEs. Our primary goal is to minimize the end-to-end jitter of the synthesized H-tree, while the secondary goal is to minimize power as well. Compared to a manually constructed buffered H-tree network, our approaches are able to reduce both jitter (by as much as 28 %), and power (by as much as 46 %).

We also propose a tiled SWO design to distribute a high speed synchronous resonant clock across the chip, and demonstrate that an SWO approach can be used to practically implement a high-frequency, low-power clocking approach with high and uniform area coverage over an IC. Our design also allows the PEs to extract a low jitter clock directly from the high speed resonant clock by division. Our simulations indicate that this tiled structure can oscillate at about 7.25 GHz, with low power (about 68 mW per SWO tile) and low jitter (about 3.1 % of the nominal clock period).

1.6 Source Synchronous Network-on-Chip Design

Using the high-speed, low power resonant clock introduced in the previous section, we propose a novel high-speed NoC design that relies on source synchronous data transfer over a ring. The source-synchronous data rings are interconnected to provide complete connectivity across the CMP. The source synchronous ring data is clocked by the standing wave resonant clock (which is routed parallel to the data), which operates significantly faster than the PEs that are served by the ring. The PEs are connected to the fast data ring and can inject data into the ring and extract data from the ring.

We implement a deadlock-free routing protocol for the source-synchronous ring-based NoC. Architectural results obtained on synthetic and real traffic demonstrate that the source-synchronous ring-based NoC has significantly lower latency and higher maximum sustained injection rate compared to a state of the art mesh-based NoC. In addition, we show that our routing scheme performs well for adversarial traffic as well.

We also propose a modified source-synchronous design where the clock and data NoC are routed in parallel, with the PEs extracting a low jitter clock directly from the high speed ring clock by division. The PEs are thereby synchronous with the NoC, yielding improved latencies since synchronization are not required. Using the above modification, we first propose a class of source-synchronous NoCs organized in an H-tree pattern which use fewer buffers and lower wire length compared to a state of the art mesh. Architectural simulations on synthetic and real traffic show that our H-tree based NoC designs can provide significantly lower latency and are able to sustain a higher injection rate compared to a state of the art mesh. Using the synchronous modification, we also evaluate two additional floorplan-friendly NoC topologies, which use fewer buffers and lower wire length compared to a state of the art mesh. Architectural simulations on synthetic and real traffic show that these NoC topologies can provide significantly lower latency while achieving same or better maximum sustained injection rate compared to a state of the art mesh. Since our proposed NoC designs are fully synchronous, they are very amenable to testing. When multiple clock domains are involved in a multi-synchronous paradigm, there are uncertainties involved in terms of event scheduling during testing. Since our design are fully synchronous, these uncertainties are eliminated.

1.7 Fast On-chip Data Transfer Using Sinusoid Signals

There is a significant incentive to improve the speed of on-chip communication. It is well known that ICs have substantially followed Moore's Law, doubling their speed and/or complexity approximately every two years. In contrast, data communication rates between the processing cores within an IC, and also between ICs have improved less rapidly, proving to be a bottleneck in the quest for faster computing. The fundamental reason for this is that data is traditionally communicated as a sequence of pulses. With such a choice, noise and signal are parallel vectors, making fast reliable data transfer difficult. For this reason, on-chip data transfer achieves a maximum data transfer rate of one bit per wire per clock cycle. In this work, we explore the use of sinusoidal signals of different frequencies as information carriers for on-chip data transfer. The advantage of our method is the ability to superimpose such sinusoids and thereby effectively send multiple logic values along the same wire in a clock cycle. Experimental results show that for the same throughput as traditional scheme, we require significantly fewer wires. Our sinusoidal data transfer scheme can be used for efficient off-chip data transfer as well.

References

N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue Gene/L torus interconnection network," *IBM J. Res. Dev.*, vol. 49, no. 2, pp. 265–276, Mar. 2005.

O. T.-C. Chen and R. R.-B. Sheen, "A power-efficient wide-range phase-locked loop," *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 1, pp. 51–62, Jan 2002.

W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *The Journal of Distributed Computing*, vol. 1(3), pp. 187–196, 1986.

W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 684–689.

Jose Duato, Sudhakar Yalamanchili, and Ni Lionel, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

A. Hajimiri, S. Limotyrakis, and T. H. Lee, "Jitter and phase noise in ring oscillators," *Solid-State Circuits, IEEE Journal of*, vol. 34, no. 6, pp. 790–804, Jun 1999.

Yamagata Hiroshi, Yamada Takaaki, "Digital voltage controlled oscillator having a ring oscillator with selectable output taps," August 1993.

G. S. Jovanovic and M. K. Stojcev, "Current starved delay element with symmetric load," *International Journal of Electronics*, vol. 93, no. 3, pp. 167–175, March 2006.

F. Karim, A. Nguyen, and S. Dey, "An interconnect architecture for networking systems on chips," *Micro, IEEE*, vol. 22, no. 5, pp. 36–45, Sep/Oct 2002.

Parviz Kermani and Leonard Kleinrock, "Virtual cut-through: a new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.

J. Kim, J. Balfour, and W. J. Dally, "Flattened butterfly topology for on-chip networks," *Computer Architecture Letters*, vol. 6, no. 2, pp. 37–40, Feb. 2007.

Woo-Cheol Kwon, Sungjoo Yoo, Junhyung Um, and Seh-Woong Jeong, "In-network reorder buffer to improve overall noc performance while resolving the in-order requirement problem," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, April 2009, pp. 1058–1063.

Charles E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, pp. 892–901, October 1985.

Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, Monica S. Lam, and Dash The Ease of use, "The Stanford DASH multiprocessor", *IEEE Computer*, 1992, 25, pp. 63–79.

A. Megej, K. Beilenhoff, and H. L. Hartnagel, "Fully monolithically integrated feedback voltage controlled oscillator [using PHEMTs]," *Microwave and Guided Wave Letters, IEEE*, vol. 10, no. 6, pp. 239–241, Jun 2000.

Matthias M. Muller, Thomas M. Warschko, and Walter F. Tichy, "Prefetching on the Cray-T3E," in *Proceedings of the 12th international conference on Supercomputing*, New York, NY, USA, 1998, ICS '98, pp. 361–368, ACM.

H. Samuelsson and S. Kumar, "Ring Road NoC architecture," in *Norchip*, 2004, pp. 16–19.

Balasubramanian Sethuraman, Prasun Bhattacharya, Jawad Khan, and Ranga Vemuri, "Lipar: A light-weight parallel router for fpga-based networks-on-chip," in *Proceedings of the 15th ACM Great Lakes symposium on VLSI*, New York, NY, USA, 2005, GLSVLSI '05, pp. 452–457, ACM.

A. Tasic, W. A. Serdijn, and J. R. Long, "Adaptivity of voltage-controlled oscillators—theory and design," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 5, pp. 894–901, May 2005.

Michael Taylor, Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal, "Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures," in *In International Symposium on High Performance Computer Architecture*, 2002, pp. 341–353.

M. Thamsirianunt and T. A. Kwasniewski, "CMOS VCO's for PLL frequency synthesis in GHz digital mobile radio communications," *Solid-State Circuits, IEEE Journal of*, vol. 32, no. 10, pp. 1511–1524, Oct 1997.

Chapter 2
Clock Distribution for Fast Networks-on-Chip

Abstract As mentioned in first Chapter, high speed, low-jitter and low-power clock distribution is a major challenge in designing a fast Network-on-Chip (NoC). In this chapter, we propose several techniques to address the issue of distributing a high-speed, low power, low jitter clock across an IC. We focus our attention primarily on resonant ring-based standing wave oscillators (SWOs) which have recently emerged as a promising technique for high-speed, low power clock generation. In Sect. 2.1, we describe the basics of resonant ring-based oscillators. In Sect. 2.2, we present a novel low-jitter, low-power clock generation and distribution methodology which uses resonant standing wave oscillators (SWOs). We also present an all-digital control loop to phase-lock the SWOs to an external reference clock. Experimental results demonstrate that the jitter of our approach is dramatically lower (by $\sim 4.6\times$) than existing schemes, while the power consumption is significantly lower (by $\sim 2\times$) as well. Next, in Sect. 2.3, we present a dynamic programming based approach to synthesize a minimum cost buffered H-tree for clock distribution. Our primary goal is to minimize the end-to-end jitter of the synthesized H-tree, while the secondary goal is to minimize power as well. Compared to a manually constructed buffered H-tree network, our approach is able to reduce both jitter (by as much as 28 %, and power by as much as 46 %). Finally, in Sect. 2.4, we present a tiled SWO structure with a plurality of ring-based SWO tiles arranged in a 2D fashion, oscillating at a high frequency. We validate that in this manner, an SWO approach can be used to practically implement a high-frequency, low-power clocking approach with high and uniform area coverage over an IC. Our simulations indicate that this tiled structure can oscillate at about 7.25 GHz, with low power (about 68 mW per SWO tile) and low jitter (about 3.1 % of the nominal clock period)

2.1 Resonant Oscillators

Recently, there has been some interest in mobius ring based resonant oscillators as a means to generate the clock signal for digital ICs. These resonant oscillators use a metallic ring along with a single (or multiple) cross coupled inverter pairs(s). Since charge is recirculated in these configurations, they exhibit a low power consumption. Resistive losses in the ring, as well as the power consumed by the inverter pair(s) contribute to the power consumption of these structures. By choosing the length of the ring carefully, oscillations of high frequencies can be sustained, as long as

DOI 10.1007/978-1-4614-9405-8_2, © Springer Science+Business Media New York 2014

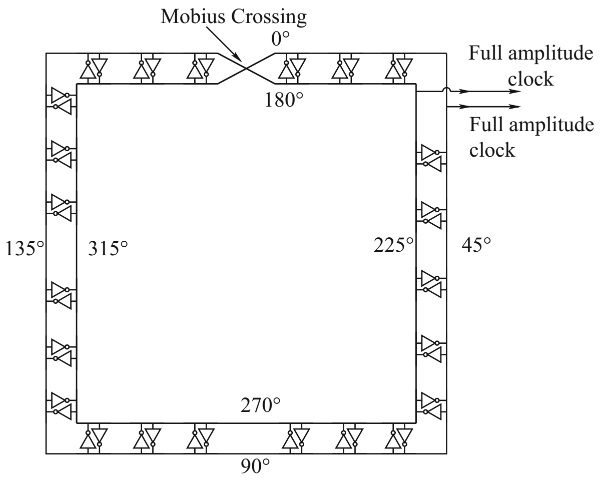


Fig. 2.1 Circuit Topology for traveling wave resonant oscillator

the inverter pair(s) can switch at these frequencies. Ring based resonant oscillators can be categorized into following two types: Traveling Wave Resonant Oscillators (TWOs) and Standing Wave Resonant Oscillators (SWOs).

2.1.1 Traveling Wave Oscillators

A *traveling wave* resonant oscillator circuit (also referred to as a *rotary clock*) was introduced in (Wood et al. 2001; 2006). The authors utilize a sufficiently long wiring ring, such that its capacitive and inductive parasitics result in a high frequency oscillatory network. This resonant clock topology is described in Fig. 2.1. Figure 2.2 shows the (overlayed) waveforms of the clock signals extracted from different locations along the ring. Oscillations in this network are sustained by a plurality of inverter pairs uniformly spaced along the ring (Fig. 2.1). However, a key drawback of the rotary clock is that the phase of the generated clock varies along the ring (as shown in Fig. 2.2), making traditional synchronous clock based design extremely difficult. Also, the clock signal at every point of the ring is a full-rail signal, resulting in a larger power consumption.

2.1.2 Standing Wave Oscillators

A *standing wave* resonant oscillator circuit was proposed in Cordero and Khatri 2008. In this approach, a long wiring ring is used, and oscillations are sustained in this resonfant ring by just using a *single* inverter pair (Fig. 2.3). The clock signal at any point in the ring is sinusoidal, and has the *same phase* at all points along the

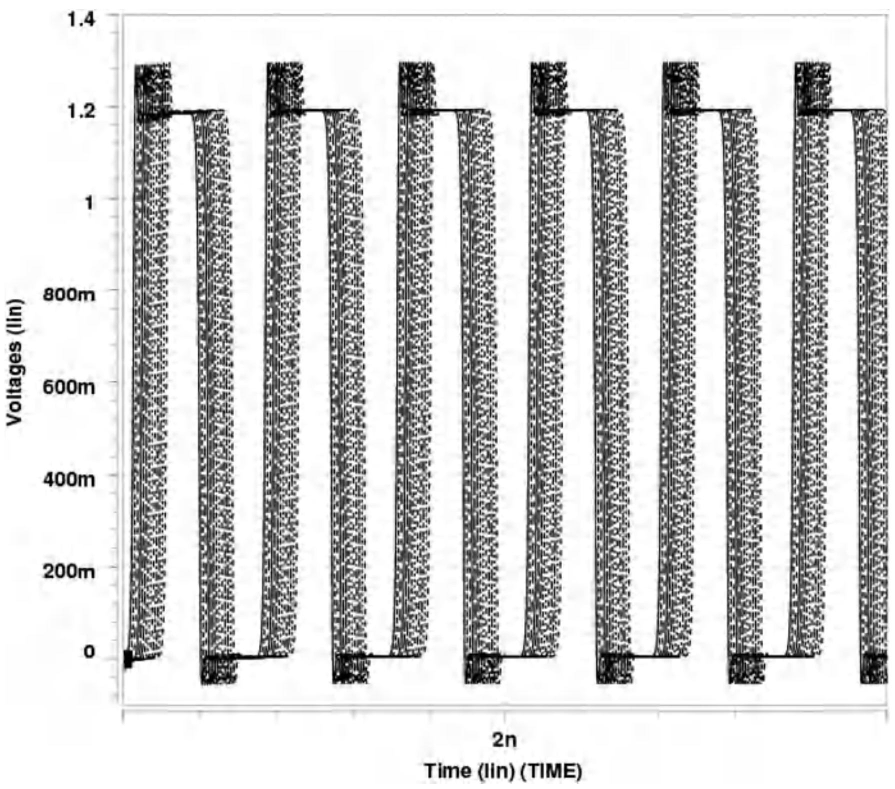


Fig. 2.2 Sample waveforms (Overlaid) for traveling wave resonant oscillators

ring as shown in Fig. 2.4. The SWO consists of a wire ring, with a mobius crossing (shown at the top). An inverter pair is inserted at the mobius crossing location (see Fig. 2.3) to provide the negative resistance necessary to sustain oscillations. At any point in the ring, the outer wire sustains a sinusoidal oscillation of the same phase, while the inner wire sustains a sinusoidal oscillation as well, but of the opposite phase. The point diametrically opposite to the mobius crossing has a zero amplitude and is a virtual ground point (in an AC sense). A full-amplitude square wave clock is recovered at any point in the ring by using a clock recovery circuit which consists of a differential amplifier.

By using differential amplifiers at different points in the ring, full rail clock signals are extracted at the locations desired (Fig. 2.5). Thus, this approach yields full-amplitude square wave clock signals that have the same phase everywhere along the ring. This is a key improvement over the rotary clock of (Lin and Kaiser 2001), since it is compatible with synchronous IC design. In addition, the reduced ring capacitance due to the use of significantly fewer inverters (in particular, just one), increases the operating speed and reduces the power consumption as well. Note that there is an AC null (virtual "zero") point in the center of the ring as shown in Fig. 2.3.

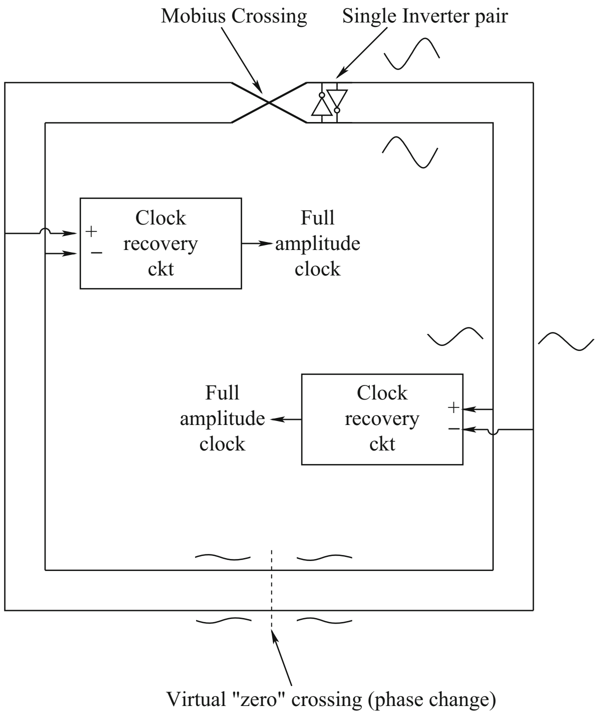


Fig. 2.3 Standing wave resonant clock

As a result, the phases of the signals on the right and the left of the null point are 180° apart. Therefore, clock recovery circuits on the left have their connections reversed compared to recovery circuits on the right of the null point. Note that clock recovery is not performed near the null point, since the signal amplitude is very low near the null point. Both Figs. 2.4 and 2.5 were obtained using the same simulation conditions that were used in (Cordero and Khatri 2008).

2.2 Phase Locked Clock Generation and Distribution Using SWOs

As mentioned in the previous section, an SWO has following advantages over a TWO: (1) SWO yields clock signals with same phase everywhere along the ring, and hence can be naturally used as a synchronous clock distribution scheme; (2) SWO uses significantly fewer inverters compared to a TWO, which results in an increasing operating speed as well as reduced power consumption. Hence, we focus our attention on an SWO-based design. In the following section, we propose a phase-locked SWO-based high-speed, low power clock generation and distribution scheme.

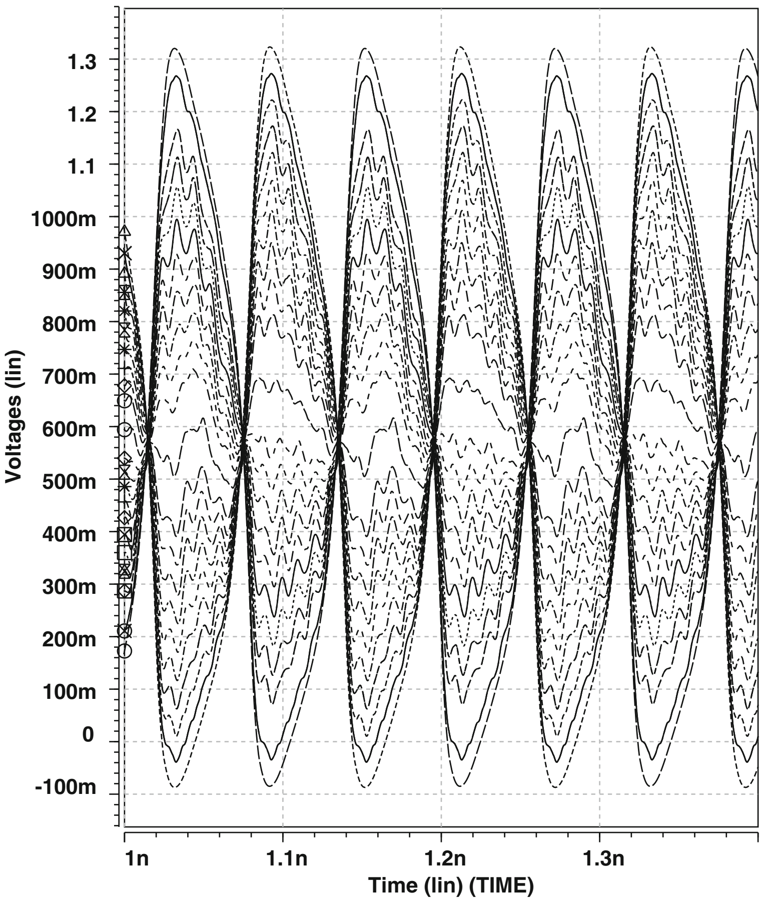


Fig. 2.4 Sample waveforms (Overlaid) for standing wave oscillator

2.2.1 Introduction

Clock generation and distribution are critically important aspects of VLSI IC design. For large digital ICs, the clock signal can contribute significantly to the power consumption of the IC. Additionally, process, voltage and temperature (PVT) variations can result in a variation in the clock arrival time at the different sinks (end-points) of the clock distribution network. This can lead to systematic skew in the clock arrival times at the different sinks, and also a dynamic variation (jitter) in the clock arrival times at the sinks due to cycle-to-cycle variations in the power supply signal at different locations in the die. A good phase-locked clock generation and distribution scheme needs to minimize the power consumption, systematic skew as well as the cycle-to-cycle jitter. Traditionally clock distribution networks have been designed using end-to-end delay of the clock signal as the key metric to minimize. We believe

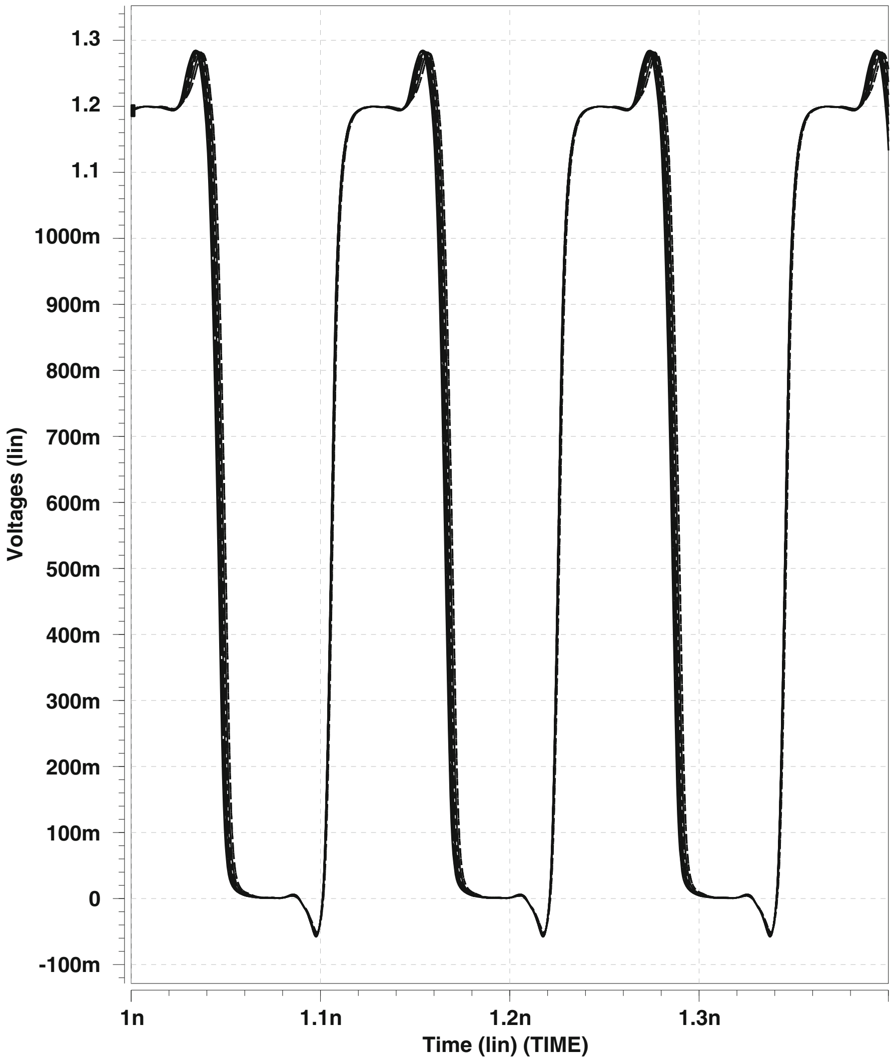


Fig. 2.5 Recovered clock waveforms (Overlaid) for standing wave oscillator

that since most ICs have an on-chip phase-locked loop (PLL), the relevant metric to minimize is instead the jitter and the systematic skew.

In this section, we present a resonant standing wave oscillator (SWO) based phase-locked clock generation and distribution scheme. Given that the scheme is resonant in nature, it exhibits a significantly lower power consumption than a traditional buffered H-tree.

Additionally, since the resonant oscillator has a relatively high Q-factor (of about 5), our scheme has superior systematic skew and jitter characteristics as well. On the other hand, a buffered H-tree exhibits poor jitter characteristics since the buffers on any path exhibit delay variations due to local cycle-to-cycle power supply variations.

The SWO in our work is based on a ring-based oscillator shown in Fig. 2.3. Our initial experiments with the above structure indicated that the jitter characteristics of the ring-based SWO were very good. As a result, we design an all-digital control loop to vary and phase-lock the SWO frequency to an external reference, and also use the same ring topology to distribute the clock all over the IC.

Our phase-locked SWO is controlled by an all-digital control loop, which consists of coarse as well as fine frequency control. Coarse frequency control is accomplished by varying the number of oscillating wires in a bundle of wires (instead of just 2 wires as shown in Fig. 2.3). Fine frequency control is also accomplished in a digital manner, using a bank of binary-weighted capacitors connected at the inverter pairs of the ring. We present the analysis and circuit simulation results that validate the correct phase-locking behavior of the SWO-based, all-digitally controlled PLL. In addition, we utilize the same ring-based structure to distribute the clock signal all over the IC as well, in a "comb" like clock distribution topology. Our proposed clock distribution is entirely resonant in nature, and hence consumes low power. Because of the relatively high Q-factor of the SWO, jitter and systematic skew are reduced as well.

The complete SWO-based phase-locked clock generation and distribution scheme has been simulated in HSPICE (Inc Meta-Software) using a 32 nm (PTM 2013) technology. The key contributions of this work are:

- We present an all-digital control loop to phase-lock the SWO to an external clock, by combining a coarse and fine control loop over the frequency of interest (which is between $\sim$3.1 GHz to $\sim$3.6 GHz). Our phase-locking circuit does not require any external analog supply voltages for operation.
- The phase-locked SWO is also used as a clock distribution mechanism, by routing the SWO wires in a "comb" like manner to all the clock distribution end-points on the die.
- Due to the resonant nature of the clock generation and distribution scheme, a reduced power consumption (upwards of $2\times$ lower) is achieved.
- Additionally, the jitter and systematic skew characteristics of our clock generation and distribution scheme are significantly (about $5\times$) better than those of a traditional buffered H-tree scheme with an overlaid mesh.

The remainder of this section is organized as follows: We discuss some previous work for clock design in Sect. 2.2.2, while Sect. 2.2.3 provides the details of our SWO-based clock generation and distribution methodology. Section 2.2.4 presents results from experiments which we conducted, while Sect. 2.2.5 presents conclusions.

2.2.2 Previous Work

Recently, there has been some interest in mobius ring based resonant oscillators as a means to generate the clock signal for digital ICs. Both traveling wave (Wood et al. 2001; Chan et al. 2004; MultiGig 2013; Nedovic et al. 2007 and standing wave Cordero and Khatri 2008; Karkala et al. 2009) oscillators have been proposed in the literature. Since charge is recirculated in these configurations, they exhibit a low power consumption. Resistive losses in the ring, as well as the power consumed by the inverter pair(s) contribute to the power consumption of these structures. By choosing the length of the ring carefully, oscillations of high frequencies can be sustained, as long as the inverter pair(s) can switch at these frequencies.

A TWO (referred to as a *rotary clock*) was described and implemented (Wood et al. 2001; 2006). The topology is similar to SWO (Cordero and Khatri 2008; Karkala et al. 2009), except that oscillations in this network are sustained by a plurality of inverter pairs spaced along the ring. The key drawback of the rotary clock is that the phase of the generated clock varies along the ring, making traditional synchronous clock based design extremely difficult. Also, the clock signal at every point of the ring is a full-rail signal, resulting in a larger power consumption. Hence, we focus our attention on a SWO-based design.

Both the traveling wave and standing wave oscillators generate a free-running clock signal. In practice, however, it is crucial that any oscillator in a digital system has the ability to modify its phase and frequency in a predictable manner, so as to allow it to be integrated into a PLL. In this section, we design a SWO-based PLL as well as a clock distribution network, with minimal power, jitter and systematic skew.

The authors of (Nedovic et al. 2007) describe a 2.5 GHz PLL using a traveling wave oscillator. The design recovers 16 bits within a clock period. Though it is advantageous for Clock Data Recovery (CDR), the varying phase of the traveling wave clock makes synchronous clock based design difficult. In contrast, our work uses an SWO-based PLL, and simultaneously accomplishes resonant clock distribution as well, with low power, jitter and skew.

In Mahony et al. (2003), a high-frequency standing wave PLL was proposed. It uses multiple coupled oscillators, each comprised of an NMOS cross-coupled pair to sustain the oscillation, and a PMOS diode connected load for setting the common mode voltage. Unlike our approach, (Mahony et al. 2003) achieves a very small (6.4 %) locking range (while we achieve a $\sim$ 15 % locking range from $\sim$3.1 GHz to $\sim$3.6 GHz). Further, the approach of (Mahony et al. 2003) does not focus on jitter, systematic skew or power, and does not concern itself with clock distribution, unlike our approach.

PLLs have been important blocks in the field of VLSI for the past few decades. Several PLLs have been designed with a coarse and a fine tuning approach. For example, the authors of (Lin and Kaiser 2001; Wilson et al. 2000; Lin and Lai 2007) have implemented a PLL using a combination of an analog and digital control loop for coarse and fine tuning. However, none of the oscillators in (Lin and Kaiser 2001;

Wilson et al. 2000, Lin and Lai 2007) were resonant. Also, in further contrast, our work uses an all-digital control, and provides resonant clock distribution as well.

Recently, in (Karkala et al. 2009), a SWO-based PLL was presented with a digitally controlled coarse frequency loop, and an analog control for the fine frequency loop. Although we borrow the coarse control approach from (Karkala et al. 2009), there are significant differences between this work and (Karkala et al. 2009). In particular, (Karkala et al. 2009) utilizes an extra 2·VDD supply for the analog control of the fine frequency, which we avoid in our all-digital control approach. Also, (Karkala et al. 2009) does not address the problem of clock distribution, while our design performs resonant clock distribution as well. Without using a resonant clock distribution, the jitter and systematic skew could be significant (if distribution was done by traditional means such as a H-tree). Additionally, unlike (Karkala et al. 2009), we compare our approach with a H-tree based clock distribution approach with a mesh overlay, and quantify the gains in power (more than 2× lower), systematic skew (about 5.5× lower) and jitter (about 4.6× lower).

In Chueh et al. (2004), the authors experimented with a 4-level H-tree, with energy-recovering flip-flops and a resonant clock distribution. The resonant clock generator was operated at 250 MHz. By varying the width and spacing of the wires, they evaluated the clock skew at different leaf nodes. Unlike our approach, they did not focus on a PLL, and rely on a small design (4-level H-tree, 4 mm die size) for their experiments. Despite this, their best case reported skew was larger than ours, even with significantly wider distribution wires that were spaced much further apart. Also, no discussion or comparison of jitter was provided in (Chueh et al. 2004).

2.2.3 *Our Approach*

In the following section, we first briefly discuss our proposed resonant oscillator based phase-locked clock generation and distribution scheme, followed by a brief discussion of the H-tree based clock distribution scheme (with an overlaid mesh) we compare our work with. Next, we discuss the details of our all-digital PLL to phase lock the resonant clock with an external reference.

2.2.3.1 SWO-based Clock Distribution

Our clock distribution scheme is based on resonant standing wave oscillators (SWOs). The physical topology of our SWO is shown in Fig. 2.3, while the equivalent circuit for our SWO is shown in Fig. 2.6. In this figure, L_w and C_w refer to the parasitic inductance and parasitic capacitance of the ring wires respectively. The capacitance due to the cross-coupled inverter pair (i.e. twice the sum of the diffusion and gate capacitances of any inverter in the pair) is C. Since C and C_w are in parallel, we obtain the equivalent circuit shown in Fig. 2.6.

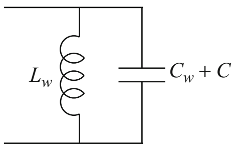


Fig. 2.6 Equivalent circuit for our resonant oscillator

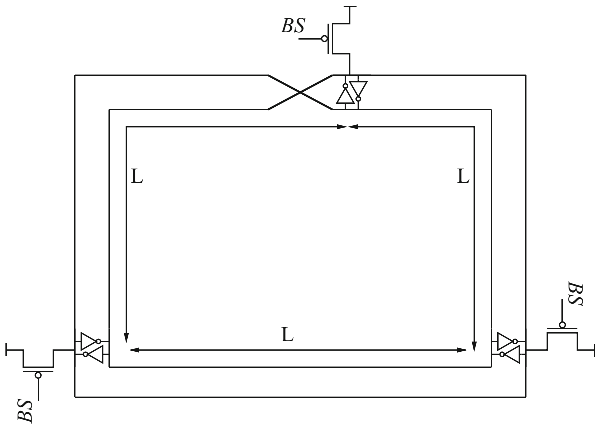


Fig. 2.7 Longer SWO

The oscillation frequency of the equivalent circuit is given by

$$f = \frac{1}{2\Pi\sqrt{L_w(C + C_w)}} \tag{2.1}$$

For reasonable oscillation frequencies, the area covered by an SWO is very small compared to the area of the typical digital IC. Hence, in order to realize a SWO based clock distribution scheme, we need to increase the size of the ring to cover a larger area of the chip. However, increasing the length of the ring decreases the oscillation frequency (due to the increase in its parasitic capacitance and inductance). Figure 2.7 shows a SWO consisting of three cross-coupled inverter pairs, placed at an equal distance from each other. The SWO of Fig. 2.7, with a length of $3 \times L$, oscillates at the same frequency of a single SWO of length L with one cross coupled pair of inverters. The SWO is bootstrapped with PMOS devices (using a BS signal) as shown in the Fig. 2.7 to provide the initial condition for the system to oscillate. In this manner, the SWO in Fig. 2.7 covers $\sqrt{3}$ times the chip area as compared with a SWO with just one inverter pair, and oscillates at the same frequency. In general, we can implement an SWO with a large perimeter $k \times L$ (where k is odd). To ensure that the clock is distributed to 2^P uniformly spaced points on the die, we propose to snake the wires of the ring to implement a "comb" topology as shown in Fig. 2.8. In this figure, we assume $P = 6$. Since the SWO has the same phase everywhere along the ring, this "comb" structure is able to distribute the clock signal to the 64 sinks on the IC with the same phase. Note that for such a SWO design, we require an odd number k of inverter pairs, and a single mobius connection (as shown in Fig. 2.7).

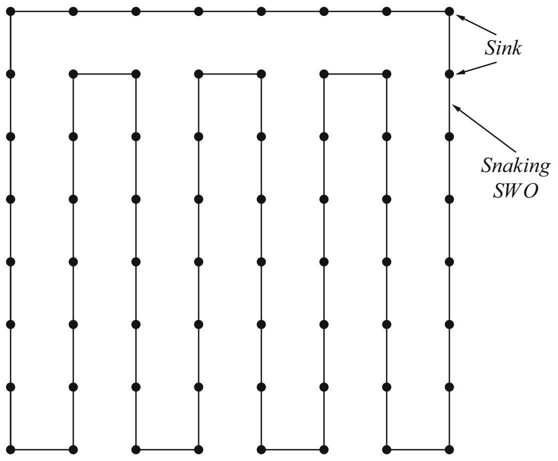


Fig. 2.8 Comb topology

Fig. 2.9 Coarse frequency configuration

x_0	x_1	x_2	x_2	x_1	x_0	
1	1	1	1	1	1	← Coarse config 1
1	1	0	0	1	1	← Coarse config 2
1	0	0	0	0	1	← Coarse config 3
outer ring			inner ring			

2.2.3.2 Phase-locked SWO

We realize a phase-locked SWO by modifying the base design of SWO (Cordero and Khatri 2008). Frequency control of the SWO consists of coarse frequency control as well as fine frequency control, as discussed next.

Coarse Frequency Control: We implement coarse frequency control by modulating the inductance and capacitance of the ring. The two wires of the ring (Fig. 2.3) are replaced by $2n$ parallel wires in our approach, just as in (Karkala et al. 2009). We realize a variable frequency SWO by selecting a subset (of even cardinality) of the $2n$ wires for oscillation. The wires used for oscillation are selected symmetrically about the center of the $2n$-wire bundle. Each subset of the $2n$ wires that we use for oscillation is referred to as a *coarse configuration*.

Figure 2.9 illustrates the coarse configurations for $n = 3$, while Fig. 2.10 illustrates the ring circuit structure for $n = 3$, at the mobius crossing point. In Fig. 2.9, x_0 refers to the outermost wire of the outer ring as well as the innermost wire of the inner ring, while x_1 refers to the middle wire, and x_2 refers to the innermost wire of the outer ring, and the outermost wire of the inner ring. In this figure, a ‘1’ against x_i indicates that the corresponding wires are oscillating, and a ‘0’ indicates that the corresponding wires do not oscillate. Therefore, coarse configuration 1 (2) uses a total of 6 (4) wires for oscillation (indicated by the values ‘1’ against the configuration). Note that the

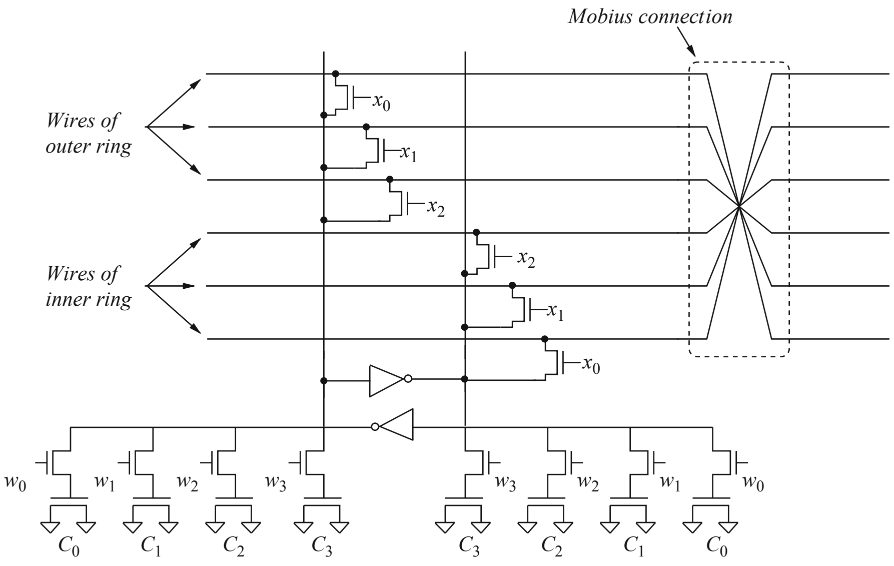


Fig. 2.10 SWO coarse and fine configuration

oscillating wires in a coarse configuration are chosen in a symmetric manner around the midpoint of the bundle of $2n = 6$ wires. We simulated our oscillator with $n = 30$.

In practice, the wire locations that are labeled as '0' in Fig. 2.9 are actually left floating by the control logic. Assuming that the supply voltage of the inverter pair is VDD, the oscillating wires oscillate around a DC value $VDD/2$, with sinusoidal waveforms which are always in phase, but whose amplitude vary as we traverse the ring (as shown in Fig. 2.3). Since a null oscillation point exists at a distance $L/2$ from the inverter pairs, we short all $2n$ wires at this virtual ground location. As a result, all wires that are left floating by the control logic actually have a $VDD/2$ voltage on them due to the short at the virtual ground location (and therefore these wires act as ground wires in an AC sense).

From Fig. 2.9, suppose we have two configurations with numerical indices P and Q respectively. Let $P < Q$, so that there are more oscillating wires in the inner (and outer) rings for P as compared to Q. Also, the distance between oscillating wires in the two rings is lower for P. This has two effects.

- The capacitance of the oscillating wires is larger for P as compared to Q, since P has more oscillating wires.
- The inductance of P is lower than that of Q, since the current return loop is smaller in P compared to Q, due to proximity effect.

The ratio of the increase in capacitance of P over Q is less than the ratio of the increase in inductance of Q over P. As a result, based on the expression for the frequency of oscillation of the ring (Eq. 2.1), P oscillates at a higher frequency than Q.

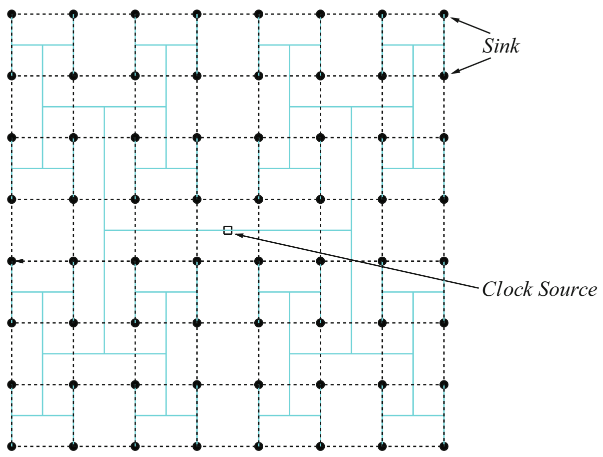


Fig. 2.11 H-Tree with overlaid mesh

The coarse frequency of our resonant standing wave oscillator is controlled by varying the values of the n-bit vector $\mathbf{x}$.

The circuitry for coarse frequency control is illustrated in Fig. 2.10. This circuit takes as an input the vector $\mathbf{x}$. Based on the values of this vector, the appropriate wires among the $2n$ wires of the oscillator are made to oscillate. If coarse configuration 2 is chosen, for example, only the top 2 and the bottom 2 wires in Fig. 2.10 oscillate. Note that this circuit resides at the mobius point of the resonant oscillator, and the cross-coupled inverter pair is shown in the figure as well. The mobius connection of the $2n$ wires is illustrated to the right of Fig. 2.10.

Fine Frequency Control: For fine frequency control, we use a set of binary weighted NMOS capacitors. In particular, we connect four NMOS capacitors at either end of the cross-coupled inverter pair for fine frequency control, as shown in Fig. 2.10. In practice, the switches in Fig. 2.10 are NMOS passgates. We tried complementary passgates, but the diffusion capacitance of complementary passgates caused a noticeable drop in oscillation frequency. In order to decrease the body effect, we connect the source and bulk terminals of these NMOS passgates.

The capacitor bank switches are controlled by the input vector $\mathbf{w}$. By applying a certain value of $\mathbf{w}$, we modulate the overall capacitance of the SWO and hence change its oscillating frequency. The capacitors are chosen such that they can cover the largest of the frequency intervals between any two coarse frequency points.

2.2.3.3 H-Tree Combined with Overlaid Mesh

We compare our approach with an H-tree with an overlaid mesh. The H-tree is a popular clock distribution approach, due to its simplicity and low power and area requirements. The structure of a 6-level H-tree clock distribution network is shown in Fig. 2.11 (with solid lines). There are $2^6 = 64$ *sinks* (end-points) in the H-tree (indicated by solid dots), which are uniformly spaced to cover the entire IC.

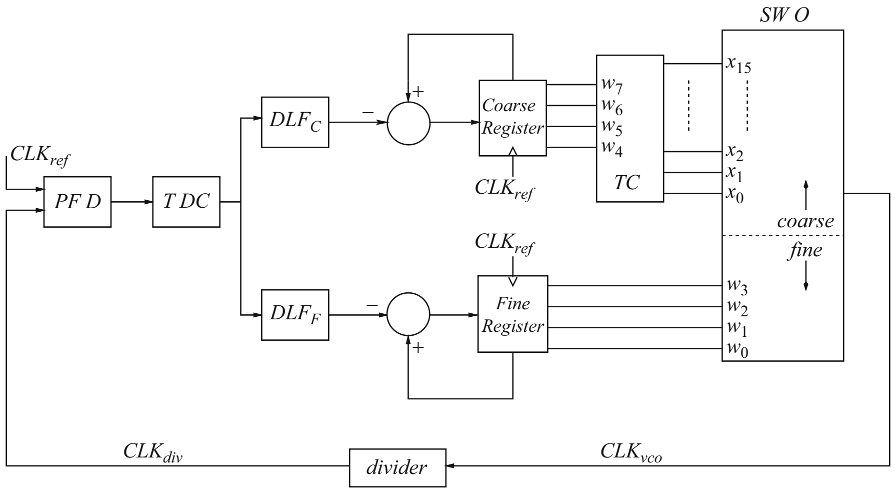


Fig. 2.12 Block diagram of the proposed PLL design

The buffers of the H-tree (omitted in Fig. 2.11) are placed symmetrically along the branches, to ensure equal delays from the clock source to all the sinks, and fast switching at the sinks. The wire segments in a buffered H-tree exhibit minimal delay variations as a consequence of PVT variations, since the delay of a wire depends solely on the RC parasitics of the wire. However, power supply variations across the die affect the delay of the buffers, resulting in an increased skew at the sinks of the buffered H-tree. Moreover, there are cycle-to-cycle variations in the *VDD* value, and hence the different buffers experience different delays on a cycle-to-cycle basis. This results in an increased cycle-to-cycle jitter. Both skew and jitter reduce the maximum operating frequency of the IC. A common industry practice to reduce the skew and the jitter is to implement a mesh on top of the H-tree (shown by dotted lines in Fig. 2.11). However, by introducing short circuit current paths between the sinks with delay differences, this topology consumes more power than the H-tree alone. Moreover, the delay variation compensation of the mesh is a strict function of the RC product of the mesh wires. This implies that in order to reduce the skew and the jitter, we require a more dense mesh, with wide wires, thereby incurring a large area and power overhead.

2.2.3.4 Proposed Digital PLL Design

Figure 2.12 shows the block diagram of our digital PLL with the SWO of Fig. 2.10 incorporated. The phase error between the reference clock (CLK_{ref}) and the divided clock (CLK_{div}) is detected by the Phase-Frequency Detector (PFD) and is quantized into a digital code by the Time to Digital Converter (TDC). The TDC output is processed by the two Digital Loop Filters (DLF) for the coarse and fine frequency

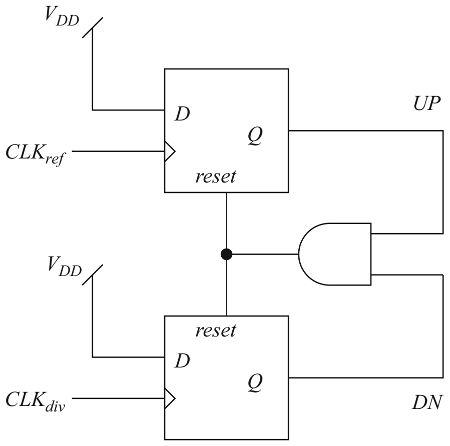


Fig. 2.13 Phase frequency detector circuit

controls. The output of the DLFs are added to coarse and fine DCO configuration words, which are registered in *Coarse Register* and *Fine Register* respectively. Note that these registers get updated every T_{ref}. A thermometer converter performs a binary to decimal encoding of the coarse control words. The output of the *TC* and the *Fine Register* are provided as input to the SWO as discussed in Sect. 2.2.3.2.

Next, we discuss the circuit details of the individual components of the PLL.

Phase Frequency Detector (PFD): The PFD consists of two flip-flops and an *AND* gate connected as shown in Fig. 2.13. The output of the PFD is dependent on both the phase and frequency difference between the input signals. The PFD has two input clocks, the external crystal clock (CLK_{ref}) and the divider output (CLK_{div}) and produces two outputs *UP* and *DN*.

The PFD is a simple state machine which has three states. Consider that the UP and DN outputs are initially low. When CLK_{ref} leads CLK_{div}, the *UP* output is asserted on the rising edge of CLK_{ref}. The *UP* signal stays high until a low to high transition of CLK_{div}. At this point of time, *DN* rises, causing both the flip-flops to reset through the asynchronous reset signal. There will be a small pulse on the *DN* output, the width of which is equal to the sum of the delay through the AND gate and the Reset-to-Q delay of the flip-flop. The pulse width of the *UP* signal is thus the phase error between the two signals. A similar situation arises when CLK_{div} leads CLK_{ref} (the phase error in this case is the width of *DN* pulse).

Under a lock condition, equally short pulses will be generated on both the *UP* and *DN* outputs. The transfer function of the PFD can be approximated as:

$$PFD(z) = \frac{T_{ref}}{2\pi} \tag{2.2}$$

Time to Digital Converter (TDC): We implement a variant of the traditional TDC with a re-circulating delay line, as shown in Fig. 2.14. It consists of m delay elements followed by a K-bit counter. Each delay element consists of 4 minimum size inverters. We define Δ_{TDC} as the delay of a single delay element, which is the TDC resolution.

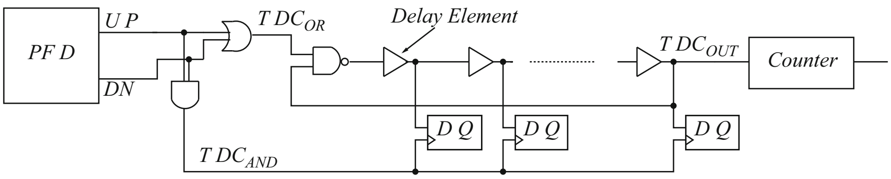


Fig. 2.14 Time to digital converter

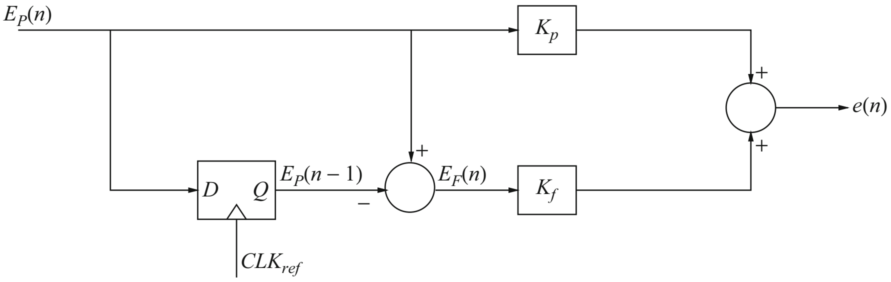


Fig. 2.15 Digital loop filter

Hence, $4 * m$ inverters and one NAND gate enabled by TDC_{OR} form an effective ring oscillator. TDC_{OR} is the logic OR, and TDC_{AND} is the logical AND of the UP and DN pulses output by the PFD. Note that, when TDC_{OR} is zero, all the outputs of the delay elements are initialized to '1'. On the rising edge of the TDC_{OR} pulse, the delay elements start toggling, causing the output TDC_{OUT} to toggle once every $m \times T_d$ seconds, where T_d is the delay of one delay element. The counter counts the number of times TDC_{OUT} toggles. On the rising edge of the TDC_{AND}, the state of the m-element delay chain and the state of the counter are latched. Together, they perform the quantization of the phase error into a digital word. In practice, we use $m = 5$, and a counter with $K = 7$ bits. The output of the TDC is $E_P[7:0]$. The 7 outputs of the counter can have 128 values. An additional 5 values (for a total of 133 values) are produced by state of the delay chain. These 133 values are encoded into $E_P[7:0]$ and driven to the DLF blocks. The transfer function of the TDC can be approximated as:

$$TDC(z) = \frac{1}{\Delta_{TDC}} \tag{2.3}$$

Digital Loop Filter (DLF) Figure 2.15 shows our implementation of the DLF. We define two error terms called the phase error (E_P) and the frequency error (E_F). E_P is the measured phase error between the T_{ref} and T_{div}. E_F is the instantaneous frequency error between the T_{ref} and the T_{div}. We observe that we can approximately obtain E_F by taking a derivative of the phase error ($E_F = \frac{dE_P}{dt}$). As shown in Fig. 2.15, we store the E_P of the last reference cycle in a register. We compute E_F for the current reference cycle as $E_F(n) = E_P(n) - E_P(n-1)$. Finally, we obtain the DLF output error

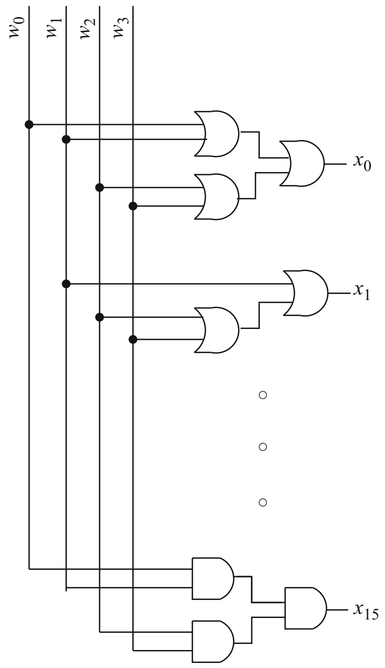


Fig. 2.16 Thermometer converter

e as $e(n) = K_P * E_P(n) + K_F * E_F(n)$, as shown in Fig. 2.15. Note that from control theory, K_P is defined as the proportional gain and K_F is defined as the differential gain. Hence, our DLF functions as a Proportional-Derivative (PD) controller. The proportional action improves the response of a system by providing an instantaneous response to the control error. Derivative action provides a fast response as opposed to the integral action, but cannot accommodate constant errors. PD control is useful for fast response controllers that do not need a steady-state phase error of 0. Though Proportional-Integral (PI) controllers are more common in PLL designs, we opt for a PD controller since it theoretically has an improved damping, reduced maximum overshoot as well as a very fast time to lock. Since our design implements a clock generation and distribution scheme, large overshoots result in a high cycle-to-cycle jitter. Moreover, our experiments suggest that the steady-state phase error obtained is very small and is at most 25 ps, which is less than 0.15 % of the reference clock cycle (T_{ref}). Note that we use two different DLFs (one for the coarse, and another for the fine frequency control). The only difference between the two is that they have a different pair of gain values for K_P and K_F. The transfer function of the DLF is given by the following equation:

$$DLF(z) = K_p + K_f * (1 - z^{-1}) \tag{2.4}$$

Thermometer Converter (TC): Figure 2.16 shows our implementation of a thermometer converter (TC) with 4 inputs and 16 outputs, to drive the coarse configuration

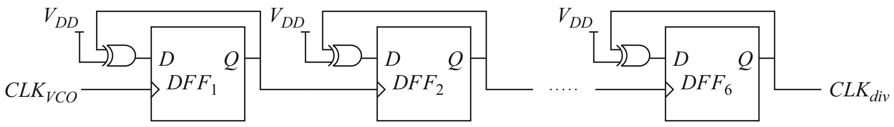


Fig. 2.17 Divider

vectors $[x_{15}\ldots x_0]$. Lets say the decimal value of the input (vector w) is m (where $0 \le m \le 15$). Then exactly m outputs (from x_0 to x_{m-1} are '1', and the rest of the $(16-m)$ outputs (from x_m to x_{15}) are '0'.

Divider: In our experiments, we nominally assume an external 50 MHz reference crystal clock. Hence we require a division factor of 64 in order to achieve an oscillator operating at a center frequency of of 3.2 *GHz*. We implement the divider as a 6-bit ripple counter, as shown in Fig. 2.17. The clock to the first stage of the ripple counter is the recovered clock from the oscillator (CLK_{VCO}). The output is the divided clock (CLK_{div}) which serves an input to the phase frequency detector.

2.2.4 Experiments

We implemented our design in the 32 nm (PTM 2013) technology, with $VDD = 0.9\ V$. All simulations were conducted in HSPICE (Inc Meta-Software). RLC parasitics for all the wires were extracted using (Raphael Interconnect Analysis Tool: User's Guide). We assume a square die of size 1 $cm \times 1$ cm for our experiments. We compare the jitter, skew, area and power of our approach with that of a buffered H-tree with an overlaid mesh (see Sect. 2.2.3.3).

2.2.4.1 Coarse Frequency Control

The inner and outer wiring rings of the oscillator consist of $n = 30$ wires each. Each of the 60 wires were 1 μm wide, with an inter-wire spacing of 0.5 μm. We implement the SWO in form of a *comb* topology as shown in Fig. 2.8. For a square die of size 1 $cm \times 1$ cm, the required length of the comb is 80 mm. We implemented a $17 \times L$ SWO, where L (the distance between two consecutive cross-coupled inverter pairs) was taken to be 5000 μm. Note that as discussed in Sect. 2.2.3.1, the frequency of the SWO is determined by the length L, and the length of the comb should be an odd multiple of L.

The 16 coarse configurations that we used are shown in Table 2.1. Note that for each configuration, we report the 30 values of $\mathbf{x}$. Note that the x_0 value is the leftmost bit of any row of Table 2.1. Also, the 2 highest order bits are always zero, and their corresponding NMOS devices are omitted from the circuit (see Fig. 2.10). Similarly the 2 lower order bits of $\mathbf{x}$ are always 1, and their corresponding NMOS devices are also omitted from the circuit of Fig. 2.10. These 2 outermost wires are statically

Table 2.1 Coarse configurations used in our experiments

Configuration	**x** value
0	111111111111111111111111111100
1	111111111111111111111111000000
2	111111111111111111111100000000
3	111111111111111111111000000000
4	111111111111111111100000000000
5	111111111111111110000000000000
6	111111111111111000000000000000
7	111111111111100000000000000000
8	111111111110000000000000000000
9	111111111000000000000000000000
10	111111100000000000000000000000
11	111111000000000000000000000000
12	111110000000000000000000000000
13	111100000000000000000000000000
14	111000000000000000000000000000
15	110000000000000000000000000000

connected to the cross-coupled inverter pair. With 2 outer wires always oscillating and 2 inner wires always floating, we can choose to oscillate any number between zero through 26 of the remaining wires, giving us a total of 27 coarse configurations. However, we use only 16 of these 27 values as shown in Table 2.1. For example, while going from configuration 15 to 14, we turn on one more wire. However, while going from configuration 10 to 9, we turn on 2 extra wires. We skip some intermediate configurations because the frequency difference between some configurations and their neighbors was very small. We also ensured that we can cover the frequency difference between any two coarse configurations with the help of the fine frequency control circuitry.

We next size the cross-coupled inverters such that they can provide the negative resistance for any configuration of the SWO. Note that the cross-coupled inverters are the only active elements in the SWO. Hence, the capacitance of the cross-coupled inverter (and hence the capacitance of the SWO) depends on *VDD*, which can therefore result in cycle-to-cycle variations in the clock period (jitter). We found that the minimum width of the cross-coupled inverters is 40μm (20 μm) for the PMOS (NMOS) device, which allowed all configurations to sustain oscillation even at a 10 % lowered *VDD*.

To size the NMOS passgates that drive the x_i signals in Fig. 2.10, we conducted a SPICE sweep starting with minimum sized passgates. As we increased the passgate size, we found that more configurations were able to sustain oscillation. For an NMOS passgate of size 2.5μm, oscillations were sustained by every configuration of Table 2.1.

We verified that our oscillator provides a continuous frequency response from $\sim$3.1 GHz to $\sim$3.6 GHz, with a center frequency of 3.3 GHz. Figure 2.18 illustrates the set of frequencies achievable by our coarse tuning approach. The fine-frequency control value for these simulations is fixed at half of its maximum value, allowing

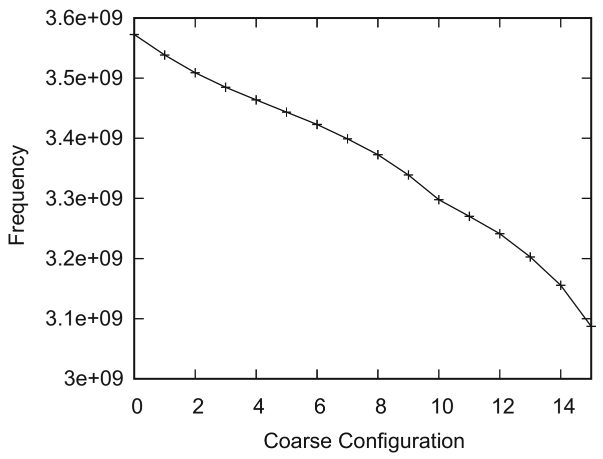


Fig. 2.18 Frequencies achieved through coarse tuning

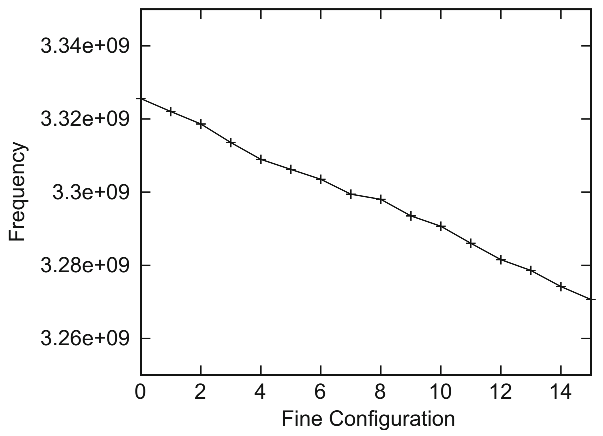


Fig. 2.19 Frequency range for fine tuning of coarse configuration 9

us to minimally extend the range reported in Fig. 2.18 in both directions. We thus achieve around 15 % tuning range around a center frequency of 3.3 *GHz*.

2.2.4.2 Fine Frequency Control

We ensured that our fine frequency circuitry can cover the largest frequency difference between any two adjacent coarse configurations shown in Fig. 2.18. We use 4 binary weighted NMOS capacitors for the fine frequency control. The width of the smallest NMOS capacitor was chosen to be $0.2u$. The NMOS pass gates which are used to turn on these capacitors are also binary weighted. The width of the smallest NMOS pass gate is $0.4u$. Figure 2.19 illustrates the continuous fine frequency range achievable for one of the coarse configurations (Configuration 9).

2.2.4.3 Phase Locking Results

From Eqs. 2.2, 2.3 and 2.4, the open loop transfer function of the of the PLL is given by the following equation:

$$H(z) = \frac{T_{ref}}{2\pi} * \frac{1}{\Delta_{TDC}} * [K_p + K_f * (1 - z^{-1})] * K_{DCO} \tag{2.5}$$

where K_{DCO} is the DCO gain. The DLF coefficients depend on two factors K_{DCO} and Δ_{TDC}. K_{DCO} varies from 34.2 MHz/unit to 68.3 MHz/unit for the coarse frequency, and is ~2.1 MHz/unit for the fine frequency control. We implement the smallest delay element in our TDC with a chain of 4 minimum size inverters. Hence the delay resolution of TDC (Δ_{TDC}) is the sum of delay of these 4 inverters (~25 ps). We choose the DLF coefficients in order to (a) ensure the stability of the control loop and (b) minimize the maximum overshoot in frequency during PLL locking. For fine frequency control, we choose the phase gain (K_p) as $\frac{1}{2}$ and the frequency gain (K_f) as 1. For the coarse frequency control, we choose the phase gain (K_p) as $\frac{1}{32}$ and frequency gain (K_f) as $\frac{1}{16}$. Note that the gain of the DCO is at most ~16× larger for the coarse frequency control compared to the fine frequency control. Hence, we adjust our DLF gains by the same amount. We verified the stability of the both coarse and fine frequency control by doing extensive MATLAB simulations around the values listed above.

Figure 2.20 shows the DCO frequency (top plot) and the measured phase error between the divided clock and the reference clock (bottom plot). For all our simulations, we observe that the PLL locks within 15-25 CLK_{ref} cycles. With a reference crystal frequency of 50 MHz, the above locking time is only $\sim 0.5us$. Note that we make changes in both fine and coarse frequency every T_{ref} cycles. Hence, we observe that there are flat lines in the DCO plot in Fig. 2.20, during which the configurations (and hence the DCO frequency) remain constant. We have verified that our PLL was able to correctly lock to any reference frequency (in the locking range) by testing it with different reference frequencies and different initial phase errors. Figures 2.21 and 2.22 show the simulation results of two such experiments. Observe that in both Figs. 2.20 and 2.21 we lock to the same target frequency of 3.33 GHz. However, the initial phase errors (as shown by the left portion of the two bottom plots) are different. In Fig. 2.22, the PLL locks to a different target frequency of 3.38 GHz.

To recover a rail-to-rail clock from any location on the SWO, a clock recovery circuit is implemented as shown in Fig. 2.23. This circuit is essentially a differential amplifier with a buffered output. A plot of several recovered signals from all the 17 inverter points around the SWO is shown in Fig. 2.24. From this figure, the rising slew is 5.83 ps, while the falling slew is 6.23 ps (for a clock of period of 300 ps). Note that the clock edges from all the inverter points coincide, suggesting almost zero skew (under no PVT variations).

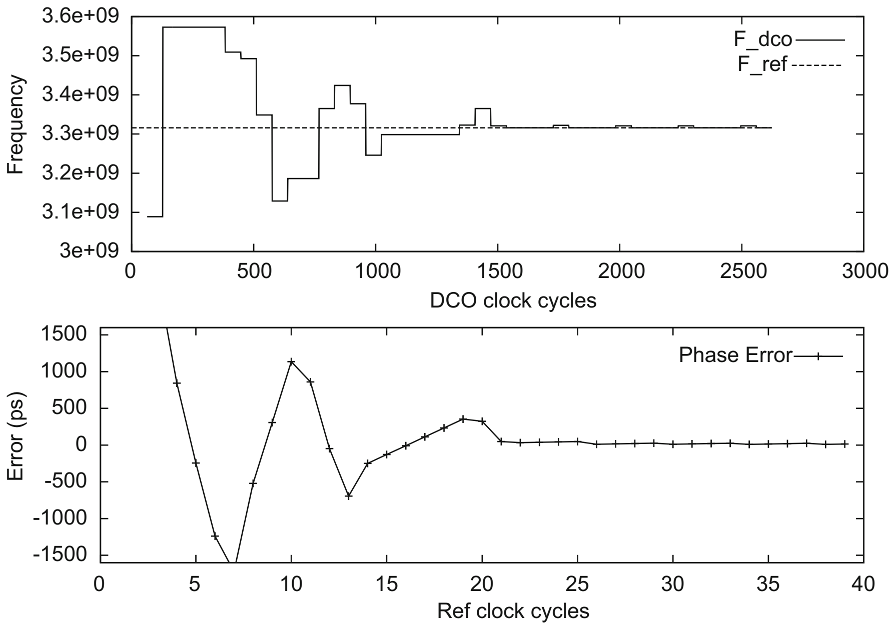


Fig. 2.20 PLL locking for a target frequency of 3.33 GHz

Table 2.2 Comparison between H-tree and SWO-based clock distribution

Scheme	Wiring Area	Circuit Area	Power	Jitter	Skew
H-tree	2.48 mm^2	137.85 μ^2	198.1 mW	30.5 ps	192.8 ps
SWO	8.67 mm^2	141.54 μ^2	81.31 mW	6.65 ps	34.7 ps

2.2.4.4 H-tree with Overlaid Mesh—Comparison

We construct a 6-level H-tree with an overlaid mesh as shown in Fig. 2.11. The width of the clock wire for the H-tree was taken to be $0.45u$. The mesh was implemented with wires of thickness $10u$. Our clock buffer consists of a $85\times$ inverter followed by a $256\times$ inverter. We insert two buffers at every bifurcation of the H-tree, one for each branch. We also make sure that the longest wire that we drive is less than 1.25 mm, to ensure that the slew is less than 50 ps at the buffer inputs.

Table 2.2 compares our SWO clock distribution with the H-tree combined with an overlaid mesh. We assume 60 mV of *VDD* variation from cycle-to-cycle in our experiments to calculate cycle-to-cycle jitter (variation in clock period between consecutive clock cycles). The *VDD* variation of 60 mV was obtained based on similar data measured from contemporary ICs (Juniper Networks 2010). To calculate the skew (variation of clock edges across different clock *sinks* on the die), we divide the die into 4 quadrants. For the North-East quadrant, we assume all the transistors to be in slow corner (10 % lower *VDD*, 10 % higher V_t, 80°C operating tempurature). The North-West and South-East quadrants have nominal *VDD* and V_t, and operate at

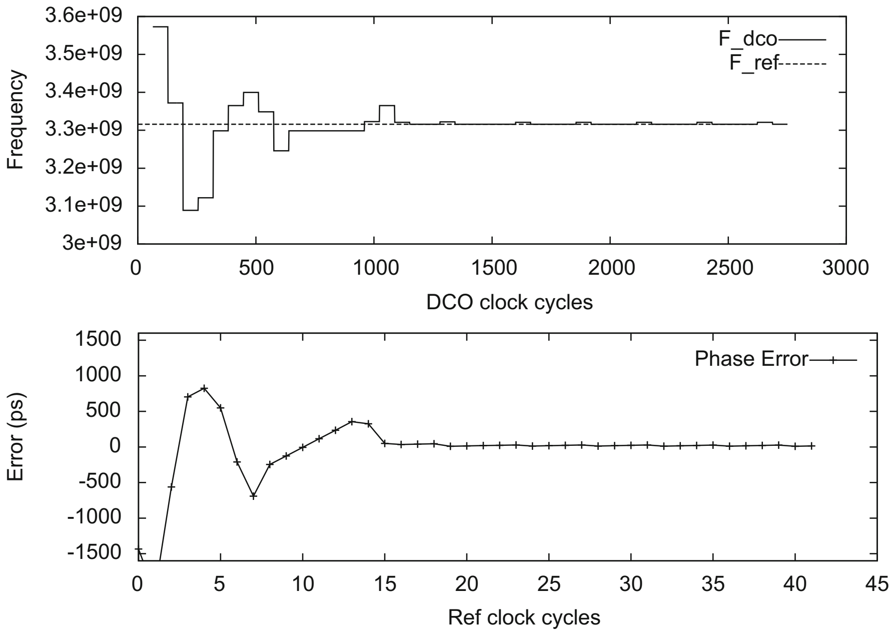


Fig. 2.21 PLL locking with a different initial phase error

room tempurature. For the South-West quadrant, we assume all the transistors to be in fast corner (10 % higher *VDD*, 10 % lower V_t, 0°C operating tempurature). Our experimental conditions are pessimistic in order to compare the performance under worst case variations.

From Table 2.2, we note that our scheme has a much lowered power requirement (by $\sim 2.4\times$) over the H-tree with overlaid mesh. In practice the power improvement would be larger, since we do not model the PLL power for the H-tree with overlaid mesh (while this portion of the power is accounted for in our method). This reduced power is attributed to the resonant nature of the SWO. The jitter of our approach is a mere 6.65 ps, which is $4.6\times$ lower than that of a buffered H-tree with an overlaid mesh. Again, this attributed to the relatively high Q-factor (which is approximately 5) for our SWO. Also, the skew of our approach is about $5.5\times$ better than the buffered H-tree with an overlaid mesh, for the same reason that results in improved jitter. The circuit area of the two approaches are similar, while the wiring area of our approach is $\sim 3.5\times$ higher. Clearly for high-performance processors, where clock skew, jitter and power are significant concerns, our SWO provides a significantly improved clock generation and distribution scheme.

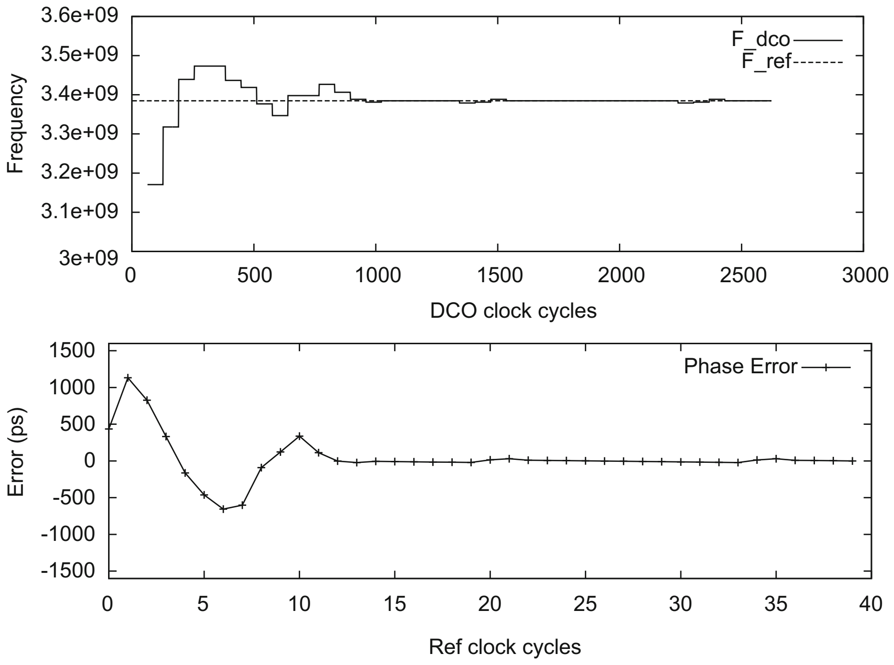


Fig. 2.22 PLL Locking for a Target Frequency of 3.38 GHz

2.2.5 Conclusion

In this work, we have presented a resonant SWO based phase-locked clock generation and distribution scheme with superior jitter, skew and power characteristics. Phase locking is accomplished by a fully digital control loop consisting of a coarse frequency control and a fine frequency control. The SWO frequency is modulated in a coarse manner by changing the ring inductance, by modifying the number of oscillating wires. Fine frequency control is achieved by modifying ring capacitance via a binary-weighted capacitor bank. The digital control loop has been analyzed and implemented in HSPICE. Clock distribution is performed by routing the resonant ring on the die in a “comb” manner. Our approach is compared to an H-tree with an overlaid mesh. Given the relatively high Q-factor of the SWO, this clocking approach exhibits a very low cycle-to-cycle jitter (about 4.6× better) and very low skew (about 5.5× better) compared to the H-tree with overlaid mesh. The power consumption of our scheme is upwards of 2.4× improved as well, given the resonant nature of the approach.

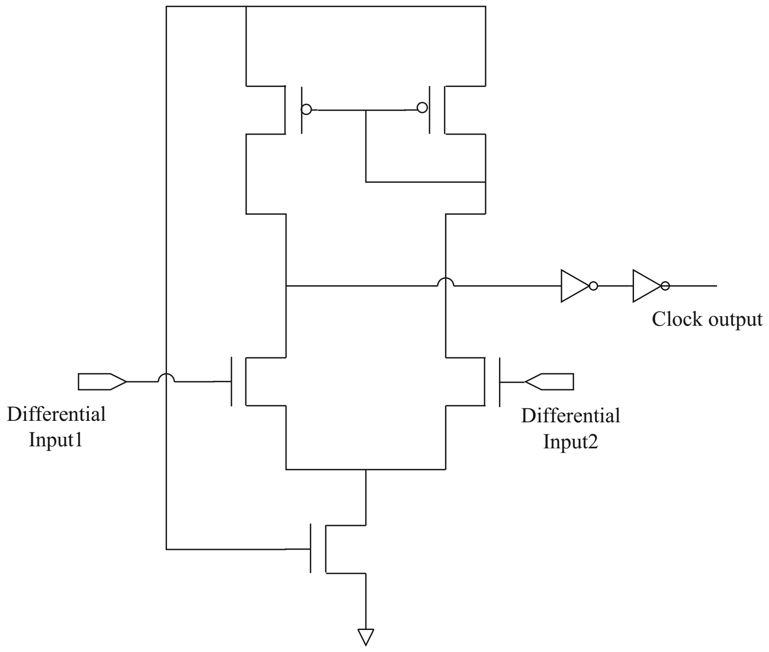


Fig. 2.23 Clock recovery circuit

2.3 Automated Methodology to Generate Low Jitter Buffered H-tree

In the previous section, we presented an SWO-based clock generation and distribution methodology and compared it with a traditional buffered H-tree, with an overlaid mesh. However, in recent fabrication technologies, the buffered H-tree clock distribution network has been quite commonly used. In the following section, we present a dynamic programming based approach to synthesize a minimum cost buffered H-tree clock distribution network. Our primary cost function is the end-to-end jitter of the synthesized H-tree, while the secondary goal is to minimize power as well.

2.3.1 Introduction

In recent VLSI fabrication processes, with decreasing feature sizes, wire widths have been shrinking. The direct consequence of this is an increase in wire resistance (R_w), since $R_w = \frac{\rho \cdot L}{W \cdot T}$, where ρ is the resistivity of the wire material, and L, W and T are the length, width and thickness of the wire respectively. Since wiring delays on a die are proportional to the RC time constant of the wires, this increase in wire resistance results in increased wiring delays, especially for long wires on a chip

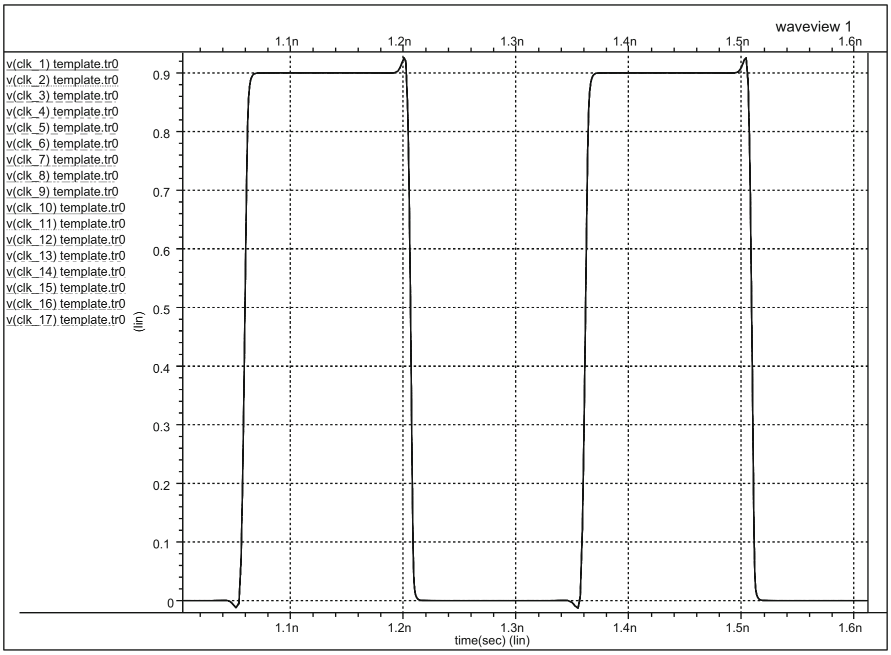


Fig. 2.24 Overlay of recovered waveforms

(which do not scale with technology). The classical approach to solve this problem is to partially compensate for the reduction of W by increasing T, but this results in wires which are thin and tall, leading to an increased cross-coupling capacitance among wires, thereby again resulting in an increased RC time constant. The design of the wiring stack (i.e. the height and minimum width of wires on each metal layer of the IC) is therefore a complex problem. Despite significant research in academia as well as industry, it remains true that wiring delays in VLSI are on the rise (The International Technology Roadmap for Semiconductors 2007), and this problem is particularly acute for global signals which need to be driven at high frequencies with tight slew-rate constraints.

The global clock signals on an IC are particularly impacted by this problem. The clock signal is operated at high frequencies, and in addition to having stringent slew-rate constraints, the clock signal also has very tight jitter constraints. Consider the rising (falling) edge of the clock. Over a number of cycles, this edge may appear early or late. Jitter is difference between the latest arrival time and the earliest arrival time of a clock signal edge. The clock period of the IC is determined by adding the worst-case clock jitter value to the longest delay between any two sequentially adjacent flip-flops in the circuit. The longest delay between two sequentially adjacent flip-flops is computed as the sum of the longest combinational delay of the circuit plus the setup time and the clock-to-output delay of the flip-flops of the design. As

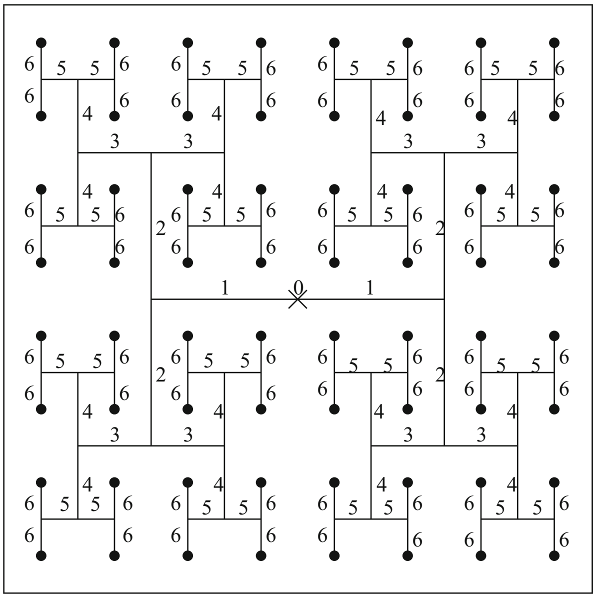


Fig. 2.25 Buffered H-tree Clock Distribution Network with 6 Levels

a result, a high clock jitter results in a reduced frequency of operation of the IC, and hence it is important to keep jitter to a minimum.

Given the timing critical nature of the clock signal, it is very important to distribute this signal carefully on an IC, to ensure high slew-rates, high frequency and low jitter. Over the years, several clock distribution methodologies have been developed, such as the H-Tree, mesh and star, among others (Friedman 2001; Anceau 1982). The H-tree is a popular clock distribution approach, due to its simplicity and low power and area requirements. In this work, we therefore focus our attention on an H-tree based clock distribution network.

The structure of a 6-level H-tree clock distribution network is shown in Fig. 2.25. The numbers on each of the branches of the H-tree indicate the level of that branch. The $\times$ notation in the center indicates the source of the H-tree. There are $2^6 = 64$ sinks (end-points) in the H-tree, indicated by dots. Note that the end-points of the H-tree distribution network are uniformly spaced, and cover the entire IC area. The lengths of the i^{th} branch of the H-tree is given by $L_i = \frac{S}{2^{\lceil \frac{i}{2} \rceil + 1}}$, where S is the dimension of any side of the (square) die. For example, for a chip which is 20 mm wide, the lengths of the branches of a 6-level H-tree are 5 mm, 5 mm, 2.5 mm, 2.5 mm, 1.25 mm and 1.25 mm, for levels 1 through 6 respectively.

The unbuffered H-tree network is a zero-skew balanced network (if process and temperature variations are not considered). A traditional H-tree is designed assuming that the clock driver at the center of the H-tree is large enough to drive the entire clock tree. Wire widths and driver sizes are fixed to make sure that the clock signal can drive the local clock regenerators at the leaves (sinks) of the H-tree for the required frequency of operation, with a sufficiently high slew-rate. The optimal (with respect to having a high slew-rate clock signal and also in terms of reducing the clock distribution area and delay) wire sizing methodology dictates that we utilize wider wires near the center of the H-tree, and narrower wires as we get closer to the leaves. This is referred to as a *tapered* wire sizing approach.

Note that with increasing wiring delays in recent technologies, it is no longer feasible to utilize unbuffered H-trees for ICs of reasonable size (greater than ~ 5 mm). For this reason, buffered H-trees are commonly utilized in today's IC designs. The buffers of the H-tree are not shown in Fig. 2.25, and can be at arbitrary locations along any path from the source to a sink, provided the locations and sizes of each buffers is the same for all 64 paths of the H-tree. Typically, when an H-tree is manually designed (which is the common practice), designers typically place identical buffers are regular intervals along the H-tree, so as to simplify the design process. Also, in a manually designed H-tree the wire topologies are typically fixed from source to all sinks. For example, a common practice is to place buffers at each branching site in the H-tree. Note that with a buffered H-tree, tapered wire sizing (commonly used in unbuffered H-trees) is no longer necessary, and the wire widths of each level of the H-tree are typically dependent on the size of the driving buffer.

The buffered H-tree alleviates the wiring delay problem in recent technologies, and enables a designer to deliver a high frequency, high slew-rate clock at all the sinks, with low end-to-end delay. However, the disadvantage of such a buffered H-tree is an increase in the jitter at the sinks. In practice, there are cycle-to-cycle variations in the *VDD* value across the die. On one hand, the wire segments in a buffered H-tree do not exhibit delay variations as a consequence of the cycle-to-cycle *VDD* variations along the die, since the delay of a wire depends solely on the RC parasitics of the wire. However, the cycle-to-cycle variations in the *VDD* value across the die affect the delay of the buffers in a buffered H-tree, since the delay of an inverter depends on its supply voltage. Hence, in practice, the different buffers in a buffered H-tree experience different delays on a cycle-to-cycle basis. This results in an increased jitter at the sinks of the buffered H-tree. This can reduce the maximum operating frequency of the IC, since the operating frequency has to be guard-banded to account for worst-case jitter, as discussed earlier.

Typically a buffered H-tree network is designed to minimize the end-to-end delay of the tree. However, most modern ICs have an on-chip Phase Locked Loop (PLL) to synchronize the off-chip clock (typically generated by a crystal oscillator) to the on-chip clock. As a consequence, the end-to-end delay of the H-tree (or any other clock distribution network) is not of consequence. Rather, our position is that the H-tree should be designed for minimum jitter. This is the key guiding principle of this work. We show that designing a buffered H-tree for minimum end-to-end delay can yield different results compared to designing the buffered H-tree for minimum jitter.

Our approach automates the design of the buffered H-tree, thereby availing significant optimization flexibility that is not possible to leverage in a manual H-tree design process. In particular, we select the best H-tree by exploring various (a) wiring configurations (referred to as *wire-codes* in the sequel), (b) buffer sizes (c) segment lengths and (d) segment topologies. Any given wiring segment in our approach has a total of 2745 choices for its implementation.

The automatic H-tree synthesis procedure utilized in this work is based on dynamic programming (DP). First, we statically characterize a large number of ways of implementing a segment, by computing their power, segment delay and segment jitter and output slew, as a function of input slew. This is done for all 2745 segment choices up-front. Then a dynamic programming engine selects the lowest cost H-tree implementation that satisfies constraints such as output slew limits and cycle-to-cycle jitter. Our approach is implemented to minimize one of two cost functions—a weighted sum of power and jitter, and a weighted sum of power and delay. The resulting H-trees are simulated in HSPICE, and the end-to-end delay, power and end-to-end jitter estimated by our DP engine matches the HSPICE results with a maximum error of 4.6 %. We also compare our DP based buffered H-tree synthesis with a manually designed H-tree. Results demonstrate that our approach, when run with jitter minimization as its goal, produces H-trees with strictly better jitter (by as much as 28 %) and power (by as much as 46 %) compared to a manually designed H-tree. Also, it achieves better jitter performance than the DP based approach run with delay minimization as its goal.

The key contributions of this work are:

- We argue that buffered H-trees should be designed to minimize jitter (as opposed to minimizing end-to-end delay which is the typical approach). We demonstrate that synthesizing an H-tree for minimal end-to-end delay does not necessarily minimize jitter, underscoring the importance of the above observation.
- We present a DP based automated approach to synthesize a buffered H-tree. This approach simultaneously explores a large number of wiring wire-codes, buffer sizes, segment lengths and segment topologies.
- With the cost function of minimum jitter, our approach produces an H-tree which has up to 28 % lower jitter than a manually designed H-tree. The power as well as the end-to-end delay of our approach is better by up to 46 % and 16.6 % respectively.
- Our DP engine can produce H-trees with a cost functions of a weighted sum of power and jitter, and a weighted sum of power and delay.

The rest of this section is organized as follows: Sect. 2.3.2 describes previous approaches in this area. Section 2.3.3 describes our approach, while Sect. 2.3.4 describes the results of experiments which we performed to validate our approach. In Sect. 2.3.5, we draw conclusions.

2.3.2 Previous Work

Several clock distribution methods (such as the H-trees, meshes, stars etc.) have been studied in the past. Some excellent survey style papers in this area are (Friedman 2001; Anceau 1982). Most of previous works on clock network design

(Chen and Wong 1996; Liu et al. 2000; Tsai et al. Tsai et al. 2004) attempt to obtain zero skew, while others try to bound the clock skew within a given hard constraint (Cong et al. 1998). Some clock synthesis works generate unbuffered clock networks that require a separate buffer insertion stage using conventional or specialized buffer insertion algorithms. In other works (Rajaram and Pan 2006; Chaturvedi and Hu 2004), buffer insertion and clock tree routing are integrated. The Deferred-Merge Embedding (DME) algorithm (Chao 1992) is the fundamental technique used in many recent clock tree synthesis approaches. It consists of a bottom-up stage and a top-down stage. A tree of merge segments, which represent the possible locations of merge nodes, is constructed in the bottom-up stage. Once all the merge segments are constructed, the exact locations of merge nodes are determined in the top-down stage. Merge segments are calculated to balance the delays of the two sub-trees. A key difference between DME and our approach is that DME is used for clock tree synthesis for ASICs, unlike the H-tree approach of our work which is more pertinent for custom IC designs. In this work we intend to develop an optimal H-Tree that will minimize the appropriate cost function by exploring a large number of wiring wire-codes, buffer sizes, and segment length and segment topologies. This automatic tool gives the flexibility to the designer to optimize delay, power and/or jitter across the clock tree. None of the previous approaches attempts to minimize jitter as a design goal.

2.3.3 *Our Approach*

Traditionally, buffered H-trees have been designed manually, with the goal of minimizing end-to-end delay. Since the end-to-end delay is unimportant in an IC which has a PLL, and since the buffers in the H-tree introduce jitter at the clock sinks, the goal of this work is to automatically synthesize a buffered H-tree, which minimizes the end-to-end jitter of the synthesized H-tree. The secondary goal is to minimize power as well.

In the remainder of this section, we will discuss the design scenario for this effort, along with the electrical constraints which we impose on the resulting synthesized buffered H-tree. We end with a discussion of our dynamic programming (DP) based automated buffered H-tree synthesis approach.

2.3.3.1 Buffered H-tree Construction

We conduct our experimentation on a 6 level buffered H-tree, implemented in a 45 nm (PTM 2013) fabrication process. The H-tree is implemented on METAL9, and we assume that 10 metal layers are available. A variety of buffers are used. Each buffer consists of two inverters, with the driving inverters of size $32\times$ to $51264\times$ of a minimum sized inverter, in increments of $32\times$. The other inverter in the buffer is sized $3\times$ smaller than the driving inverter.

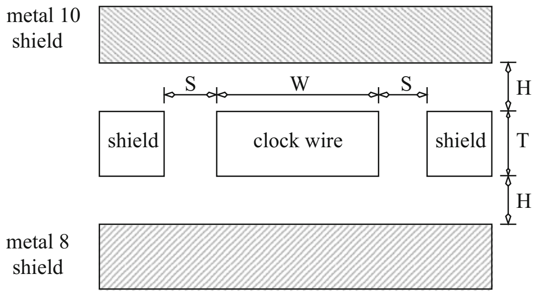


Fig. 2.26 Cross section of wires in any H-tree segment

Table 2.3 Wire-codes and their characteristics

Wire-code	$W(\mu)$	$S(\mu)$	Res. (Ω/mm)	Cap. (F/mm)
WC1	0.45	0.45	69.78	1.68E-13
WC2	0.45	0.9	69.78	1.08E-13
WC3	0.9	0.45	34.89	1.84E-13
WC4	0.9	0.9	34.89	1.23E-13
WC5	1.35	0.45	23.27	1.98E-13
WC6	1.35	0.9	23.27	1.38E-13

A total of 6 wiring configurations (referred to as wire-codes) are available for any buffered segment of the H-tree. There is no restriction on the number of wire-codes utilized for the synthesized H-tree. These wire-codes are referred to as WC1 through WC6, and their details are described next. Each wire-code consists of a METAL9 wire, shielded on either side as well as above and below by grounded wires, as shown in Fig. 2.26.

Table 2.3 describes the details of the 6 wire-codes, along with their parasitic resistance and capacitance values, which are extracted using Raphael (Raphael Interconnect Analysis Tool: User's Guide). All resistive values are reported for an operating temperature of 100°C, to model the worst case wiring delay condition. For all wire-codes, we assumed $T = 0.9$ μm, $H = 1.4$ μm and a dielectric constant $\kappa = 2.5$. The shield wire has a width of 0.45 μm in all wire-codes.

A variety of segments are selectable during the automatic synthesis of the buffered H-tree using our approach. These segments fall into three categories, as shown in Fig. 2.27. These segments are (a) a straight segment (b) an I segment and (c) a T segment. The total length of these segments is referred to as x. The straight portion of a T segment has a length y. For all segments, $x \in [1\ mm, 2.5\ mm]$, with increments of 0.25 mm. For a T segment, $y \in [0.25, x - 0.5\ mm]$, with increments of 0.25 mm.

As discussed earlier, jitter in a buffered H-tree occurs due to cycle-to-cycle local variations of *VDD* across the die, which result in relative changes in the clock buffer delays. We next discuss our methodology for computing jitter. For our 45 nm process, the supply voltage is 1.1 V. The maximum variation in the *VDD* value is assumed to be 10 % of *VDD*, or 110 mV. This variation is due to various sources, including variations in the voltage regulator output, IR drops in the power distribution network, etc. Of this 110 mV, about 70 % is attributed to cycle-to-cycle variations, and 30 % constitutes long term variations. Therefore the cycle-to-cycle variation of the supply

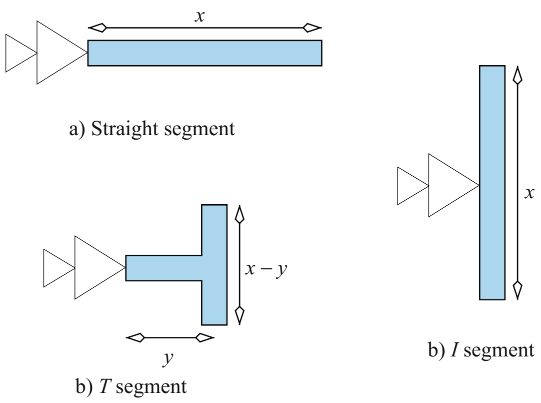


Fig. 2.27 Types of wire segments used For Buffered H-Tree construction

voltage is 77 mV. For purposes of jitter measurement, we therefore consider jitter to be the difference in delay of the clock buffer for *VDD* values of 990 mV and 1067 mV (i.e. 990 mV + 77 mV). These assumptions are supported by experimental data obtained from an industrial IC design (Juniper Networks 2010).

The jitter measurement methodology is illustrated in Fig. 2.28. Consider the clock buffer operating at 1067 mV. Assume that the solid thick line is the input to the buffer, and the dashed thick line is its output. Therefore its propagation delay at 1067 mV is T_{pd}^{1067} as marked. Now consider the clock buffer operating at 990 mV. Assume that the solid thin line is the input to the buffer, and the dashed thin line is its output. Therefore its propagation delay at 990 mV is T_{pd}^{990} as marked. The jitter of the clock buffer is therefore $J = T_{pd}^{990}$ - T_{pd}^{1067}.

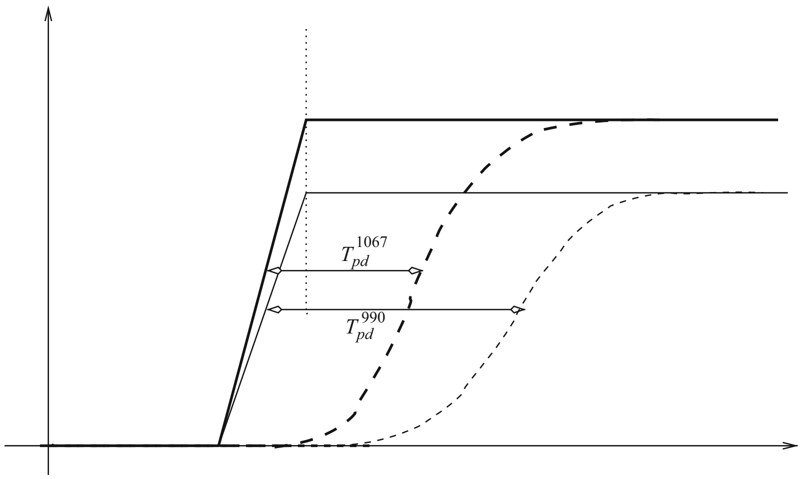


Fig. 2.28 Measuring Jitter

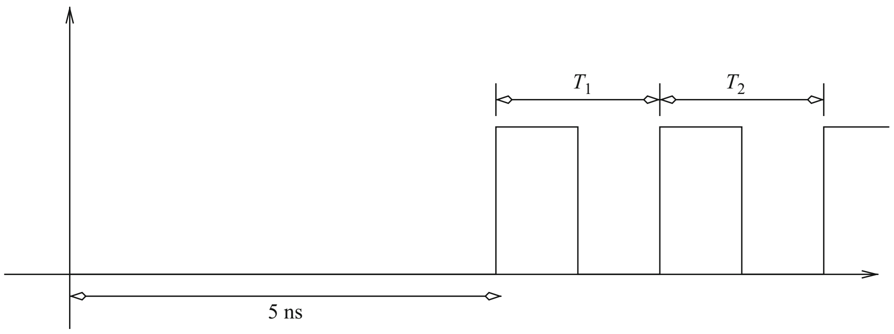


Fig. 2.29 Wake-up Jitter experiment

Note that the input to the clock buffer operating at 1067 mV and 990 mV in our experiments could be driven in one of two ways—with the same slew-rate (Method A) or with the same transition time from rail to rail (Method B). Figure 2.28 shows the input of the clock buffer being driven using Method B. We use Method B instead of Method A based on experiments in which we drove a long chain of identical H-tree segments with inputs of both kinds. We noticed that the output after the signals traverse 5 H-tree segments closely match Method B, which is why we utilize this method to drive the clock buffer inputs when measuring jitter.

We additionally invoke several constraints on the wire segments utilized by our buffered H-tree synthesis algorithm

- First, if any segment being considered by the synthesis algorithm has a output slew greater than 150 ps, we reject that segment. The output slew is computed as the 10 % to 90 % (or 90 % to 10 %) transition time for the output of the segment, divided by 0.8. This constraint ensures that the clock signal at the sinks has a high slew-rate which is essential for a clock signal, and also results in reduced crow-bar currents and hence lower power.
- We also ensure the cycle-to-cycle ("wake-up") jitter is less than 1 ps. Figure 2.29 illustrates this idea. Modern ICs frequently need to be designed in a manner in which the IC needs to be put in "sleep", in order to save power. In such a case, the clock signal is held static during the "sleep" mode. After the IC "wakes" up, the pulse widths of the first two clock pulses must be tightly controlled for correct functionality. The wake-up jitter is defined as $|T_2 - T_1|$ (see Fig. 2.29). We constrain each segment to have a wake-up jitter of less than 1 ps. All segments which fail this requirement are rejected. This requirement essentially requires that each segment be able to drive its clock signal to rail values during normal operation of the IC. If this is not true for any segment, such a segment will fail the wake-up jitter test.

2.3.3.2 Buffered H-tree Construction

Our DP based algorithm for automated synthesis of the buffered H-tree is described next. We begin by first characterizing (using HSPICE (Inc Meta-Software)) each possible segment configuration, which consists of the different wire-codes, buffer sizes, segment lengths and segment types described earlier in this section. There are a total of 2745 unique segment configurations available to the DP algorithm. For each such segment configuration, we drive it with varying input slews (ranging from 20 ps to 300 ps, with 1 ps increments). We record the jitter, output slew, delay and the power of the segment configuration as a function of its input slew. This characterization data is now used by the DP algorithm. We implement two cost functions for the DP algorithm. Suppose that the cost is being computed for a segment configuration i which builds upon a partial solution with total jitter J, total delay D and total power P. The segment configuration i has jitter J_i, delay D_i and power P_i.

- A weighted sum of power and jitter. This cost function is expressed as
  $C_{PJ} = \alpha \frac{(P+P_i)}{P*} + (1-\alpha)\frac{(J+J_i)}{J*}$
  Note that $P + P_i$ and $J + J_i$ are the total power and total jitter after segment i is appended to the partial solution. Also note that P^* and J^* are the average power and jitter values respectively, for all candidate segment configurations that may be used for segment i. Since the scale of power and jitter are different, these values are introduced in order to equalize the contribution of power and jitter to the cost function.
- A weighted sum of power and delay (or slew). This cost function is expressed as
  $C_{PD} = \alpha \frac{(P+P_i)}{P*} + (1-\alpha)\frac{(D+D_i)}{D*}$
  Note that $P + P_i$ and $D + D_i$ are the total power and total delay after segment i is appended to the partial solution. Also note that P^* and D^* are the average power and delay values respectively, for all candidate segment configurations. Since the scale of power and delay are different, these values are introduced in order to equalize the contribution of power and delay to the cost function.

Suppose we have a optimal solution to the buffered H-tree construction problem S. Clearly, this solution consists of making a choice of the last segment s (closest to the sink) to be selected. Hence the solution to the buffered H-tree construction problem from the source of the H-tree to the input of segment s must also be optimal (otherwise we contradict the optimality of S). Since the cost functions of jitter, delay and power are additive, an optimal solution to the buffered H-tree construction at any location n can be constructed from the optimal solution of the buffered H-tree construction problem for every location m which is downstream from n (i.e. m is closer to the source of the H-tree than n). Hence the criterion for optimality of DP is satisfied and as a result, DP can yield an optimal solution (for the cost functions of delay, jitter or power) to the buffered H-tree construction problem.

The DP based algorithm selects the lowest cost buffered H-tree implementation that satisfies constraints such as output slew limits and cycle-to-cycle wake-up jitter. The resulting H-trees are simulated in HSPICE, and the end-to-end delay, power and end-to-end jitter estimated by our DP engine are compared with the corresponding values obtained via HSPICE.

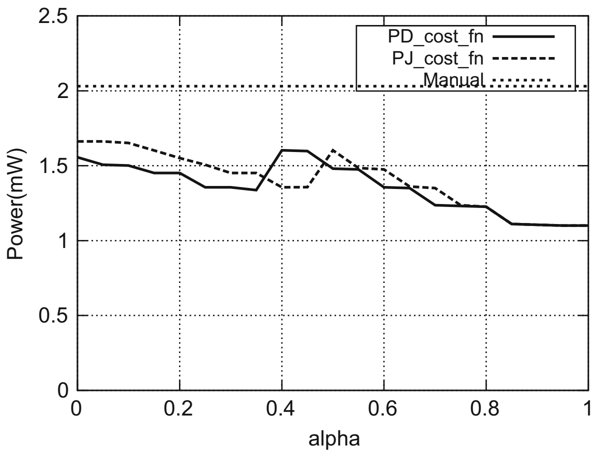


Fig. 2.30 Power (512× Maximum Buffer size)

2.3.4 Experimental Results

We implemented our DP algorithm in the C programming language. We assumed that the H-tree is implemented in a 45 nm (PTM 2013) technology, with $VDD = 1.1\ V$. All the segment configurations were pre-characterized using HSPICE (Inc Meta-Software). We used the *accurate* option in HSPICE. We set the parameter *delmax* set to 1 ps, for maximum accuracy, which ensures that the maximum timestep that HSPICE takes is limited to 1 ps. We observed that the slew rate at the end of each segment was substantially constant for a given segment configuration, independent of the slew rate at the input to the segment.

In order to compare our results with a manually generated buffered H-tree, we consulted an industrial clock tree designer and constructed a minimum delay H-tree based on their input. This H-tree utilized the wire-code with the lowest RC delays (WC-2), with the first 6 segments of length 2.5 mm, and the last 2 segments of length 1.25 mm. 512× buffers used were used exclusively to minimize the delay. The DP algorithms' results were compared with the manually generated H-tree, and the comparison is presented next.

We split the experiments into three sets. In the first set, the maximum buffer size allowed was limited to 512×, while in the second and third sets, up to 256× and 128× buffers were allowed respectively. Figures 2.30, 2.31 and 2.32 corresponds to the first set of experiments (with up to 512× buffers). Figures 2.33, 2.34 and 2.35 corresponds to the second set of experiments (with up to 256× buffers). Finally, Figs. 2.36, 2.37 and 2.38 corresponds to the third set of experiments (with up to 128× buffers). We report the power results, delay results, and the jitter results. For each plot, the x-axis corresponds to α, while the y-axis corresponds to the quantity expressed by the plot. Also, for each plot, the DP results are presented for both cost functions described in Sect. 2.3.3. The results of the manual buffered H-tree are reported in each plot as well.

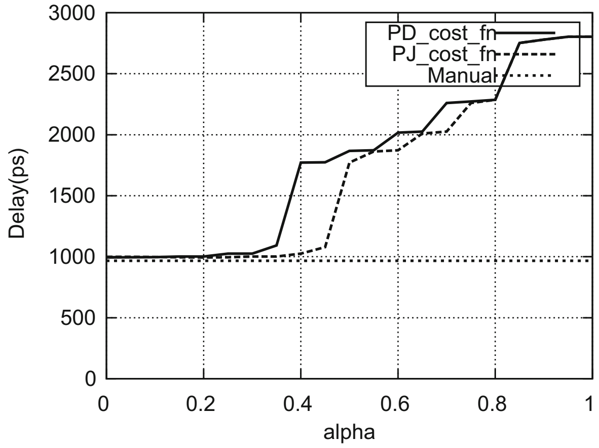


Fig. 2.31 Delay (512× Maximum Buffer size)

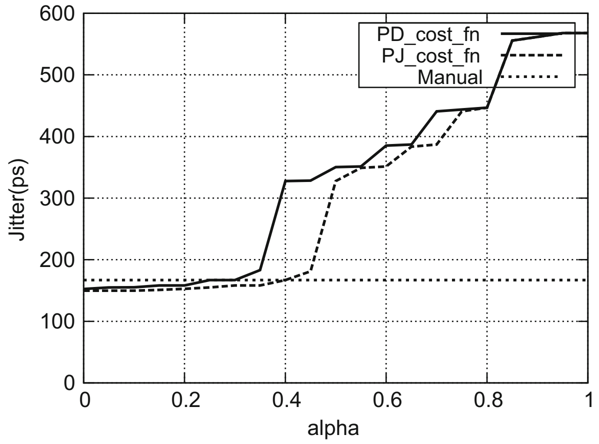


Fig. 2.32 Jitter (512× Maximum Buffer size)

From these plots, we note that for all 3 experiments, the lowest jitter of the buffered H-tree returned by the DP engine is strictly lower than that of the manually synthesized design, by as much as 28 %. This indicates that our DP based buffered H-tree synthesis technique is of value. Further, since our approach is able to select segment configurations freely, it is able to produce strictly better values of power compared to the manual approach, for *all* values of α. The C_{PJ} cost function yields lower jitter values compared to the C_{PD} cost function, especially for $\alpha > 0.5$. For values of $\alpha < 0.5$, C_{PJ} achieves better jitter results than C_{PD}, but the improvement is lower. However, for power-constrained designs, it may be desirable to use $\alpha > 0.5$. The delays of the DP based approaches are higher than the manual design for $\alpha > 0.5$, but this is not of importance for PLL based designs, as we have mentioned earlier.

To validate the accuracy of our DP engine, we simulated the buffered H-tree returned by our DP engine (for the cost functions of minimum jitter, minimum

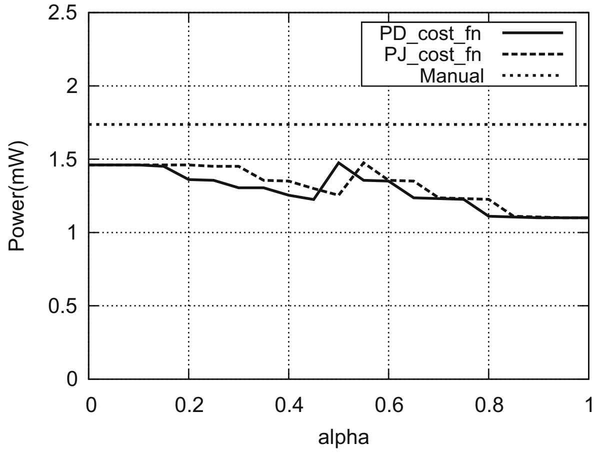


Fig. 2.33 Power (256× Maximum Buffer size)

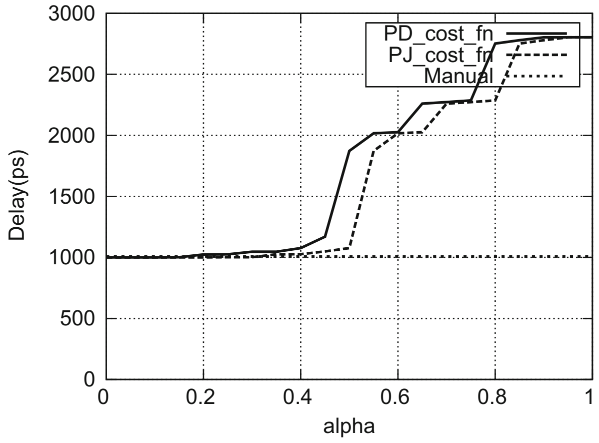


Fig. 2.34 Delay (256× Maximum Buffer size)

power and minimum delay) in HSPICE. The results were compared for the DP engine running with a maximum allowed buffer size of 128×, 256×, and 512× respectively. Table 2.4 reports the results of this comparison. The delay, jitter and power estimated by our DP engine are reported in Columns 3, 4 and 5 respectively. Columns 6, 7 and 8 report the percentage error in the DP estimate of delay, jitter and power respectively, compared to the HSPICE value. Note that the maximum error in any of these estimates is 4.6 %. The DP's estimate of these three quantities was always lower than the value returned by HSPICE.

The segments selected by the DP engine (for a maximum allowable buffer size of 512×) are reported in Tables 2.5 and 2.6 (for the cost function of minimum jitter and minimum delay respectively). Segments with lower indices (Column 1) are closer to the H-tree source. For each segment, we report the segment index, wire-code (Column 2), segment length in mm (Column 3), segment style (one of Straight (*S*),

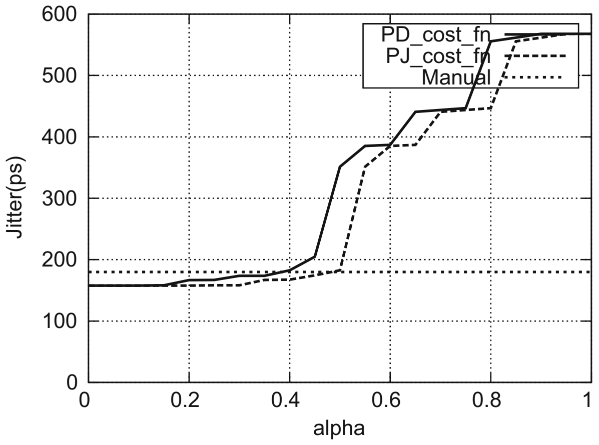


Fig. 2.35 Jitter (256× Maximum Buffer size)

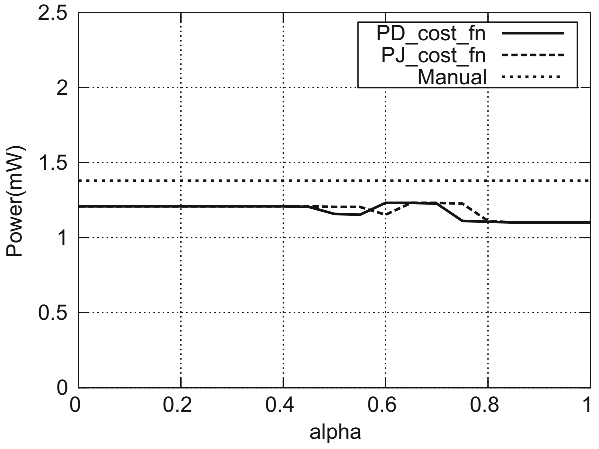


Fig. 2.36 Power (128× Maximum Buffer size)

T or I) (Column 4) and buffer size (Column 5). Note that the segment lengths for Tables 2.5 and 2.6 are quite different, indicating that the minimum delay buffered H-tree is not the same as minimum jitter buffered H-tree. Also, only 2 of the wire-codes end up getting used. All segments use a 512× buffer since power is not constrained.

2.3.5 Conclusion

Buffered clock distribution networks have become increasingly popular due to increasing on-chip wiring delays. Since clock buffers are liable to add jitter in the clock signal on account of cycle-to-cycle *VDD* variations in the die, we argue that the design goal of minimizing end-to-end jitter is more relevant for buffered H-tree synthesis than minimizing end-to-end delay (which is irrelevant for PLL based designs). In this

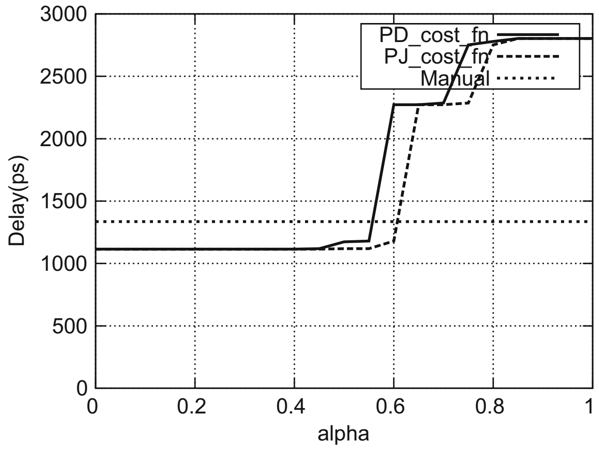


Fig. 2.37 Delay (128× Maximum Buffer size)

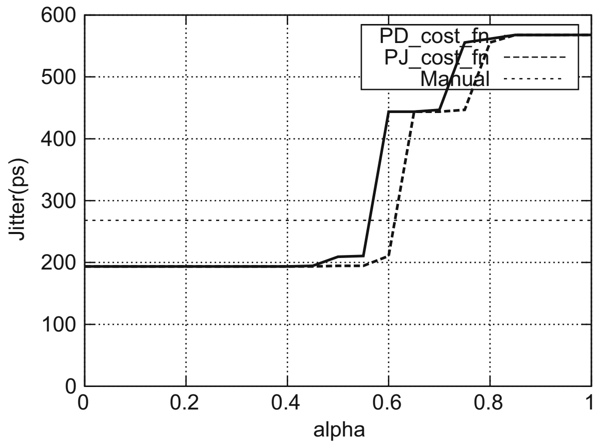


Fig. 2.38 Jitter (128× Maximum Buffer size)

work, we therefore present a dynamic programming based approach to synthesize a minimum cost buffered H-tree clock distribution network. Our cost functions are a weighted sum of power and jitter, Juniper Networks (2010) or a weighted sum of power and end-to-end delay of the distribution network. After pre-characterizing the delay, jitter and power of buffered segments (2745 in all) of different lengths, topologies, buffer sizes and wire-codes, a dynamic programming (DP) engine automatically generates the optimal H-tree that minimizes the appropriate cost function. Compared to a manually constructed buffered H-tree network, our approaches are able to reduce both jitter (by as much as 28 %) and power (by as much as 46 %). When optimizing for minimum jitter, the DP engine generates a H-tree with lower jitter than when optimizing for minimum delay, thereby validating our approach, and proving its utility.

Table 2.4 Comparison of DP results with HSPICE

		DP Estimate			Difference versus HSPICE (%)		
	Cost Func.	Delay (ps)	Jitter (ps)	Power (mW)	Delay (ps)	Jitter (ps)	Power (mW)
Up to 128×	Min. Delay	1114.7	193.5	1.21	0.59	1.88	4.58
	Min. Power	1334.9	247.6	1.08	0.57	1.94	2.44
	Min. Jitter	1114.7	193.5	1.21	0.59	1.88	4.58
Up to 256×	Min. Delay	1000.1	157.7	1.46	0.80	2.23	4.58
	Min. Power	1334.9	247.6	1.08	0.57	1.94	2.44
	Min. Jitter	1000.1	157.7	1.46	0.80	2.23	4.58
Up to 512×	Min. Delay	994.2	152.0	1.55	0.89	2.69	2.52
	Min. Power	1334.9	247.6	1.08	0.57	1.94	2.44
	Min. Jitter	999.3	149.0	1.66	1.11	2.17	2.92

Table 2.5 Segments selected by Our DP Algorithm (Min. Jitter, 512× max)

Seg. #	Wire-code	Length (mm)	Seg. type	Buf. size
1	WC1	1	I	512×
2	WC2	2	S	512×
3	WC2	2	S	512×
4	WC1	1	I	512×
5	WC2	2	S	512×
6	WC2	2	S	512×
7	WC1	1	I	512×
8	WC2	1.5	S	512×
9	WC1	1	I	512×
10	WC2	1.5	S	512×
11	WC2	1.25	I	512×
12	WC2	1.25	I	512×

Table 2.6 Segments selected by our DP Algorithm (Minimum Delay, 512× Maximum)

Seg. #	Wire-code	Length (mm)	Seg. type	Buf. size
1	WC1	1	I	512×
2	WC2	1.75	S	512×
3	WC2	2.25	S	512×
4	WC1	1	I	512×
5	WC2	1.75	S	512×
6	WC2	2.25	S	512×
7	WC1	1	I	512×
8	WC2	1.5	S	512×
9	WC1	1	I	512×
10	WC2	1.5	S	512×
11	WC2	1.25	I	512×
12	WC2	1.25	I	512×

2.4 Tiled SWO-based Clock Distribution

In the above section, we present a dynamic programming based approach to synthesize a minimum cost buffered H-tree clock distribution network. However, in Sect. 2.2, we observe that buffered H-tree suffers from a high jitter. Moreover, buffered H-tree has a high power consumption at higher frequencies. In order to address the above issues, we explore a SWO-based high-speed clock distribution in the following section. We validate that a SWO approach can be used to practically implement a high-frequency, low-power, low jitter clocking approach with high and uniform area coverage over an IC.

2.4.1 Introduction

As discussed in Sect. 2.1, there has been much interest in ring-based resonant oscillators as a means to generate the clock signal in digital ICs. The parasitic inductance and capacitance of the rings are fixed once the ring perimeter, wire dimensions, layer and spacing are determined. The oscillation frequency of a resonant oscillator is determined by the parasitic inductance and capacitance values, provided the inverter pair(s) can switch at this frequency. The equivalent circuit for such a resonant oscillator is shown in Fig. 2.6. In this figure, the parasitic inductance of the ring is referred to as L_w. The parasitic capacitance of the ring is called C_w. The capacitance due to the cross-coupled inverter pair (i.e. twice the sum of the diffusion and gate capacitances of any inverter in the pair) is C. Since C and C_w are in parallel, we obtain the equivalent circuit shown. The oscillation frequency of the equivalent circuit is given by Eq. 2.1.

Although the resonant oscillator structure has tremendous potential as a means to generate a high-frequency on-chip clock signal with low power consumption, it has one fundamental drawback. To enable high-frequency oscillations, the parasitic inductance and capacitance of the ring need to be held at low values, which means that the ring perimeter necessarily needs to be small. Typical values of the ring perimeter are $\sim$2 mm. Since ICs can be as large as 20–30 mm on a side, the resonant clocking idea cannot practically be used to generate a high-frequency $chip - wide$ clock signal. This key focus of this section is to address this issue. Since a SWO Cordero and Khatri (2008) generates a clock signal which has identical phase at each point along the ring (unlike a TWO), we focus our attention on SWOs.

An SWO with a large area coverage on the IC die can be generated in one of two ways.

- **Option A: Implement a $k \cdot \lambda/2$ ring:** Consider an SWO with perimeter p. If we traverse such a ring once, the total phase change is $\lambda/2$. To ensure a larger area coverage, we can increase the size of the ring to $k \cdot p$, where k is odd, while forcing the total phase change over a single traversal of the ring to be $k \cdot \lambda/2$. This approach does not compromise oscillation frequency, and grows the area coverage of the clock signal by a factor of $\sqrt{k}$. This approach was used in Sect. 2.2.
- **Option B: Tile several SWO on a 2D plane:** Another approach is to arrange several identical SWO rings in a 2D tiled structure. Each SWO ring oscillates at the same frequency. We additionally force each adjacent ring to oscillate with an identical phase.

Even though we can set k to a very high value in principle, the first option may be impractical since it realizes a single ring with a very large perimeter, with no clock coverage in the regions in the center of the ring. We overcome this limitation in Sect. 2.2 by snaking the wires of the ring to ensure a uniform clock coverage.

The second approach solves the uniform clock coverage issue, but since each individual SWO has a small perimeter, the second approach alone would require a large number of SWO rings to be implemented. For example, with a chip of size 1 cm on a side and a ring perimeter of 2 mm, 400 SWO rings would be required, making the approach impractical.

In this section, we propose to use a combination of the above two approaches to achieve a large area coverage for the clock signal, in a uniform manner. We present our approach by means of an example in which $k = 3$, and a 3×3 tiling structure is used. We show that $k = 3$ is a good choice since it results in an elegant 2D embedding of the SWO tiles. We have experimented with $k = 5$, 7 and 9 as well, and have validated correct operation of such SWOs. Also, our tiling structure can be easily generalized to an arbitrarily large $n \times n$ arrangement of SWO tiles. With $k = 3$, and a chip of size 1 cm on a side, the number of required SWO rings for complete chip coverage is reduced by a factor of 8, compared to the case where $k = 1$.

The remainder of this section is organized as follows: Previous work is described in Sect. 2.4.2, while Sect. 2.4.3 provides the details of our tiled high-speed, low-power and high area coverage clock distribution strategy using SWOs. In Sect. 2.4.4 we present results from experiments which we conducted to validate our approach. We conclude in Sect. 2.4.5.

2.4.2 Previous Work

In Sect. 2.1, we introduced resonant ring-based oscillators. A traveling wave resonant oscillator (TWO) circuit (referred to by the authors as a *rotary clock*) was described and implemented (Wood et al. 2001; 2006). The key idea in this approach is to utilize a sufficiently long wiring ring, such that its capacitive and inductive parasitics result in a high frequency oscillatory network. The main problem with a TWO is the phase change incurred around the ring. In response to this, a standing wave resonant oscillator circuit was proposed (Cordero and Khatri 2008). The clock signal at any point in the ring is sinusoidal, but has the *same phase* at all points along the ring. To recover a full-rail clock anywhere along the ring, differential amplifiers need to be connected to the ring signals at these locations.

In (Mahony et al. 2003; Karkala et al. 2009), a high-frequency standing wave oscillator was used to implement a clock Phase Locked Loop (PLL). The work of (Mahony et al. 2003) is based on the use of multiple coupled oscillators (each comprised of an NMOS cross-coupled pair to sustain the oscillation, and a PMOS diode connected load for setting the common mode voltage). The approach of Karkala et al. (2009) implements a resonant SWO based PLL, with an inductance control based coarse frequency adjustment mechanism. Fine frequency adjustment is achieved by controlling the body bias of the PMOS transistor of the inverter pair. Unlike the approach proposed in this work, these approaches did not address the key problem of the IC area coverage of resonant SWOs or TWOs. This work has the ability to cover large die areas with a high-frequency, low power clock signal by using interlinked tiled SWOs, each of which have a total phase change of $3\lambda/2$ across the ring.

In Mahony et al. (2003), the authors present a tiled SWO based resonant grid for high frequency clock distribution. Each SWO is implemented as $\lambda/2$ ring, using a short circuit at the far end (instead of a mobius connection as in our case). Multiple SWOs are coupled by injection locking. The key differences between (Mahony et al. 2003) and our approach are i) we utilize $3\lambda/2$ rings (which results in fewer SWOs being required) and ii) we utilize a mobius termination in each SWO ring, while (Mahony et al. 2003) utilizes a short circuit termination for each SWO ring. It was shown (Cordero and Khatri 2008) that short circuit termination results in a lower oscillation frequency as well as higher power in comparison to a mobius termination based SWO.

2.4.3 Our Approach

Resonant oscillators (SWOs as well as TWOs) are a promising technique to generate a high-frequency on-chip clock signal with low power. However, they possess a key weakness when used in typical ICs, where the goal is to uniformly distribute a chip-wide, high-frequency, low power clock signal. To achieve a high frequency of operation, the typical values of inductance and capacitance required for the resonant oscillators are such that the total perimeter of the resonant ring is small (typically $\sim$ 2 mm). Since many complex ICs can be as large as 20-30 mm on a side, the ratio of the chip area to the area covered by a typical resonant ring is as high as $\sim$ 3600. Hence it would be impossible to distribute a chip-wide, high-frequency resonant clock signal with a single SWO or TWO ring. Our goal is to present approaches to achieve complete and uniform area coverage of the resonant clock signal across the IC die. By uniform area coverage, we mean that at any position on the IC die, a resonant clock signal is no further away than the perimeter of an individual resonant ring (i.e. $\sim$ 2 mm). Since a SWO (Cordero and Khatri 2008; Karkala et al. 2009) generates a clock signal which has identical phase at each point along the ring (unlike a TWO), we focus our attention on SWOs.

From the equivalent circuit for our resonant oscillator (Fig. 2.6) and the equation for the oscillation frequency of the resonant oscillator (Eq. 2.1), we observe that increasing the perimeter of the ring is not an acceptable option to achieve high clock coverage on the IC die. This is because increasing the perimeter of the ring increases both C_w and L_w linearly, resulting in an unacceptable drop in clock frequency. As a result, we explore two alternative options:

Option A: For an SWO with perimeter p, if we traverse the ring once, the total phase change is $\lambda/2$. To ensure a larger area coverage, we can increase the perimeter of the ring to $k \cdot p$, where k is odd, thereby making the total phase change over a single traversal of the ring to be $k \cdot \lambda/2$. In such a design, we require p equally spaced inverter pairs, and an odd number (typically one) of mobius connections. The circuit configuration for a $3 \cdot \lambda/2$ ring is shown in Fig. 2.7. Note that it in order to ensure that the resonant structure bootstraps in a standing wave configuration, the signals at the inverter pair are initialized using a global bootstrap signal (labeled *BS* in Fig. 2.7). We have validated that the $k \cdot \lambda/2$ SWO ring oscillates correctly and reliably, and at the same frequency as the corresponding $\lambda/2$ ring, for $k = 3, 5, 7$ and 9.

Although the $k \cdot \lambda/2$ approach does not compromise oscillation frequency, and also increases the area coverage of the clock signal by a factor of $\sqrt{k}$, it still possesses a significant drawback. For large ICs, the value of k needs to be significantly large. For example, for a chip of size 30 mm on a side, a k value of about 61 is required. With such a configuration, a key problem is the uniformity of the coverage of the clock across the die. The center of the ring in such a case is about 15 mm from the nearest resonant clock location, making the approach impractical. Implementing the resonant SWO ring in a snaked manner (as designed in Sect. 2.2) is one solution to this problem.

Option B: Another approach is to arrange several identical $\lambda/2$ SWO rings in a 2D tiled structure. Each SWO ring oscillates at the same frequency. We additionally force adjacent rings to oscillate with an identical phase by introducing an appropriate number of shorts across these rings. Suppose our chip size is 10 mm on a side. In this case, assuming $\lambda/2 = 2$ mm, we would require 400 SWO rings. The advantage of this approach is that it enables us to achieve a uniform and complete area coverage of the clock signal on the IC die, but the drawback is that we need a large number of SWOs.

Option C: The third option is a hybrid of the Options A and B. In this case, we arrange several identical $k\lambda/2$ SWO rings in a 2D tiled structure. This approach therefore retains the best features of both Options A and B. This work utilizes Option C (with $k = 3$) to implement a complete and uniform chip-wide resonant clock distribution network.

Next, we discuss the details of our approach.

2.4.3.1 Tiled SWO Topology

For a tiled $k\lambda/2$ SWO, we first need to choose the value of k. The problem becomes that of embedding a regular k-sided polygon on a plane. This is illustrated via Fig. 2.39.

The internal angle of a regular k-sided polygon is given by $Z = \frac{(n-2) \cdot 180^{\circ}}{n}$. In order that the k-sided polygon can be embedded on a plane, we require that $nZ = 360°$, where n is an integer. Given that k is odd, the only value of k that satisfies the above condition is 3. Hence we choose $k = 3$. Figure 2.39 illustrates how a uniform triangle can be embedded on a plane, while a uniform pentagon cannot. Note that each dot in Fig. 2.39a represents six inverter pairs (one for each SWO ring).

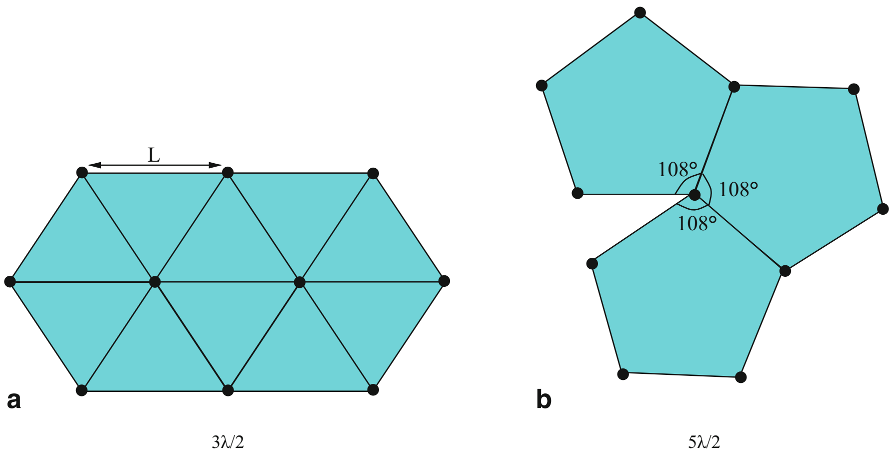


Fig. 2.39 Embedding a triangle and a pentagon on a 2D plane

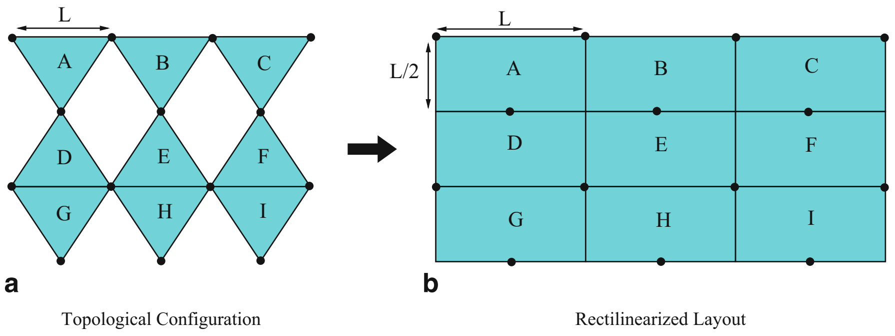


Fig. 2.40 Rectilinear realization of a triangle on a 2D plane

Note that the embedding of the equilateral triangle on a 2D plane shown in Fig. 2.39 cannot be directly implemented in a VLSI IC, since wires on an IC are constrained to be rectilinear. In order to perform the embedding of a $3\lambda/2$ SWO ring on a 2D surface (using rectilinear wires), we first remove every alternate SWO ring. Now the resulting structure (shown in Fig. 2.40a) has half as many SWO rings. Each dot in this figure represents 2 inverter pairs in the even numbered rows, and 4 inverter pairs in the odd numbered rows. We rectilinearize the segments of Fig. 2.40a, and the result is shown in Fig. 2.40b.

In order to achieve the rectilinearized embedding of $3\lambda/2$ SWO rings, we transform each non-rectilinear wire of the embedding of Fig. 2.40a and convert it into a single wire with a horizontal and a vertical segment. Thus each $3\lambda/2$ ring is transformed into a rectangle with length L and height $L/2$, where L is the perimeter of the corresponding $\lambda/2$ ring. Each edge in Fig. 2.40b represents 4 wire segments, with one pair of wire segments being utilized by each of the two separate $3\lambda/2$ rings which share the edge. Each dot of Fig. 2.40b represents 2 inverter pairs, with each inverter pair being utilized by the two separate $3\lambda/2$ rings which share the dot.

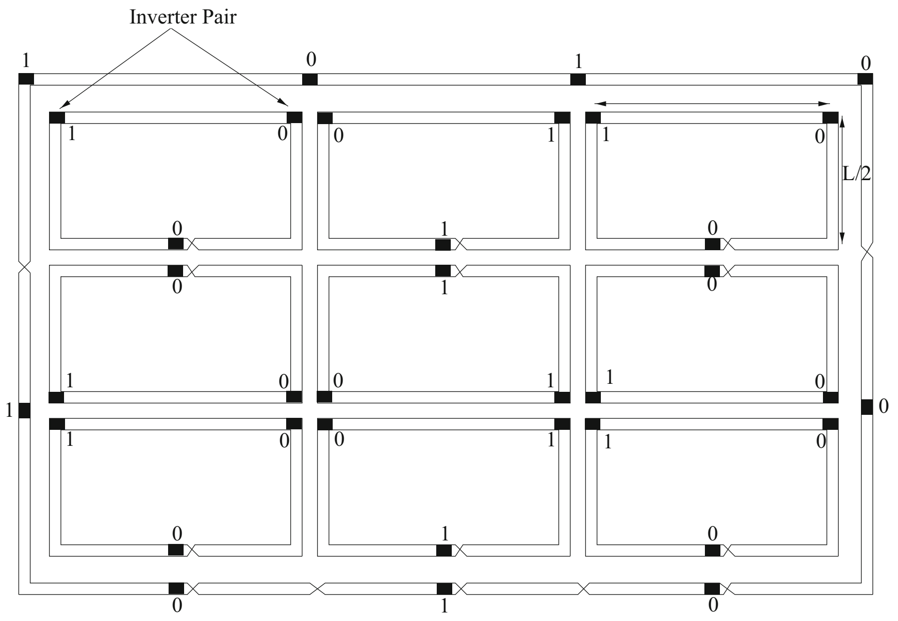


Fig. 2.41 Layout organization of a 3x3 SWO tile

A more detailed view of the tiled $3\lambda/2$ SWO rings (for a 3×3 tiled array) is shown in Fig. 2.41. Each ring consists of 2 (inner) wires, with two outer wires corresponding to rings that are above, below, or on either side of the said ring. We need to make all rings oscillate with the same frequency and phase. In addition, we need to ensure that all 4 wires of any segment, at any location, have the same voltage at any time instant. In order to guarantee these conditions, we utilize bootstrap devices, to force initial conditions at various locations along the tiled SWO structure. Although the bootstrapping devices are not shown in Fig. 2.41, the values that are asserted at various locations in the ring by these devices are shown (by means of the '0' and '1' labels). In order to guaranteed that all rings oscillate with the same frequency and phase, it is crucial to ensure that the electrical environment around each location of any ring is identical to the electrical environment around the same location of all other rings. In order to do this, we insert an outermost peripheral ring as well, whose length is 9 L. This ring also oscillates, ensuring that the electrical environment of each tiles is identical. Note that the mobius connections of each of the tiles are illustrated in Fig. 2.41. The outer ring has 4 extra mobius flips (in addition to those required to sustain oscillations) shown along its lower edge. These flips are introduced in order to ensure that every location of the outer wire oscillates with the same frequency, phase and amplitude as the wire in the SWO tile adjoining it. We experimented with several ways of connecting the outer ring (such as grounding it at regular intervals and leaving it floating), and found that in order to ensure low jitter and uniform oscillation frequency, it was essential to connect the outer ring in the configuration shown in Fig. 2.41.

2.4.4 Experiments

We implemented the tiled SWO described in this work, using a 90 nm BSIM3 (PTM 2013) process technology. The power supply voltage was 1.2 V, and all simulations were conducted in HSPICE (Inc Meta-Software). We simulated a 3×3 tiling structure (as described in Fig. 2.41). The ring consisted of 7 wires in all, each of which had a width of 20 μm and a inter-wire spacing of 20 μm as well. The outermost and innermost wires as well as the middle wire are connected to ground, with the remaining wires are utilized to carry the 4 oscillating signals. The RLC parasitics of the 7-wire bundle were extracted using (Raphael Interconnect Analysis Tool: User's Guide), and adjusted for skin-effect in our simulations. The nominal oscillation frequency of each of the 9 SWO rings was 7.267 GHz (yielding a nominal clock period T_{nom} = 137.6 ps. Each SWO ring consists of 72 smaller segments in our HSPICE simulation deck.

The two wires of any SWO ring sustain a sinusoidal oscillation. To recover a rail-to-rail clock from any point on the ring, a clock recovery circuit (shown in Fig. 2.3) is required. This circuit is essentially a differential amplifier with a buffered output. We implemented 42 regenerator circuits per SWO ring. An overlay plot of all 42×9 recovered signals is shown in Fig. 2.42. From this figure, we observe that the falling skew is 4.56 ps, while the rising skew is 1.45 ps (for a clock of period of 137.6 ps).

The power consumption of our oscillator is 55.07 mW per SWO ring (without any regenerators) and 68.56 mW per SWO ring (with 42 regenerators per SWO ring). This would indicate that for a chip with size 10 mm per side, the total power consumption in the clock distribution network would be $\sim$ 2.75 (3.43) Watts without (with) regenerators.

To measure the quality of the tiled SWO based clock distribution network, we report several quantities obtained after simulating the structure for 140 clock cycles. These quantities serve as figures of merit of the design, and are listed below:

- We computed, at each of the 27 inverter pair sites (3 for each of the 9 SWO rings), the clock period for each clock cycle. Let T_{max} and T_{min} be the maximum and minimum clock periods, and $\Delta_T = T_{max} - T_{min}$. The worst case value of $\frac{\Delta_T}{T_{nom}}$ over all the 27 inverter pair locations was 1.56 % (measured at the ring) and 2.61 % (measured after the regenerators).
- Recall that the location between two inverter pairs is a virtual ground location. Therefore the amplitude of the sinusoidal signal on either side of the virtual ground location is small, making it hard to reliably regenerate a square wave clock from the ring locations on either side of the virtual ground. In our experiments, we did not connect regenerators to ring locations which were within .345 mm on either side of a virtual ground node (which yielded 42 regenerators per ring, for a total of 42×9 = 378 regenrators). We found that the worst case value of $\frac{\Delta_T}{T_{nom}}$ over all points (where the clock can be extracted) over all rings was 1.89 % (measured at the ring) and 3.13 % (measured after the regenerators).

Figure 2.43 displays an overlay plot of 3 virtual ground nodes (for rings A, G and H as labeled in Fig. 2.40b). The virtual ground waveforms have a peak-to-peak voltage

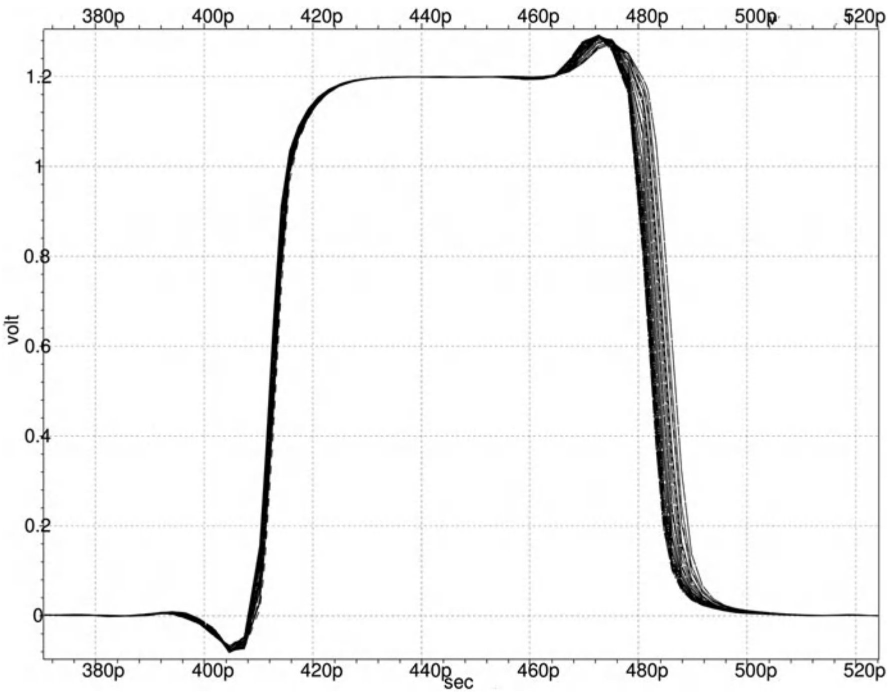


Fig. 2.42 Overlay of recovered waveforms from all 378 regenerators

of about 200 mV over all 27 virtual ground locations. Finally, Fig. 2.44 displays the overlay plot of the SWO ring waveforms (for ring A). The waveforms correspond to all 24 internal ring nodes encountered between two adjacent inverter pairs of ring A.

We also computed the Q factor for any of the 9 rings of our tiled SWO oscillator. To compute the Q factor, we first removed the inverter pairs of the ring, and replace them by the equivalent average capacitance of the inverter pair terminals. Now an differential AC current with differential amplitude of 1 A is applied across these terminals. The resulting voltage across the terminals is measured as a function of the frequency of the differential current. The voltage has a peak at the oscillation frequency of the ring. The Q factor is computed by finding the ratio of the resonant frequency to the 3dB bandwidth of the voltage waveform. We determined the Q factor of of our tiled SWO oscillator to be about 19.

2.4.5 Conclusion

Resonant oscillators can sustain extremely high oscillation frequencies with very low power consumption. However, a single resonant oscillator covers a very small fraction of the area of a typical IC. In this work, we present an approach to completely and uniformly cover an IC using an SWO. This is achieved by combining two

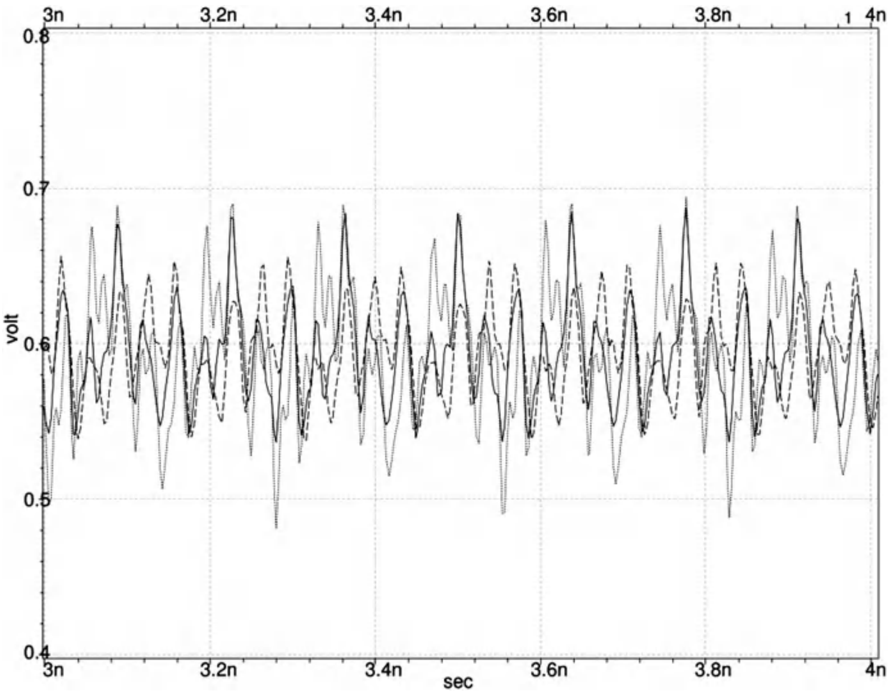


Fig. 2.43 3 Overlaid virtual ground waveforms

techniques. The first technique increases the area coverage of an individual SWO by ensuring that it sustains 3 standing waves along the ring. The second approach further increases the area coverage by tiling multiple such SWOs side by side, and connecting them such that they oscillate with the same high frequency and phase. We carefully ensure that the electrical environment around each SWO ring is identical. Skin effect adjusted 3D RLC parasitics are utilized for our experiments. For a 90 nm process, our tiled SWO based resonant clock distribution approach achieves an oscillation frequency of about 7.267 GHz, with a low power consumption of about 68.5 mW per SWO ring, and a jitter of 3.1 % of the nominal clock period.

2.5 Chapter Summary

In this chapter, we address the issue of distributing a high-speed, low power, low jitter clock with the aim of serving the NoC and the PEs in a CMP. In Sect. 2.1, we introduce resonant ring-based oscillators, which have recently emerged as a promising technique for high-speed, low power clock generation. In Sect. 2.2, we use a resonant SWO based phase-locked clock generation and distribution scheme with superior jitter, skew and power characteristics. Phase locking is accomplished

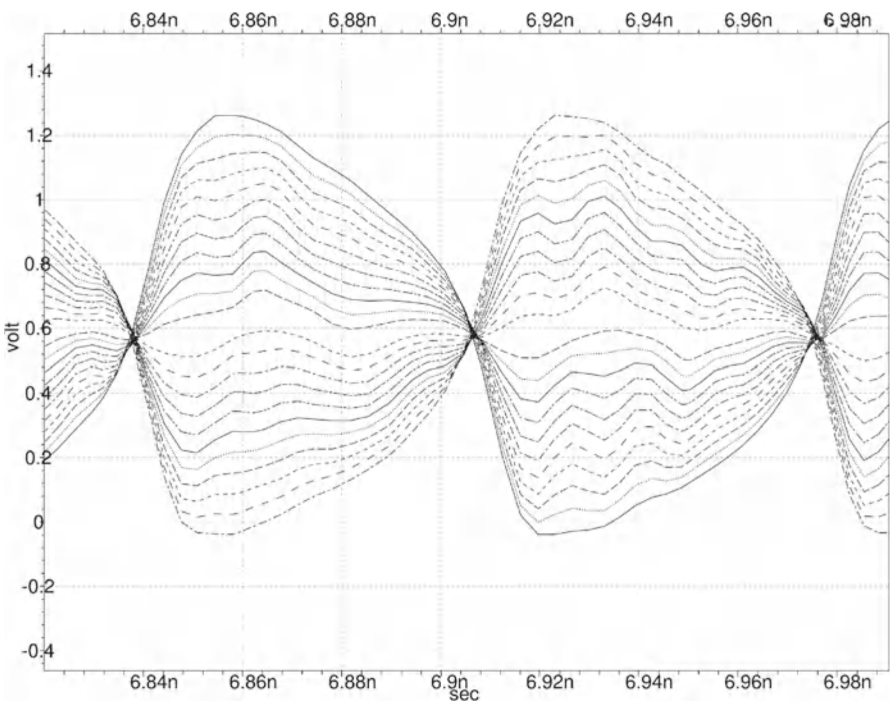


Fig. 2.44 Overlaid waveforms of ring signals between 2 Adjacent inverter Pairs

by a fully digital control loop consisting of a coarse frequency control and a fine frequency control. The SWO frequency is modulated in a coarse manner by changing the ring inductance, by modifying the number of oscillating wires. Fine frequency control is achieved by modifying ring capacitance via a binary-weighted capacitor bank. Clock distribution is performed by routing the resonant ring over the die in a "comb" manner. Our approach is compared to an H-tree with an overlaid mesh. Given the relatively high Q-factor of the SWO, this clocking approach exhibits a very tight cycle-to-cycle jitter (about 4.6× better) and very low skew (about 5.5× better). The power consumption of our scheme is upwards of 2× improved as well, given the resonant nature of the approach.

In recent fabrication technologies, increasing on-chip wiring delays have contributed to the popularity of buffered clock distribution network. In Sect. 2.3, we present a dynamic programming based approach to synthesize a minimum cost buffered H-tree clock distribution network. Our two cost functions are a weighted sum of power and jitter, and a weighted sum of power and end-to-end delay of the distribution network respectively. After pre-characterizing the delay, jitter and power of buffered segments (2745 in all) of different lengths, topologies, buffer sizes and wire-codes, we invoke a dynamic programming (DP) engine to automatically generate the optimal H-tree that minimizes the appropriate cost function. Compared to a manually constructed buffered H-tree network, our approaches are able to reduce both jitter (by as much as 28 %), and power (by as much as 46 %). When optimizing for minimum jitter, the DP engine generates a H-tree with lower jitter than when optimizing for minimum delay, thereby validating our approach, and proving its utility.

In general, SWO based clock distribution networks can suffer from a non-uniform clock coverage at different die locations. In order to address this issue, we explore a tiled SWO-based high-speed clock distribution in Sect. 2.4. We validate that a SWO approach can be used to practically implement a high-frequency, low-power, low jitter clocking approach with high and uniform area coverage over an IC. This is achieved by combining two techniques. The first technique increases the area coverage of an individual SWO by ensuring that it sustains 3 standing waves along the ring. The second approach further increases the area coverage by tiling multiple SWOs side by side, and connecting them such that they oscillate with the same high frequency and phase. We carefully ensure that the electrical environment around each SWO ring is identical. Skin effect adjusted 3D RLC parasitics are utilized for our experiments. Our tiled SWO based resonant clock distribution approach achieves an oscillation frequency of about 7.267 GHz, with a low power consumption of about 68.5 mW per SWO ring, and a jitter of 3.1 % of the nominal clock period.

References

F. Anceau, “A synchronous approach for clocking VLSI systems,” *Solid-State Circuits, IEEE Journal of*, vol. 17, no. 1, pp. 51-56, Feb 1982.

Y Chen and D. F. Wong, “An algorithm for zero-skew clock tree routing with buffer insertion,” *Design and Test of Computers*, pp. 230-236, 1996.

S.C. Chan, P.J. Restle, K.L. Shepard, N.K. James, and R.L. Franch, “A 4.6 GHz resonant global clock distribution network,” *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, pp. 342-343 Vol.1, Feb. 2004.

T-H Chao, Y-C Hsu, J-M Ho, and A. B. Kahng, “Zero skew clock routing with minimum wirelength,” *IEEE Transactions on Circuits and Systems II*, vol. 39, pp. 779-814, Nov 1992.

R. Chaturvedi and J. Hu, “Buffered clock tree for high quality IC design,” in *Proceedings, International Symposium on Quality Electronic Design*, 2004, pp. 381-386.

J-Y Chueh, C Ziesler, and M Papaefthymiou, “Experimental evaluation of resonant clock distribution,” in *Proceedings, IEEE Computer society Annual Symposium on VLSI, 2004*, Feb 2004, pp. 135-140.

J Cong et al., “Bounded-skew clock and Steiner routing,”*ACM Transactions on Design Automation of Electronic Systems*, vol. 3, pp. 341-388, 1998.

V Cordero and S Khatri, “Clock distribution scheme using coplanar transmission lines,” in *DATE*, 2008, pp. 985-990.

E.G. Friedman, “Clock distribution networks in synchronous digital integrated circuits,” *Proceedings of the IEEE*, vol. 89, no. 5, pp. 665-692, may 2001.

Juniper Networks San Jose CA N Jayakumar, VLSI Design Engineer, “Private Communication,” July 2010.

Inc Meta-Software, “HSPICE user’s manual,” Campbell, CA.

V. Karkala, K.C. Bollapalli, R. Garg, and S.P. Khatri, “A PLL design based on a standing wave resonant oscillator,” in *IEEE International Conference on Computer Design (ICCD) 2009*, Oct 2009, pp. 511-516.

T. H. Lin and W. J. Kaiser, “A 900-MHz 2.5-mA CMOS Frequency Synthesizer with an Automatic SC Tuning Loop,” *IEEE Journal of Solid-State Circuits*, vol. 36, no. 3, pp. 424-431, 2001.

I. Liu, T. Chou, A. Aziz, and D. F. Wong, “Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion,” in *Proceedings, Intl Symposium on Physical Design*, 2000, pp. 33-38.

T. H. Lin and Y. J. Lai, “An Agile VCO Frequency Calibration Technique for a 10-GHz CMOS PLL,” *IEEE Journal of Solid-State Circuits*, vol. 42, no. 2, pp. 340-349, 2007.

F. O’Mahony, “10 GHz Global Clock Distribution using Coupled Standing-Wave Oscillators,” Ph.D. dissertation, Stanford University, 2003.

F O’Mahony, P Yue, M Horowitz, and S Wong, “Design of a 10GHz clock distribution network using coupled standing-wave oscillators,” in *DAC ’03: Proceedings of the 40th conference on Design automation*. 2003, pp. 682-687, ACM.

“MultiGig,” http://www.multigig.com (Accessed April 22, 2013).

N. Nedovic, N. Tzartzanis, H. Tamura, F. M. Rotella, et al., M. Wiklund “A 40-44 Gb/s 3 X Oversampling CMOS CDR/1:16 DEMUX,” *IEEE Journal of Solid-State Circuits*, vol. 42, no. 12, pp. 2726-2735, 2007.

“PTM website, ” http://www.eas.asu.edu/˜ptm (Accessed April 22, 2013).

A. Rajaram and D Pan, “Variation tolerant buffered clock network synthesis with cross links,” *Proceedings, Intl Symposium on Physical Design*, pp. 157-164, 2006.

“Raphael Interconnect Analysis Tool: User’s Guide,” .

W. B. Wilson, U. K. Moon, K. Lakshmikumar, and L. Dai, “A CMOS Self-Calibrating Frequency Synthesizer,” *IEEE Journal of Solid-State Circuits*, vol. 35, no. 10, pp. 1437-1444, 2000.

Wood et al. 2001 J, T Edwards, and S Lipa, “Rotary traveling-wave oscillator arrays: a new clock technology,” *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1654-1665, Nov 2001.

J. Wood, T. Edwards, and C. Ziesler, “A 3.5 GHz Rotary-Traveling-Wave-Oscillator Clocked Dynamic Logic Family in 0.25 m CMOS,” *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, pp. 1550-1557, Feb. 2006.

itrs“The International Technology Roadmap for Semiconductors,” http://public.itrs.net/ (Accessed April 22, 2013), 2007.

J-L Tsai, T-H Chen, and C Chen, “Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing,” *IEEE Transactions on CAD*, vol. 23, pp. 565-572, 2004.

Chapter 3
Fast Network-on-Chip Design

Abstract In previous Chapter, we showed how resonant clocking can be used as a high-speed, low power, stable, on-chip clock generation and distribution schemes. In this chapter, we use such a clock to design a high speed source-synchronous ring-based NoC architecture. In Sect. 3.1, we introduce our NoC design, which comprises of extremely fast, intersecting source-synchronous data rings. These source-synchronous data rings traverse the CMP in both the horizontal and vertical directions providing complete connectivity to all the PEs in a CMP. In our approach, the interconnection network operates on a different clock domain which runs significantly faster than the PE clocks. This helps us achieve inter-processor communication with minimal latency. We perform architectural simulations of the ring-based NoC in Sect. 3.2. We propose a deadlock-free routing protocol of the source-synchronous ring-based NoC by using link ordering and virtual channel based buffered flow control. Architectural results obtained on synthetic and real traffic demonstrate that the source-synchronous ring-based NoC has significantly lower latency and higher maximum sustained injection rate compared to a state of the art mesh-based NoC. Next, in Sect. 3.3, we propose a modified source-synchronous design in which the PEs extract a low jitter clock directly from the high speed ring clock by division, and hence are synchronous with the NoC. This is feasible due to the extremely good jitter characteristics of the SWO based clock generation and distribution scheme of Sect. 2.2. Using the above modified design, we propose a class of source-synchronous NoCs organized in an H-tree topology which consume lower logic and wiring area compared to a state of the art mesh. Architectural simulations on synthetic and real traffic show that our H-tree based NoC designs can provide significantly lower latency and are able to sustain a higher injection rate compared to a state of the art mesh. Using the modified source-synchronous design proposed in Sect. 3.3, we also evaluate two more floorplan-friendly NoC topologies in Sect. 3.4. These two floorplan-friendly NoC topologies consume significantly lower logic and wiring area compared to a state of the art mesh. Architectural simulations on synthetic and real traffic show that they can provide significantly lower latency while achieving same or better maximum sustained injection rate compared to a state of the art mesh.

DOI 10.1007/978-1-4614-9405-8_3, © Springer Science+Business Media New York 2014

3.1 Circuit Design of a Source Synchronous Ring-based NoC

In this section, we introduce our source-synchronous ring-based NoC design. Our proposed NoC runs significantly faster than the PEs, which help us achieve an ultra low latency inter-processor communication.

3.1.1 Introduction

As discussed in Sect. 1.4, multi-core processors require an efficient and scalable NoC infrastructure to handle the inter-processor on-chip communication needs. There has been significant research in NoC topologies (Bononi et al. 2007; Kim et al. 2008; Tota et al. 2006), global wire management (Balasubramonian et al. 2005), and power optimization (Peh et al. 2003) in this context. In terms of topology, a 2D mesh interconnection network has received the greatest attention by NoC designers due to its simple implementation, high bandwidth and overall scalability. However the large diameter of the mesh has a negative effect on the communication latency. Other popular topologies include Ring (Samuelsson and Kumar 2004), Fat Tree Leiserson (1985), 2D Flattened Butterfly (Kim 2007), Octagon (Karim 2002) and Torus (Duato 2002).

As the size of a chip multi-processor (CMP) grows, it becomes increasingly difficult to distribute a synchronous clock over the entire chip. Hence, designers are opting for a separate synchronous communication network between the regions on the die, while each region is clocked in a synchronous manner. This is referred to as multi-synchronous or mixed clock communication approach. Most of the NoCs are implemented using the multi-synchronous paradigm. In this section, we propose a high-speed, source-synchronous NoC architecture, where the PEs communicate with the NoC using the multi-synchronous paradigm.

In Sect. 1.7, we showed that resonant clocking can be used to develop a high-speed, low power, stable on-chip clock generation and distribution schemes. In this work, we utilize such a clock to develop a high-speed, source-synchronous NoC architecture. Each PE operates in a synchronous manner, and is assumed to operate at 2 GHz. The NoC is comprised of a series of (horizontally and vertically arranged) flattened rings. Each ring operates significantly faster (about 7× faster from our HSPICE (Inc Meta-Software) simulations) than the PEs. The data on the ring is transmitted in a source synchronous manner with reference to a fast resonant clock. A *Junction Station* (JS) is placed wherever these rings intersect, allowing data to switch between the horizontal ring and the vertical ring (or vice versa) at the junction. Each PE is connected to a ring by means of an *Insertion-Extraction Station* (IES), which allows it to insert/extract data into/from the ring. Each IES contains two asynchronous FIFOs. One FIFO is written from the ring NoC, and read by the PE. The other FIFO is written by the PE and read by the ring NoC driver logic.

The block diagram of a single (flattened) ring of our ring based NoC is shown in Fig. 3.1. Note that since this figure depicts a single ring, JS' are not shown. The ring

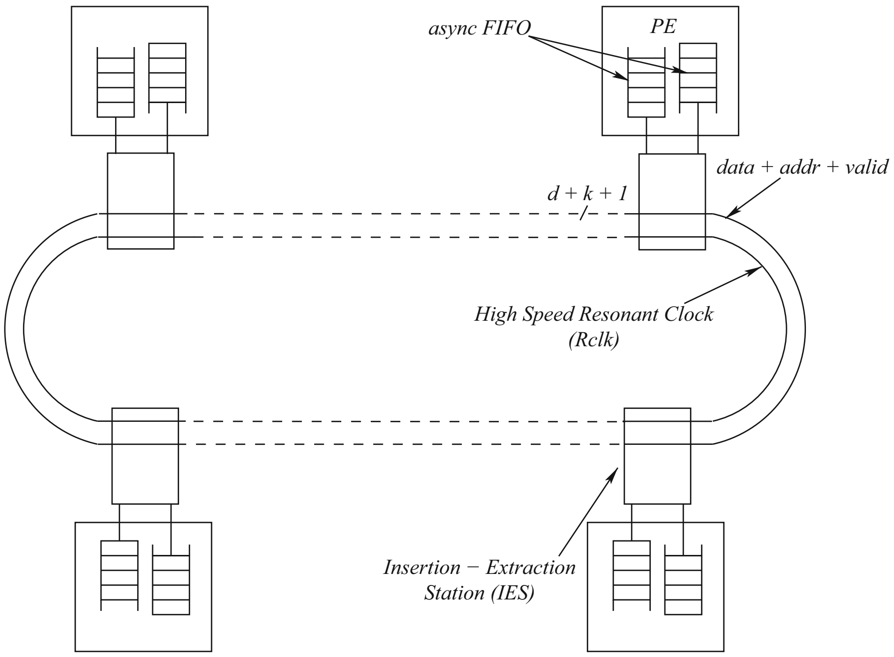


Fig. 3.1 Single ring based NoC architecture

carries three fields of information—*d* bit *data*, *k* bit *address*, and 1 *valid* bit. If there are P PEs in the CMP, then $k = log_2 P$. A high speed resonant clock signal *Rclk* runs parallel to the ring signals mentioned above, as shown in Fig. 3.1. The data, address and valid signals are source synchronous with the *Rclk* signal.

The rest of this section is organized as follows: Sect. 3.1.2 describes some popular approaches for NoC design. Section 3.1.3 presents our approach on high-speed NoC design, while Sect. 3.1.4 describes the results of HSPICE Inc Meta-Software (Inc Meta-Software) experiments which we performed to validate our approach. In Sect. 3.1.5, we draw conclusions.

3.1.2 Previous Work

Any NoC architecture should have a combination of the following desirable features: (1) scalability and modularity, (2) low interconnect latency, (3) minimal power and (4) high link data-rates. There have been several NoC topologies that have been developed to satisfy these requirements.

For small NoCs, a good solution is the crossbar, which connects each pair of the PEs with dedicated links. However, this topology has a complexity that is quadratic in the number of PEs, and does not scale well. The simplest and most ubiquitous

NoC topology is the 2D mesh. Dally and Towles (Dally and Towles 2001) first proposed a 2D mesh as a NoC architecture. The topology consists of a 2D mesh of wires, with routers at the intersections of horizontal and vertical wires. Every router has five ports, one connected to the local resource (PE) and the others connected to the closest neighboring routers. The torus architecture was proposed in (Duato 2002), with routers at the edges connected to the routers at the opposite edge through wrap-around channels. The long end-around connections can yield excessive delays which can be avoided by folding the torus (Dally and Seitz 1986). The inherent disadvantage of the mesh or torus topologies is their large communication radius, resulting in large amounts of interconnect and large numbers of arbiters at the N-S-E-W crossings. These in turn leads to a large power consumption. Karim et. al. Karim (2002) proposed the Octagon architecture. The Octagon network consists of a basic *octagon unit* having eight nodes and 12 bidirectional links. It has a simpler implementation compared to the 2D mesh, with a higher throughput. Unlike the crossbar, the Octagon's implementation complexity increases linearly with the number of nodes (PEs). A 2D Flattened Butterfly Kim (2007) is derived by flattening the routers in each row of a conventional butterfly topology and hence provides the connectivity of a mesh with additional links. The Fat Tree Leiserson Leiserson (1985) connects routers in a tree manner, with sources and destinations at the leaves. The major advantage of the Fat Tree is the large amount of bandwidth available, with the downside of the requirement for large-radix routers toward the root of the tree. In Thonnart (2010), the authors propose an asynchronous 2D mesh NoC infrastructure. Compared to a synchronous mesh, their realization increases the area utilization by more than $3\times$, with a 30–50 % gain in speed and a $5\times$ reduction in power and a $6.9\times$ energy improvement. The Ring Samuelsson and Kumar (2004) topology implements concentric connected rings (similar to ring road in city), which helps to reduce the risk of congestion in the central parts of the network. The approach is motivated by the smooth flow of traffic in ring roads. Their approach is fundamentally different than ours in its topology, as well as in the fact that their simulations are conducted purely at the architectural level.

In Ludovici et al. (2011), the authors implemented a power-efficient mesochronous NoC which is mesochronous with the PEs with no area and latency overhead. Our NoC design is multi-synchronous, and it operates at a significantly higher speed than the PEs. Additionally Ludovici et al. (2011) uses a traditional mesh topology in contrast with our source-synchronous ring.

In all the above implementations, design decisions are made based on the fact that the interconnection network operates at the same or lower frequency as the PEs. In contrast, our focus is an NoC architecture which runs significantly faster ($7\times$ in our simulation) than the PEs. This allows more architectural flexibility compared to existing NoC solutions. For example, the significantly higher bisection bandwidth allows the designer to implement our NoC architecture with narrower links (yielding a lower area and power for the same bisection bandwidth). The significantly lower latencies allow our approach to scale more elegantly for larger CMPs. In this section, we devote our attention to the circuit aspects, showing the validity of the approach by means of thorough circuit simulations. We evaluate performance in terms of total available contention-free bandwidth.

In 2010, Sanchez et. al. compared various network topologies of interconnection networks in terms of latency, throughput, and energy dissipation Sanchez et al. (2010). The authors report that for a 64-core CMP, the total area utilization is lowest in case of the mesh topology. The flattened butterfly was shown to consume the largest area (by a factor of $\sim 3\times$). In terms of power consumption, the flattened butterfly (the topology with the largest occupied area) consumes only slightly more power than the mesh due to the higher leakage of the extra links in the butterfly network. On the other hand, the fat tree consumes the most power because of the large number of high-radix router hops and link stages that a flit traverses, on average. Based on these observations, it was concluded that the 2D mesh is best NoC topology overall. As a consequence, the results of our work are compared with the 2D mesh results shown in (Sanchez et al. 2010). Other popular mesh based NoCs are reported in (Bjerregaard 2005; Tran et al. 2010).

In this work, we present circuit design results for a fast, low-latency, source-synchronous ring-based NoC architecture. Inter-processor communication is achieved with minimal latency with the use of extremely fast, intersecting source-synchronous data rings, which traverse the CMP in both the horizontal and vertical directions. Our approach is multi-synchronous since the interconnection network operates on a different clock domain than the PEs. The router complexity as well as the link lengths determines the frequency of operation of the network. We have simulated the routers and links in HSPICE (Inc Meta-Software), with link parasitics extracted from (Raphael Interconnect Analysis Tool: User's Guide). A 4×4 section of the NoC is simulated, to validate routing functionality, as well as to provide accurate circuit-level delay, area and power estimates.

3.1.3 *Our Approach*

In this subsection, we first provide an overview of our source-synchronous ring-based NoC design. Next we discuss our processor modeling assumptions, followed by a discussion of the key logic blocks of our design.

3.1.3.1 Overview

A single ring of our ring-based NoC is shown in Fig. 3.1. Each ring is flattened, and consists of k bits of address, d bits of data and 1 valid bit. These wires are driven source-synchronously, along with a high speed resonant standing-wave clock. The clock operates at 14 GHz, and the transmission rate of the address, data and valid bits is also 14 GHz. The PEs of the CMP connect to the ring at discrete locations through Insertion-Extraction Stations (IESs) which contains two FIFOs as shown in Fig. 3.1. PEs are assumed to operate at 2 GHz. Note that the ring-based NoC operates at a

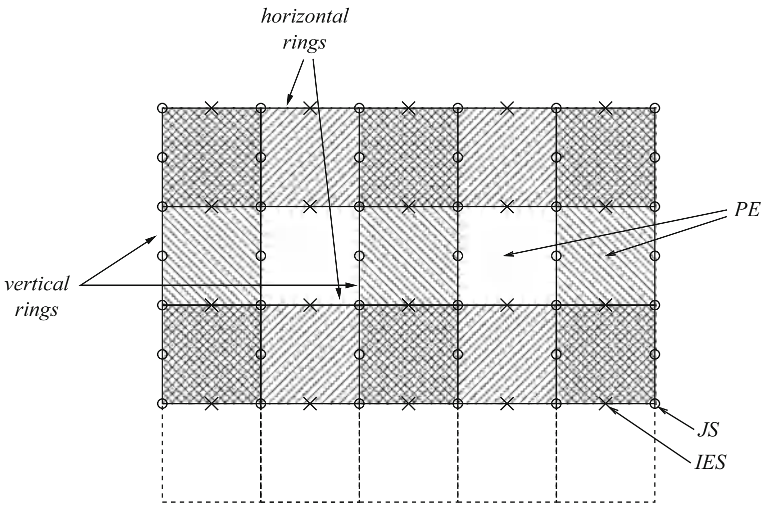


Fig. 3.2 Ring-based NoC architecture

significantly faster clock speed than the cores (or PEs). Because of the extreme high speed of the ring, the latency and transfer characteristics of our ring-based NoC are extremely good.

Figure 3.2 illustrates a section of the CMP, with 2 horizontal rings and 3 vertical rings. The vertical and horizontal rings are shaded with different diagonal patterns, to distinguish them. Regions where the vertical and horizontal rings overlap are shaded using a pattern which consists of the overlay of the horizontal and vertical patterns. Each IES (marked with a $\times$ symbol) services a single PE. Also, a Junction Station (JS) (marked with a $\circ$ symbol) is located at each location where the horizontal and vertical rings intersect. Since there are a total of 20 IES's in Fig. 3.2, therefore 20 PEs are serviced in the NoC fragment shown in this figure. Each PE is shown with a dotted outline. Our source-synchronous rings are unidirectional. Without loss of generality, we assume that each ring transmits data in a counter-clockwise manner. The distance between two adjacent IES's or JS's in the ring is fixed. A ring based NoC is not natively fault tolerant. This can be fixed by using bidirectional rings.

In the following subsections, we discuss our processor modeling assumptions, followed by a discussion of the key components of our design (resonant clock, asynchronous FIFO design, IES and JS design). All plots, tables and figures in this work are generated for a 22 nm PTM (2013) fabrication process.

3.1.3.2 Processor Modeling Assumptions

The most important determinant of the speed of our source-synchronous ring-based NoC is the link length. A long link will force the ring-based NoC to operate slower. This section discusses our methodology to determine the length of each link. Based

Table 3.1 Area and power projections for the cores based on the Sun Niagara2 and Intel Atom Designs. Sanchez et al. (2010)

	Core Area (mm^2)	Core TDP (mW)
Niagra2	1.418	0.483
Atom	1.607	0.220

on Fig. 3.2, we notice that each side of the PE corresponds to two links. This is because JS's and IES's alternate along each horizontal section of any ring. Therefore, assuming a square PE, the length of each link is half the side of the PE.

In Sanchez et al. (2010), the authors studied architectural implications of interconnect design for CMPs with up to 128 cores. The authors approximate the core area and power by scaling down two existing core designs: the Sun Niagara 2 (Nawathe 2007) and the Intel Atom (Gerosa et al. 2009) to a 32 nm process. Since our process is a 22 nm process, we further scale their numbers. Table 3.1 shows the area and power for a 22 nm implementation of the Niagara 2 and the Atom processors. Power numbers in this table are scaled from those of (Sanchez et al. 2010) by multiplying their numbers by the ratio of the square of the saturation I_{ds} at 22 nm and 32 nm respectively, from 22 nm and 32 nm PTM (2013) processes. Just like (Sanchez et al. 2010), we take our PE area to be the average of the two areas shown in Table 3.1. Assuming a square die, this means that each link is 615 μm long.

3.1.3.3 Asynchronous FIFO

Asynchronous FIFOs are used to reliably pass data from one clock domain to another clock domain. Our IES implementation utilizes two asynchronous FIFOs (an *Infifo* and an *Outfifo*). We implemented the design of these FIFOs as suggested in (Cummings and Alfke 2002). In the following discussion, signals with a "*R*_" prefix refer to ring signals, while signals with a "*P*_" prefix refer to PE signals. We describe the *Infifo* in this section. The *Outfifo* is identical, except that "*R*_" signals are replaced with "*P*_" signals and vice versa.

- Block Diagram of *Infifo*: The block diagram of the *Infifo* is shown in Fig. 3.3. The asynchronous FIFO read (*P_Read_en*) and write (*R_Write_en*) enable signals are clocked by independent read (*Pclk*) and write (*Rclk*) clocks respectively. *Din* contains the input data to be written, *Dout* contains the output data. The *Reset* signal is used to initialize the FIFO. Two status flags denote whether the FIFO is empty (*Empty_infifo*) of full (*Full_infifo*). Additionally, the read pointer *P_read_ptr* and the write pointer *R_write_ptr* are outputs of the FIFO. We next discuss how the Write/Read operations are done, followed by how the FIFO full/empty conditions are detected.
- *Infifo* Write and Read: The write pointer (*R_write_ptr*) always points to the next word to be written and the read pointer (*P_read_ptr*) always points to the current FIFO word to be read. Figure 3.4 illustrates the core of the FIFO (the block for the FIFO core is also shown in Fig. 3.3). Writing is done by converting *R_write_ptr* to a one-hot signal, and ANDing it with *R_write_en*. A single FIFO entry

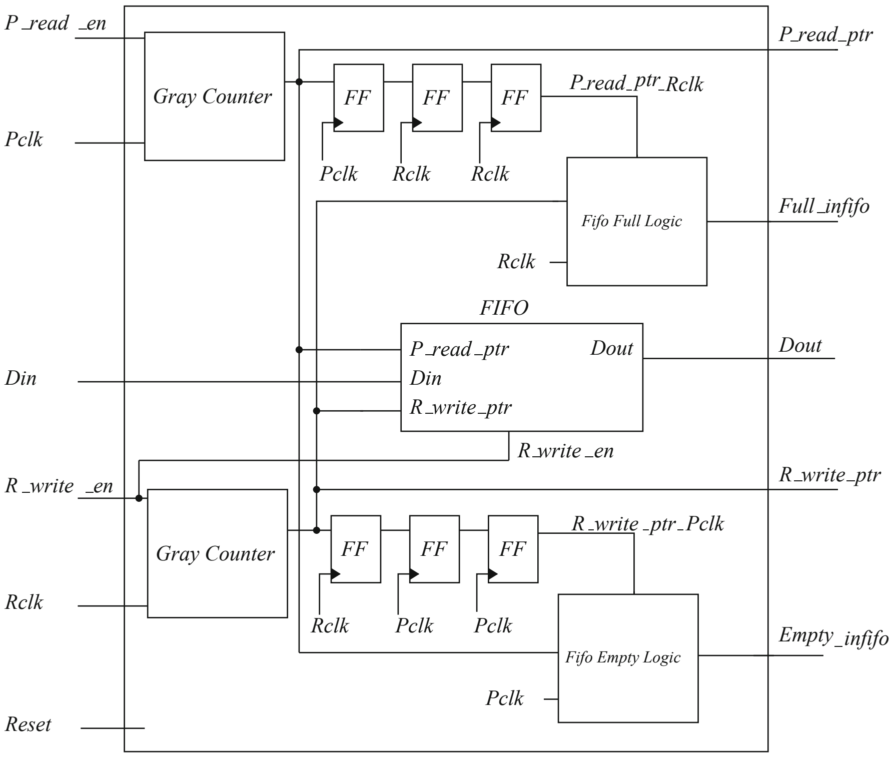


Fig. 3.3 Asynchronous FIFO (*Infifo*) block diagram

(the one that is selected) is therefore written. All other unselected entries are re-written into the FIFO through the '0' input of their corresponding MUXes.
For a read operation, the *P_read_ptr* appropriately selects a FIFO entry, which is driven out on the *Dout* output of the FIFO on *Pclk* unless the FIFO is empty.

- *Infifo* Full and Empty Detection: A FIFO full condition occurs when the write pointer (*R_write_ptr*) catches up to the synchronized and sampled version of the read pointer (*P_read_ptr*).
  Before checking the full condition, the *P_read_ptr* is synchronized to the *Rclk* domain to generate *P_read_ptr_Rclk*. This value is compared with the *R_write_ptr* and when the two are equal, *Full_Infifo* is asserted.
  FIFO empty condition happens when the read pointer (*P_read_ptr*) catches up with the synchronized and sampled version of the write pointer (*R_write_ptr*). Before checking the empty condition, the *R_write_ptr* is synchronized to the *Pclk* domain, to generate *R_write_ptr_Pclk*. This value is compared with the *P_read_ptr* and when the two are equal, *Empty_Infifo* is asserted.
  Synchronization is performed using a 2 flip-flop synchronizers (Dally and Poulton 1998) as shown in Fig. 3.3. In order to achieve a higher MTBF, we can use more

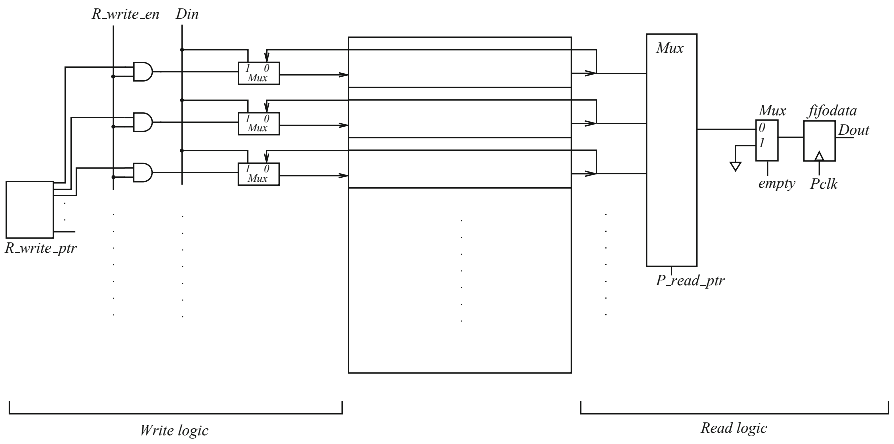


Fig. 3.4 FIFO core circuitry

flip-flops in the synchronizer. Another way of implementing the synchronizer is outlined in Chelcea and Nowick (2000).

For a FIFO depth of n, we need $logn + 1$ bits for the read and write pointers. One extra bit is required to distinguish between full and empty condition. When the write pointer increments past the maximum FIFO address, it will increment the unused MSB while setting the rest of the bits back to zero. The same is done with the read pointer. If the MSBs of the two pointers are different, it means that the write pointer has wrapped around one more time that the read pointer. If the MSBs of the two pointers are the same, it means that both pointers have wrapped the same number of times. This allows us to implement both the empty and the full condition and requires only XOR and XNOR gate for the logic.

Assuming there are n FIFO entries, the empty and full conditions are computed as:

$$Empty\_infifo = (R\_write\_ptr\_Pclk[3:0] == P\_read\_ptr[3:0])$$
$$Full\_infifo = ([R\_write\_ptr[3], R\_write\_ptr[2:0]] == P\_read\_ptr\_Rclk[3:0])$$

Gray code counters are used to keep track of the read and write pointers, as shown in Fig. 3.3. Gray codes only allow one bit to change for each clock transition, eliminating the problem associated with trying to synchronize multiple changing signals on the same clock edge. The XNOR gate for the gray counters were implemented using pass-gates.

Dynamic flip-flops were used as these had to operate at a very high frequency. For the FIFOs of PE_i, *Rclk* and *Pclk* were taken to be 14 GHz and 2 GHz respectively. We have verified 100 % correct functional as well as at-speed operation of the FIFOs.

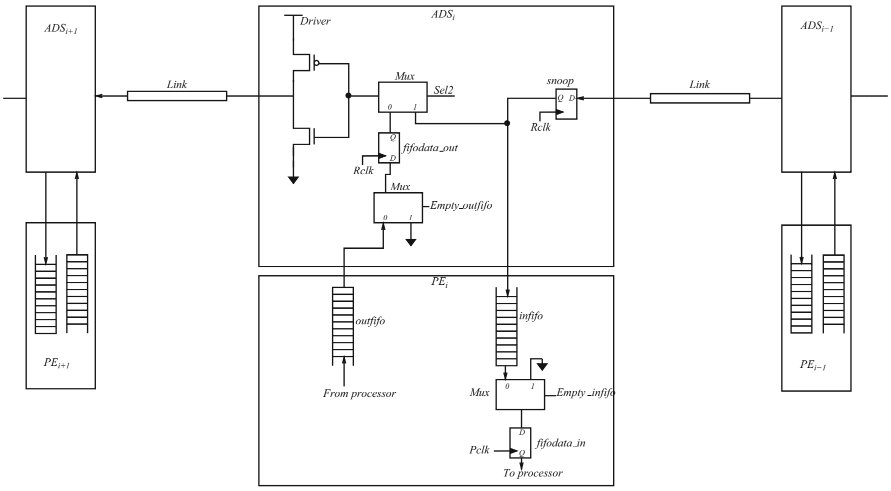


Fig. 3.5 Insertion-Extraction station

3.1.3.4 Insertion-Extraction Station

Each PE communicates with one ring in the ring-based NoC. This communication is achieved using an Insertion-Extraction Station (IES). An IES consists of two FIFOs, and the related logic to insert and extract data into and from the FIFOs. Figure 3.5 shows a section of the data ring in detail. This figure shows three IES's and PE's (indexed $i - 1$, i and $i + 1$). The i^{th} IES and PE are expanded in the center of the figure. Each IES can perform one of three operations – it can *insert* data into the ring, or *extract* data from the ring, or simply *repeat* the data and pass it along. The IES communicates with the PE through an inbound asynchronous FIFO (*Infifo*), and with the ring through an outbound (*Outfifo*) asynchronous FIFO. We next discuss the three operations of the IES, using Fig. 3.5 as a guide.

Insert Operation: This operation requires a read operation from the *Outfifo*. From the left portion of the i^{th} IES of Fig. 3.5, we see that the FIFO output (which is the data stored in the entry pointed to by *R_read_pointer*) is captured in the *fifodata_out* register on the rising edge of *Rclk* whenever the *Outfifo* is not empty. If *valid* is low, indicating that the link is not carrying valid data in this clock cycle, then this data is driven out over the link. Hence, we only drive out data from the *Outfifo* when the link is not carrying valid data.

Extract Operation: This operation requires a write operation in the *Infifo*. From the right part of the i^{th} IES of Fig. 3.5, we note that the link data is captured at each cycle of *Rclk* into a *snoop* register. If *R_write_en* is true, then, on the rising edge of *Rclk*, data is entered into the *Infifo*, into the location pointed at by *R_write_pointer* (as shown in Fig. 3.4).

$R\_write\_en = (valid) \cdot (addrmatch \cdot \overline{Full\_Infifo})$

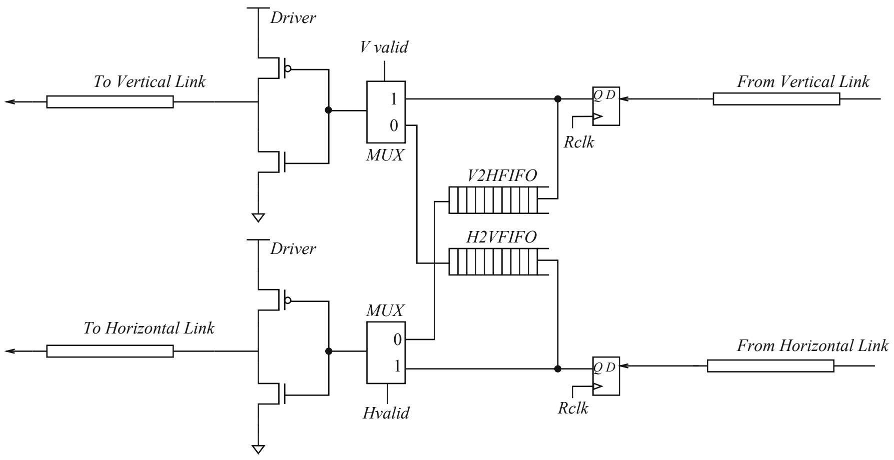


Fig. 3.6 Junction station

In other words, data is extracted if the packet is valid (first term above) and if the address matches, and the input FIFO is not full (second term).

If there is data in the *Infifo* (i.e. *empty_infifo* is false), then data is captured into the register *fifodata_in* synchronous with $Pclk$, for consumption by the PE, as shown in the bottom right of the PE_i block in Fig. 3.5.

Repeat Operation Based on Fig. 3.5, we note that data or address information repeated iff the packet satisfies $(valid) \cdot \overline{((addrmatch \cdot \overline{Full\_Infifo}))}$. In other words, data or address information is forwarded if the packet is valid, and does not match the IES address, or if the *Infifo* is full. In case valid is not true, address information is sent out from the *Outfifo* on to the ring, provided the *Outfifo* is not empty.

Valid information is forwarded when the disjunction of $(valid) \cdot \overline{((addrmatch \cdot \overline{Full\_Infifo}))}$ and $\overline{(valid)} \cdot \overline{Empty\_outfifo}$ is true.

3.1.3.5 Junction Station

A Junction Station (JS) operates in a same manner as an IES, other than the fact that it does not have any PE attached to it. Figure 3.6 depicts a JS present at the intersection of two rings.

A JS consists of 2 asynchronous FIFOs. The H2V FIFO is responsible for collecting data from the horizontal ring and transferring it to the vertical ring. The V2H FIFO is responsible for collecting data from the vertical ring and transferring it to the horizontal ring. Both clocks in both these FIFOs operate at 14 GHz (but are mesochronous). Instead of asynchronous FIFOs, mesochronous FIFOs may be used as well Chelcea and Nowick (2000).

Each JS station can perform the following two operations – it can *transfer* data from one ring to the other or simply *repeat* the data and pass it along in the same ring. WLOG, we discuss the horizontal to vertical (H-V) transfer and horizontal to horizontal (H-H) repeat operations. The vertical to horizontal (V-H) transfer and vertical to vertical (V-V) repeat operations are similar.

Transfer Operation: Consider the input from the horizontal ring on right bottom of Fig. 3.6. If *Hvalid* is true (indicating that there is a valid packet in the horizontal ring), and the data is destined for the vertical ring, and the H2V FIFO is not full, the packet is inserted into the H2V FIFO. In this case, *Hvalid* is driven low on the horizontal ring as the packet is being consumed in the H2V FIFO. In subsequent cycles, when *Vvalid* is low (indicating that the vertical ring is free) and the H2V FIFO is not empty, a packet is driven from the H2V FIFO through the vertical link (top left). In this case, *Vvalid* is driven high on the vertical ring.

Repeat Operation: Repeat operation relays the data in the same horizontal ring. This happens when the destination PE is present in the same horizontal ring where the packet is traveling, or if the H2V FIFO is full. Consider the input from the horizontal ring on right bottom of Fig. 3.6. If *Hvalid* is true (indicating that there is a valid packet in the horizontal ring), and the data is destined for the horizontal ring, then data is driven out through the horizontal link (bottom left) in the same cycle.

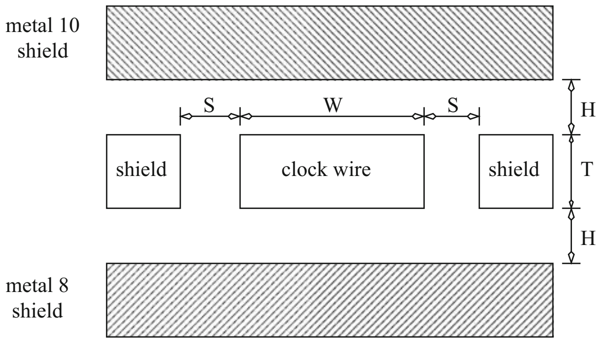
3.1.4 Experimental Results

We implemented our design in the 22 nm PTM (2013) technology, with $VDD = 0.8\ V$. All simulations were conducted in HSPICE Inc Meta-Software (Inc Meta-Software). RLC parasitics for all the wires were extracted using Raphael (Raphael Interconnect Analysis Tool: User's Guide). Next, we provide our circuit simulation results followed by the network performance projection of our ring-based NoC on a 16 × 16 CMP.

3.1.4.1 Circuit Validation

For at-speed validation purposes, we simulated a 4×4 tile where each PE tile was assumed to be 1.229 mm × 1.229 mm. Hence the length of each link was taken to be 0.615 mm. Since the JS were 1.229 mm apart in the vertical ring, we had to a put a repeater between two JS's in the vertical ring. The address field (k) has 4 bits (to address 16 PEs) and the data width (d) was taken to be $9B$, $18B$ and $36B$. There are 2 horizontal and 2 vertical rings each of length 12.29 mm. Our circuit simulation successfully routed a packet which traversed several IES's and JS's before reaching its destination. We have calculated the power assuming that 26 % of the links are active. The benchmark used in Sanchez et al. 2010 (2010) for power consumption had a link activity of 26 %, hence this choice. The areas reported are standard cell

Fig. 3.7 Link cross section



areas. The ring clock ($Rclk$) was assumed to be 14 GHz (since the IES and JS operate at this frequency with a nominal guard-band) while the PE clock ($Pclk$) was 2 GHz.

The wire dimensions for the links between two stations are shown in Fig. 3.7. Wires are implemented in Metal 9. In both the IES and the JS, a driver of size 30× of a minimum sized inverter was used to drive a single bit link with a stage ratio of 3×. The delay of the driver was 13.86 ps, while the delay of the mux was 16.08 ps. As the length of the link being driven is very small, the wire delay was found to be negligible. The clock-to-Q delay and the setup time adds to sum of the delay of the driver and the mux to determine the clock frequency. We found the entire 4×4 NoC operates with 100 % functional correctness at 14 GHz. PVT or On-Chip variation (IR drop, local hot spots etc) can reduce the operating frequency of our ring-based NoC. Currently we address this issue by adding a nominal guard-band, which results in a operating speed of 14 GHz for the NoC. All IES operations (*Insert*, *Extract* and *Repeat*), and all JS operations (*Transfer* and *Repeat*) were simulated to complete correctly at 14 GHz.

We have verified the correct operation of the *Outfifo* and *Infifo* for *Rclk* of 14 GHz and *Pclk* of 2 GHz, while randomly varying the phase between the *Rclk* and the *Pclk*. In addition, we also verified the correct operation of the FIFOs in the JS where both the clocks were 14 GHz.

The ring which distributes the 14 GHz clock at the IES and JS was assumed to be laid out on Metal 8, with wires of width $1\mu m$, spacing $1\mu m$ and height $0.9\mu m$. For a ring of length 1.229 mm between two stations and an inverter of size $W_p = 2.5\mu m$ and $W_p/W_n = 2$, the oscillation frequency of 14 GHz was obtained. In order to achieve a ring of length 12.29 mm, we have implemented a $11 \cdot \lambda/2$ ring. The power consumed by a single $\lambda/2$ ring was 2.67 mW.

3.1.4.2 Performance Projections

- **Power and Area** In order to compare our results with a 16×16 mesh topology in Sanchez et al. (2010), we have scaled the area and power appropriately to account for the additional IES's, JS's and different k values. The area and power comparison is given in Table 3.2. Ring results are expressed as a ratio of their

Table 3.2 Power and area comparison with Mesh Topology in. Sanchez et al. (2010)

Flit Width	Mesh Power (mW)			Mesh Area (mm^2)			Ring Power			Ring Area		
	Router	Link	Total	Logic	Wire	Total	Router	Link	Total	Logic	Wire	Total
d=9B	0.70	5.81	6.51	3.400	8.976	12.376	2.24	0.36	0.57	0.66	1.12	1
d=18B	1.41	7.74	9.15	6.616	17.952	24.568	1.4	0.46	0.6	0.64	1.06	0.95
d=36B	2.46	11.97	14.43	14.936	35.904	50.84	1.17	0.54	0.65	0.55	1.03	0.89

Table 3.3 Comparison of network configurations with Mesh in. Balfour and Dally (2006)

Topology	H	$t_r(cycles)$	B_C	$B_B(bits)$	$\frac{B_B}{T_{clk}^{NoC}}(Gbits/sec)$	$T_c(cycles)$	$T_0(cycles)$
Mesh (16×16)	12.25	2	32	4608	9.216	10.6	35.10
Fast Ring (16×16)	24.25	3/14	16	2304	32.256	0.88	6.08
Mesh (n×n)	$\frac{3n+1}{4}$	2	2n	288n	576n	$n \times 0.6625$	$2.1625 \times n + 0.5$
Fast Ring (n×n)	$\frac{6n+1}{4}$	3/14	n	144n	2016n	$\frac{3n+1}{56}$	$\frac{21n+4}{56}$

corresponding values compared to the mesh. Area and power values for the ring-based NoC are normalized to the corresponding values for the mesh. For a flit width of 36 bytes, we achieve a 35 % gain in total power and a 11 % gain in total area. The reason why our approach yields a better area is that we utilize unidirectional rings, which reduces the number of ports per intersection of horizontal and vertical links.

- **Latency and Bandwidth** In Balfour and Dally (2006), the authors have presented detailed area and energy models for on-chip interconnection networks. Table 3.3 compares the network performance reported in Balfour and Dally (2006), with our approach, for a 16×16 NoC as well as an $n \times n$ NoC.
  We report the average contention-free latency (T_0) incurred by a flit from source s to destination d which includes: (1) the average hop count (H) from s to d, (2) router traversal latency (t_r) and (3) the average channel traversal latency (T_c), which is the latency induced by repeaters in long links.
  $T_0(s, d) = H(s, d) * t_r + T_c(s, d)$
  In order to calculate the average hop count for an $n \times n$ NoC, we first select a source with co-ordinates (i, j), where $(0 \leq i \leq n - 1$ and $0 \leq j \leq n - 1)$. Next, we calculate the average distance of all possible destinations (n^2 in all). Then, we perform the average over all possible values of i and j (for a total of n^2 sources). The obtained average is reported as the average hop count for a given NoC topology. The router latency (t_r) for the mesh as reported in Balfour and Dally (2006) is 2 PE clock cycles. For our ring-based NoC, we use a deflection based routing scheme in which a flit follows the shortest path from a source to a destination, switching between rings (making turns) at JS'. However, if a flit cannot make a turn at a JS due to congestion, it continues to travel in the same

direction. The JS which acts as a router consumes only 1 cycle if the data stays in the same ring, else it takes a minimum of 3 cycles if it has to switch rings. Since the horizontal and vertical rings communicate through asynchronous FIFOs, we incur a 2 cycle penalty for synchronization. The IES always consumes only 1 cycle to repeat or drop the data. Hence the average router latency (t_r) is 1.5 cycles since it will encounter the IES (with 1 cycle latency) and a JS (with expected penalty of 2 cycles) equally often. Since our NoC clock is 7 $\times$ faster than the PE clock, we further divide by 7 to arrive at a t_r value of 3/14 PE clock cycles. B_C corresponds to bisection channel count and B_B corresponds to bisection bandwidth (in bits), as defined in Balfour and Dally (2006). The flit width is taken to be 144 bits for both the topologies. Recall that the mesh has bidirectional links, while the ring-based NoC does not. $\frac{B_B}{T_{clk}^{NoC}}$ defines the aggregate bandwidth (expressed in Gbits/sec) available. T_c for the mesh as reported in Balfour and Dally (2006) is 10.6 PE clock cycles for a 16 $\times$ 16 NoC. Note that for our scheme, we do not introduce any link repeaters in the horizontal ring. However, in the vertical direction, we insert one repeater between two JS'. Hence, T_c is zero by design for horizontal rings and 1 NoC clock cycle for vertical rings. Similar to the way we calculate the average hop count, we consider every possible source and destination and find the average delay in terms of NoC cycles introduced by the repeaters. Since our ring-based NoC operates 7$\times$ faster, we further divide this number by 7.

In Balfour and Dally (2006), the clock frequency for the NoC was assumed to be same (2 GHz) as the PEs. Our source synchronous ring based NoC runs 7$\times$ faster than the PEs. We report the results in Table 3.3 in terms of PE clock cycles. For a 16 $\times$ 16 topology, we improve the average contention free latency by 5.8$\times$. This is achieved because our NoC clock is 7$\times$ faster than the PE clock. For the same reason, available aggregate bandwidth is 3.5$\times$ compared to a regular mesh. Note that our rings are unidirectional and hence we have half the number of links between two PEs compared to a bidirectional mesh as reported in Balfour and Dally (2006). Also, our projections on a general $n \times n$ NoC (last 2 rows) suggests that for large n, we achieve a significantly lower contention free latency (5.76$\times$ lower) and higher bisection bandwidth (3.5$\times$ higher) for ring-based NoC compared to a mesh.

3.1.5 Conclusion

Traditionally, Network-on-Chip (NoC) architectures are based on mesh interconnection structures. We propose a ring based NoC architecture which is based on a source synchronous data transfer model over a ring. The source synchronous ring is clocked by a resonant clock which operates significantly faster than the individual processors that are served by the ring. This allows us to significantly reduce the area devoted to the NoC logic and wiring. We have validated the design using a 22 nm predictive process. Results indicate that our approach achieves 3.5$\times$ better bandwidth, 5.8$\times$ better contention free latency with lower area and power than a 2D mesh.

3.2 Architectural Simulations of a Source Synchronous Ring-based NoC

In the previous section, we only considered the circuit design aspects of the ring-based NoC, without performing architectural simulations. In this section, we perform architectural simulations of the ring based NoC. Moreover, in the previous section, we assumed all the SWO clock rings to be mesochronous, and hence requiring asynchronous FIFOs at the junction of horizontal and vertical rings. In the following part of our work, we propose a design where all the ring clocks are synchronous. The above is achieved by using a SWO-based “comb" clock distribution topology as discussed in Sect. 2.2 or a SWO-based tiled clock distribution topology as discussed Sect. 2.4. The above modification improves the NoC performance by allowing us to avoid the synchronization latency when switching rings at the intersection of horizontal and vertical rings. Also, the PE clocks and the NoC clock are multi-synchronous.

3.2.1 Introduction

The mesh interconnection network has been preferred by the Network-on-Chip (NoC) research community due to its simple implementation, high bandwidth and overall scalability. Most of the existing NoCs operate at the same or lower clock speed as the processing elements (PEs), hence slowing down the communication system. In the previous section, we introduced a new source-synchronous ring based NoC architecture, which runs significantly faster than the PEs and offers a high bandwidth and low contention free latency. We validated the NoC design at the circuit level, using a 22 nm predictive process technology, and showed that the proposed architecture can operate at a clock speed of 14 GHz, with the clock derived from a standing wave resonant oscillator. In this section, we explore the architectural aspects of the fast ring based NoC. Our architectural simulations show that the ring-based NoC design as described in Sect. 3.1 suffers from deadlock. In this work, we avert deadlock by using link ordering and virtual channels. This requires a redesign of the routers used in Sect. 3.1, as described in this sequel.

We showed the block diagram of a single ring for the ring based NoC in Fig. 3.1. The ring data path comprises of three fields of information – d bit *data*, k bit *address*, and 1 *valid* bit. A high speed resonant clock signal *Rclk* runs parallel to the ring signals mentioned above, as shown in Fig. 3.1. The data, address and valid signals are source synchronous with the *Rclk* signal. Each ring operates significantly faster (14 GHz) than the PEs (2 GHz). We used a deflection based routing scheme in which a flit follows the shortest path from a source to a destination, switching between rings (making turns) at JS’. However, if a flit cannot make a turn at a JS due to congestion, it continues to travel in the same direction. The above routing protocol results in a deadlock due to the cyclic dependency of the resources on various paths in the network. Hence, we present modifications to the IES' and JS' to provide deadlock free routing. We also perform HSPICE based circuit simulations of these modifications. Next, we present architectural simulations of the ring-based NoC on synthetic and real traffic patterns, showing significant improvements in latency and throughput compared to a state of the art mesh-based NoC.

The rest of the section is organized as follows: Sect. 3.2.2 describes previous approaches in NoC design. Section 3.2.3 presents our design of a deadlock-free routing of our ring-based NoC, while Sect. 3.2.4 describes the circuit and architectural experiments which we performed to validate our approach. We conclude in Sect. 3.2.5.

3.2.2 Previous Work

The performance of an NoC depends on the combination of several characteristics such as topology, routing algorithm, flow control mechanism and router micro-architecture. The topology determines the way in which the PEs are connected, affecting the bandwidth and latency of a network. The routing mechanism determines the path taken by a packet from a source to a destination, and can be either deterministic or adaptive. A primary requirement for any routing algorithm is that it must be deadlock-free. Flow control refers to the policy of network resource allocation as packets travel from source to destination. Popular approaches include (a) deflection (bufferless) (b) virtual channel (buffered). The complexity of the router micro-architecture and the link traversal delays determine the speed of operation of the entire NoC.

As mentioned in Sect. 3.1.2, Sanchez et. al. (2010) compared the mesh, flattened butterfly and fat tree topologies in terms of latency, throughput, and energy consumption. Their results indicate that for a 64-core CMP, the total area utilization is lowest for a mesh. The flattened butterfly was shown to consume the largest area (by a factor of $\sim 3\times$), consuming slightly more power than the mesh due to the higher leakage of the extra links of the butterfly. On the other hand, the fat tree, with a large number of high radix routers and link stages consumes the most power. Based on these observations, the authors concluded that the mesh is best NoC topology overall. In Michelogiannakis (2010), the authors compare following two flow control mechanisms of the mesh topology: (a) bufferless with deflecting flow control and (b) virtual channel (VC) buffered flow control. Results indicate that for a 64-core CMP, VC flow control provides a 12 % smaller average latency and 21 % higher sustained injection rate compared to deflection-based flow control. As a consequence, we compare our architectural results with a state of the art mesh with virtual channel buffered flow control.

In all the above implementations, design decisions are made based on the fact that the NoC runs at the same or lower frequency as the PEs. In the Sect. 3.1, we presented a fast source synchronous ring based NoC architecture which runs significantly faster ($7\times$) than the PEs. Data is driven in a source-synchronous manner along with a high speed resonant clock. This allows us to achieve significantly higher bisection bandwidth with narrower links (yielding a lower area and power for the same bisection bandwidth). The significantly lower latencies allow the proposed NoC architecture to scale more elegantly for larger CMPs. We evaluated the performance in terms of total available contention free bandwidth. In this section, our key focus is on the architectural aspects of a ring-based NoC. We propose modifications to provide deadlock free routing, using link ordering and virtual channels. We implement a virtual channel based buffered flow control in contrast to baseline deflection-based flow control as proposed in Sect. 3.1. We also propose and simulate (in HSPICE) a modified router architecture to support our new deadlock-free design. We perform architectural simulations and compare performance of the ring-based NoC with a state of the art mesh-based NoC, using synthetic and real traffic.

3.2.3 Our Approach

In this section, we first show that the baseline ring-based NoC of Sect. 3.1 suffers from deadlock. Next, we propose a modified NoC architecture to avoid deadlock by the use link ordering and virtual channels.

3.2.3.1 Presence of Deadlock in Ring-based NoC of Sect. 3.1

We first perform architectural simulations of the ring-based NoC of Sect. 3.1. We use our proposed deflection based routing scheme, in which a flit follows the shortest path from a source to a destination, switching rings (making turns) at JS'. However, if a flit cannot make a turn at a JS due to congestion, it continues to travel in the same direction. Note that the architecture has no support for buffering at the routers and hence the flit has to keep circulating in the ring(s) till it reaches it's destination.

Figure 3.8 presents the total latency (in terms of PE cycles) of the ring-based NoC of as a function of flit injection rate for uniform random traffic. On average, the ring-based NoC of has a $4.3\times$ lower latency compared to the mesh. The mesh can only sustain a maximum injection rate of 38 % compared to the ring-based NoC, which can sustain a 44 % injection rate. However, around an injection rate of 44 %, the number of flits received for the ring-based NoC falls drastically, indicating a deadlock situation. The deadlock occurs due to a cyclic dependency of resources on the various paths in the network. In particular, the channel dependency graph (defined in Sect. 3.2.3.2) is cyclic, resulting in the deadlock.

In the following section, we first devise mechanisms to avoid a deadlock in the ring-based NoC. In practice, we use link ordering and virtual channels, as described in Sect. 3.2.3.2. These modifications require changes in the routers (IES' and JS'), which we describe Sect. 3.2.3.3.

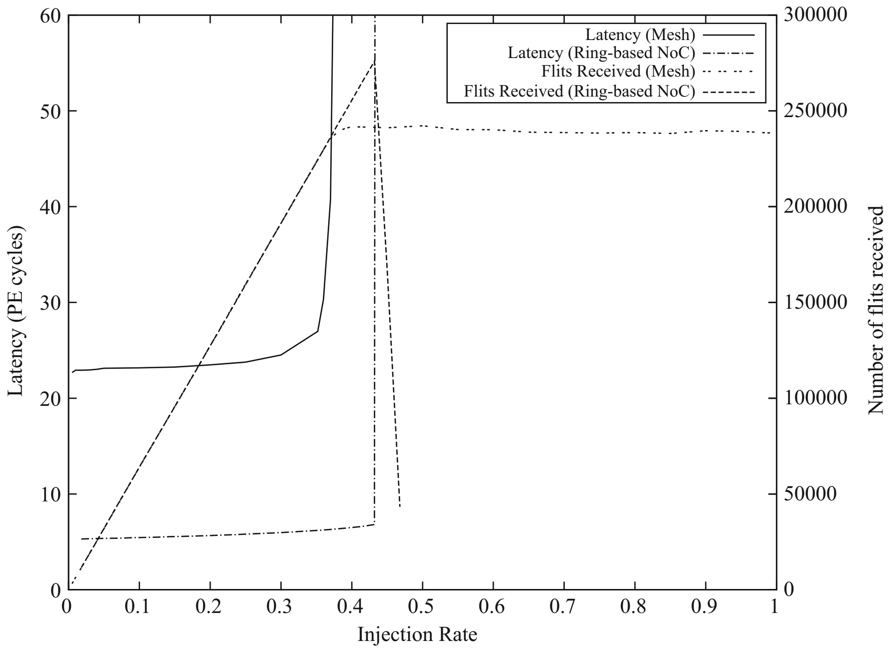
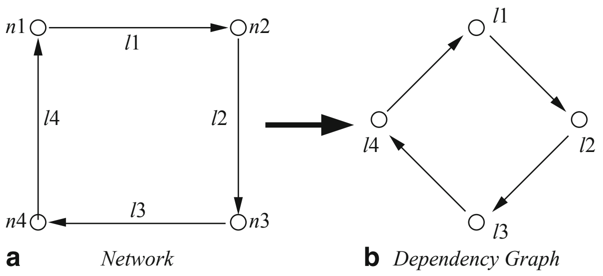


Fig. 3.8 Latency comparison for Uniform Traffic

Fig. 3.9 Channel dependency graph



3.2.3.2 Deadlock Avoidance in a Ring-based NoC

Consider the case of a unidirectional four-node network as shown in Fig. 3.9 a), with nodes N = n_1, n_2, n_3, n_4 and links L = l_1, l_2, l_3, l_4. Let us assume that each node has a queue of unit length and also the queue of each node is filled with the message destined for the diametrically opposite node (for example, node n_1 has a message destined for node n_3, node n_2 has a message destined for node n_4, and so on). Clearly, no message can advance in this situation and this will cause deadlock.

The corresponding channel dependency graph D (Fig. 3.9 b) is constructed where the vertices of D are channels, and edges are pairs of channels that are adjacent. In Dally and Seitz (1987), the authors prove that the necessary and sufficient condition for deadlock-free routing is the absence of cycles in the channel dependency graph. Clearly D in Fig. 3.9 has a cycle and hence suffers from deadlock. The authors propose to remove cycles of the dependency graph by splitting a physical channel into groups of virtual channels. Every virtual channel shares a physical channel but needs to implement its own queue. We observe that the number of virtual channels required grows exponentially with the number of cycles present in D. The ring-based NoC proposed in Sect. 3.1 is constructed as a grid of unidirectional rings similar to Fig. 3.9a. The dependency graph of such a ring-based NoC has cycles, and hence exhibits a deadlock. This is confirmed by the simulation results in Fig. 3.8.

In order to avoid the deadlock, we use link ordering Chiu et al. (2002) and virtual channels Dally and Seitz (1987). Consider a $n \times n$ ring-based NoC as shown in Fig. 3.10. Each node corresponds to a JS in this figure. We split a link of width L in the ring-based NoC into two bidirectional links, each of width $L/2$. Now we label each link in the following way. For each row of nodes $(JS_{11}, JS_{12}, \ldots JS_{1n})$ we label eastward links with an increasing index starting from zero. Hence the link JS_{11}-JS_{12} has label 0, JS_{12}-JS_{13} has label 1, etc. Then, we label the westward links with an increasing index starting from n. The same labeling is used for every row of the $n \times n$ ring-based NoC. Similar labeling techniques are used for vertical links as well. Consider the column $(JS_{11}, JS_{21}, \ldots JS_{n1})$. Southward links are labeled in increasing order starting with label $2n$. Therefore the link JS_{11}-JS_{21} has label $2n$, JS_{21}-JS_{31} has label $2n+1$, etc. Then, we label the northward links with an increasing index starting from $3n$. Using this labeling scheme, we induce a deadlock-free routing algorithm between any two JS' according to Chiu et al. (2002). The necessary and sufficient condition for deadlock free routing is that routes use links only in increasing order of labels. Suppose that there exists a total order of the labeling of the links in L. Also let us consider that the minimum index link (with index l_m in this order has a full queue. Every link l_k that l_m feeds has a larger index than l_m and thus does not have a full queue. Hence, no flit in the queue for l_m is blocked, and no deadlock exists.

We next show that our labeling honors this condition, yielding a unique path for any source destination route. Consider a route from source s = (i, j) to (a different) destination d = (i', j') such that $(i \leq i', j \leq j')$. There exists a unique route from s to d, constructed as follows. Since row labels are strictly smaller than the column labels, we first proceed eastward from (i, j) to (i', j) encountering labels $i, i+1, \ldots$ Now we proceed north from (i', j) to (i', j') encountering labels $j+2n, j+2n+1, \ldots$ Note that there exists a unique route for any (s,d) and this route is same as the dimension-ordered route (DOR)-XY for a mesh. This unique route satisfies the necessary and sufficient condition for deadlock freedom.

The labeling of links in Fig. 3.10 guarantees that there exists a deadlock-free path for any route. However, the nodes in Fig. 3.10 are JS', and the first and last hop of any route must travel in the horizontal ring (whose indices are lower than those of a vertical ring). This violates the DOR-XY constraint, creating a cycle in the channel dependency graph. To solve this problem, we borrow the idea of Dally and Seitz (1987) and introduce a separate virtual channel for each JS to the IES on its immediate right. Hence all the flits destined for any PE travel to the JS on its left,

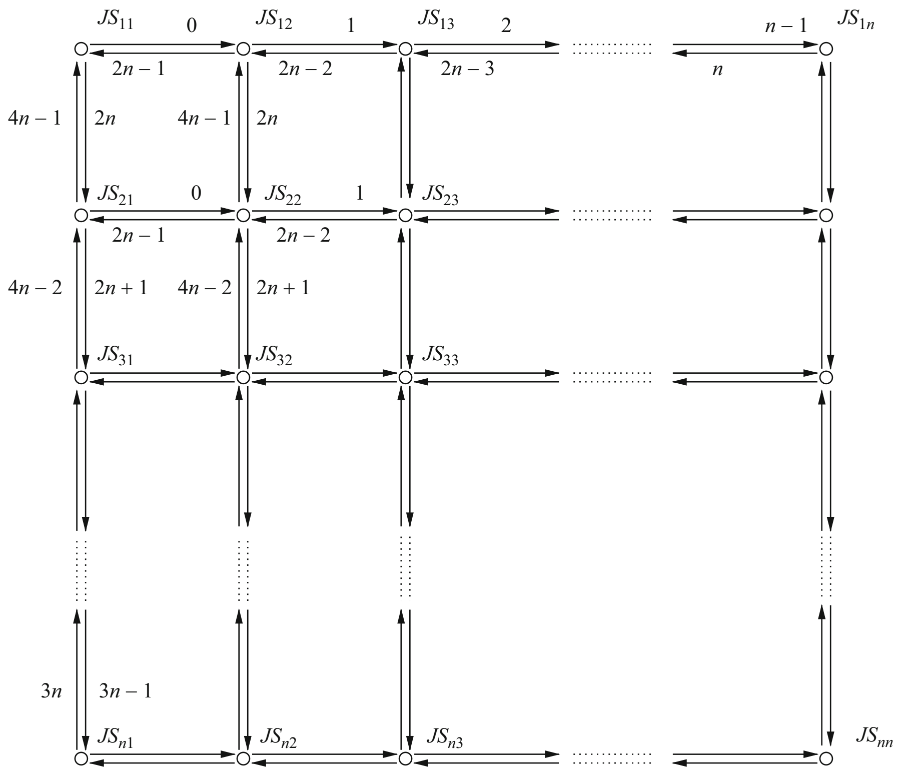


Fig. 3.10 Link ordering for ring-based NoC

and then uses this separate virtual channel to reach the destination, hence breaking the cycle in the channel dependency graph.

In this work, we implement virtual channel based buffered flow control. We assume that each link can support six virtual channels which can be allocated to the three incoming ports of that router. We also implement an input buffer based router micro-architecture. In this design, each input port sends a back-pressure signal to its upstream router indicating the non-availability of input buffers. Output buffering requires more complex arbitration and hence we avoid it in our design. Another drawback of output buffering is that multiple back-pressure signals (one for each output port) needs to be routed to upstream routers.

3.2.3.3 Router Architecture in the Ring-based NoC

The ring-based NoC uses two different types of routers called IES and JS. We split a unidirectional link of width L in the original ring-based NoC into two bidirectional links each of width $L/2$. Hence, the IES needs to be modified in comparison to the baseline design. It has three input and three output ports (including a pair of ingress/egress ports connected to the local PE). A modified JS has a maximum of four input and four output ports. Figure 3.11 shows the major components of a JS, implemented as a 3 stage pipelined virtual-channel router. It has 4 pairs of input and output ports, supporting 6 virtual-channels (VCs) per port. Note that the structure of the IES is similar, except that it only has 3 pairs of input and output ports.

The three pipeline stages of the JS are described below:

- **Buffer Write (BW) + Route Compute (RC)**: A flit arrives at the input port and is registered at the *Buffer_inlink* block which is a register. In the first pipeline stage, the address of the flit is sent to the route computation (*RC*) block to calculate the output port. The flit, along with its output port, is written into one of the free virtual channel buffers pointed by the *VC_slto*. A one bit back-pressure signal (*BP*) is send by each input port to its upstream router, signifying the availability of virtual channel buffers.
- **Output Port Allocation (OPA)**: Individual flits waiting at the input virtual channel buffers (VC_1 - VC_6) arbitrate for access to physical channels at the output port. Arbitration is performed in two stages. The first stage of arbitration is performed between 6 input virtual-channels. This requires a 6 input arbiter for each input-output pair (for a total of 12 arbiters since there are 4 input ports and 3 output ports per input port). There is one less output port since flits do not return to the direction they arrive from. Let us consider the arbiter ($M1$) from input port zero to output port 3 as shown in Fig. 3.11. The *rr_vc* signal chooses one of the 6 virtual-channels. The *rr_vc* is generated with the help of a modulus 6 counter ($C1$). However the *rr_vc* signal is updated with the new counter value only when $sel0$ is true. The signal $sel0$ is a conjunction of two events: the virtual channel buffer pointed at by the new counter value is not empty and the output port value of the flit pointed at by the new counter is 3.
  The second stage of arbitration occurs between the input ports. Once a flit is selected from the input port for a certain output port, the output port performs an arbitration between the three input ports. Let us consider the arbiter ($M2$) at output port 3, whose control input is the *rr_port* signal. The *rr_port* signal is generated with the help of a modulus 3 counter ($C2$). However the *rr_port* signal is updated with the new counter value only when $sel1$ is true. This happens when the port pointed by the new counter value is not empty. Note that the *rr_port* signal is driven to zero when the corresponding back-pressure (*BP*) signal is low, signifying the unavailability of buffers at the downstream router. The flit which passes through both arbiters ($M1$ and $M2$) gets written into the *Buffer_outlink*.
- **Link Traversal (LT)**: Flits from *Buffer_outlink* are driven out through the output link in the third pipeline stage.

3.2.4 Experimental Results

In this section, we first present the circuit level simulation results for the virtual-channel enabled JS/IES. Next we present the architectural simulation results of a 8×8 NoC under different synthetic traffic patterns.

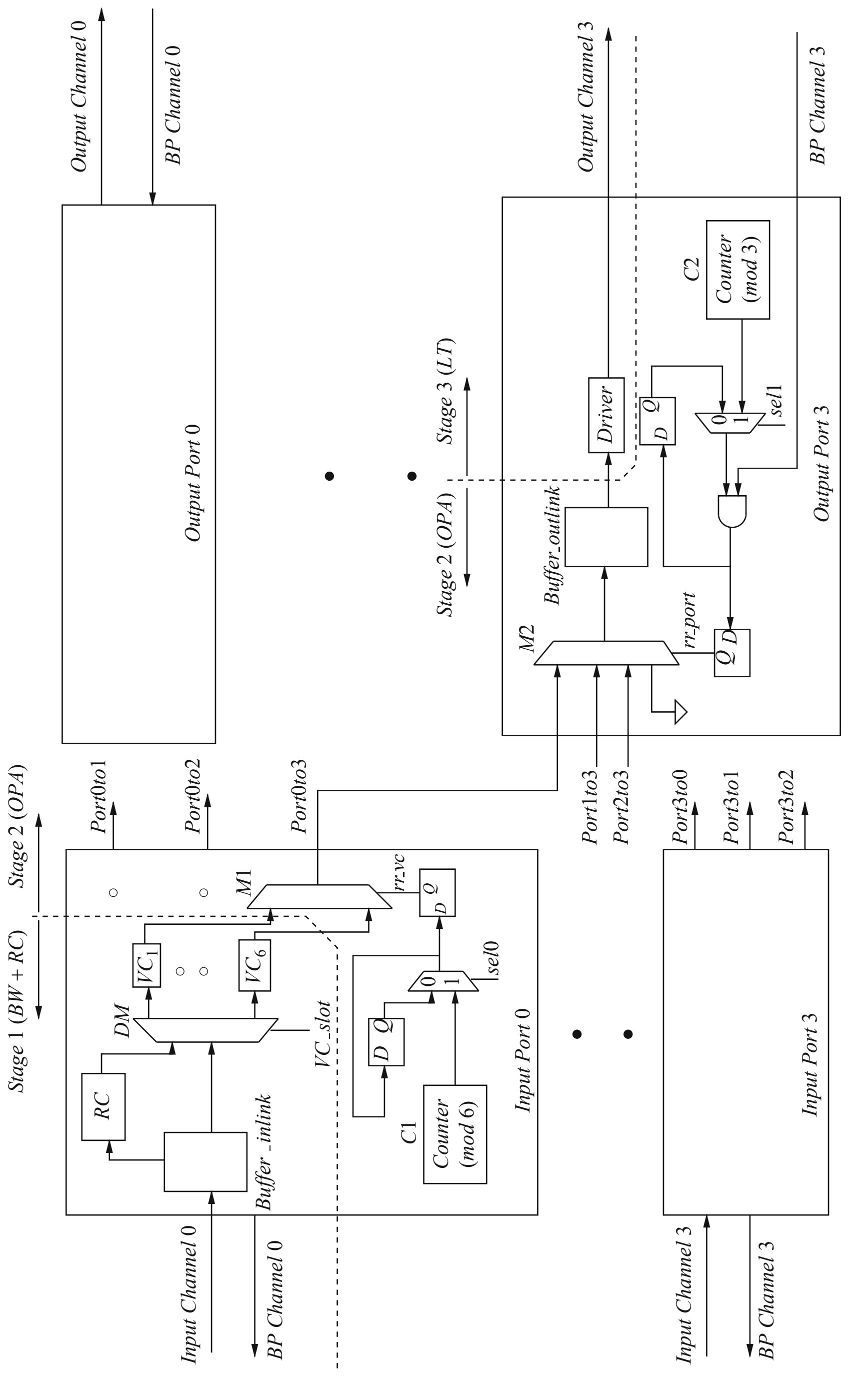


Fig. 3.11 Modified router architecture of a ring-based NoC

3.2.4.1 Virtual Channel Enhanced Routers

We performed circuit simulations to validate our IES/JS router design as described in Sect. 3.2.3.3. Our design was implemented in the 22 nm PTM (2013) technology, with $VDD = 0.8\ V$. All circuit simulations were conducted in HSPICE Inc Meta-Software (Inc Meta-Software). RLC parasitics for all the wires were extracted using Raphael (Raphael Interconnect Analysis Tool: User's Guide). As discussed in Sect. 3.1.3.2, we assume our PEs to be $1.229\ mm \times 1.229\ mm$. Hence the length of each link (which is half of the PE side as shown in Fig. 3.2) was taken to be $0.615\ mm$. Since the JS were $1.229\ mm$ apart in the vertical ring, we had to a place a repeater between two JS' in the vertical direction.

The delay of the first stage pipeline (BW + RC) was found to be 44.2 ps. The delay of the second stage pipeline (OPA) was found to be 42.8 ps. For the last pipeline stage, which is the link traversal (LT), the delay was found to be 41.7 ps. Worst-case cross talk patterns were assumed to occur among any 3 adjacent wires in a link. The clock-to-Q delay and the setup time of the registers are added to each of the above pipeline stage delays to determine the clock frequency. PVT or on-chip variations (IR drop, local hot spots etc) can reduce the operating frequency of the ring-based NoC. We address this issue by adding a nominal guard-band, which results in a operating speed of 14 GHz.

3.2.4.2 Synthetic Traffic Results

We use a modified version of GEM5 (Binkert et al. 2011) for cycle-accurate micro-architectural NoC simulations. We compare the performance of a state of the art mesh and the ring-based NoC by running synthetic traffic (uniform, bit complement and tornado). Let us consider an $n \times n$ NoC. Under uniform traffic, for each source, the destination is selected uniformly and randomly from the remaining $n^2 - 1$ cores. Under tornado traffic, a source (x, y) will have its destination as $(x + n/2 - 1)\%n$, y). Finally, under bit complement traffic, a source (x, y) will have its destination as $(n - x - 1, n - y - 1)$.

We simulate a network with 64 PEs. We assume a clock frequency of 2 GHz for the PEs. The link width in any direction is assumed to be 144 bits for the mesh and 72 bits for the ring-based NoC. We assume a single flit packet of 144 bits for the mesh. Thus each 144 bit flit requires two sub-flits in the ring-based NoC. Currently in our design, we do not have any support for the wormhole routing and hence we route the two sub-flits of a single flit independently. As discussed in Sect. 1.3.2, this flit-switched flow control results in a low overhead (less than 8 %). The routers in the mesh support 6 virtual channels and perform a dimension-ordered (DOR) XY routing. In the ring-based NoC, routers support 6 virtual channels and perform routing as discussed in Sect. 3.2.3.2.

Figure 3.12 presents the latency (in terms of PE cycles) as a function of flit injection rate for uniform random traffic. On average, the ring-based NoC has a 3.5× lower latency compared to the mesh, across injection rates. At a sample 20 % injection rate,

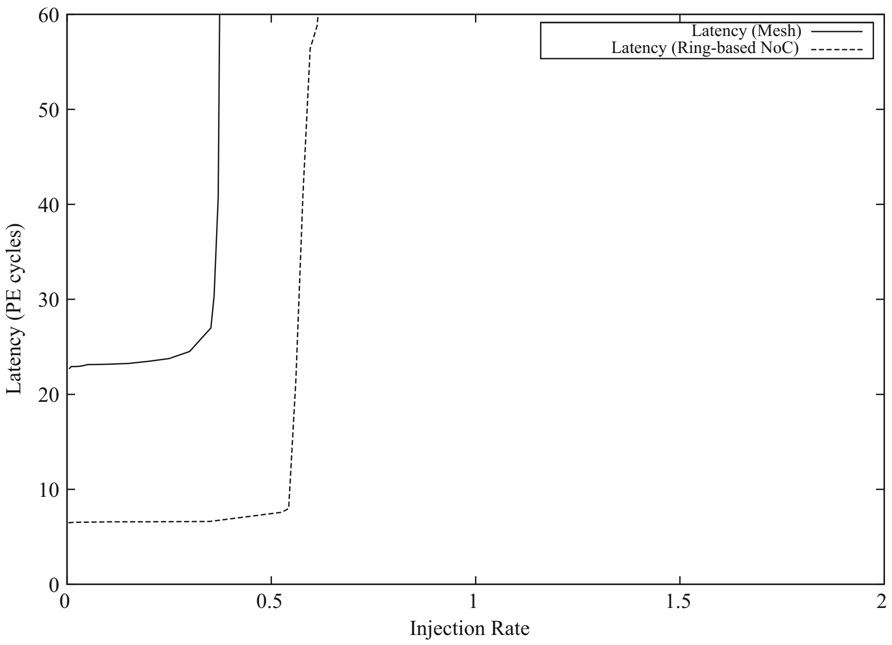


Fig. 3.12 Latency comparison for Uniform Traffic

the mesh has an average latency of 23.5 PE cycles, whereas the ring-based NoC has an average latency of 6.6 PE cycles. The mesh can only sustain a maximum injection rate of 0.38. Figure 3.13 presents the latency (in terms of PE cycles) as a function of flit injection rate for uniform traffic (just as Fig. 3.12), but using a different scale for the left y-axis. From Fig. 3.13, we observe that the ring-based NoC can sustain a maximum injection rate of 1.1. From Fig. 3.13, the injection rate for which the latency increases rapidly is the same as the injection rate for which the number of flits received saturates (for both mesh and ring-based NoCs). However, we observe that the latency curve for the ring-based NoC has a more gradual increase (kink) between the injection rates of 0.55 and 1.1. Figure 3.12 illustrates the beginning of the kink. The presence of this kink is because of the additional virtual channel from each JS to the IES on its right. Since we provide only one virtual channel for a flit for the last hop from a JS to the IES on its right, the corresponding virtual channel buffer becomes a throughput bottleneck at higher injection rates, and hence we observe the kink. The kink can be suppressed by providing additional virtual channels for the last hop from each JS to the IES on its right. We have verified that removing this virtual channel and assuming that the flit reaches the PE at the same time as it reaches the last JS (located at the left of destination IES) in the route, erases this kink. Figure 3.13 also illustrates that the number of flits received (right y-axis) saturates at an injection rate of 0.38 for the mesh and at an injection rate of 1.1 for the ring based NoC. Clearly the ring-based NoC can sustain a 2.9× higher maximum injection rate

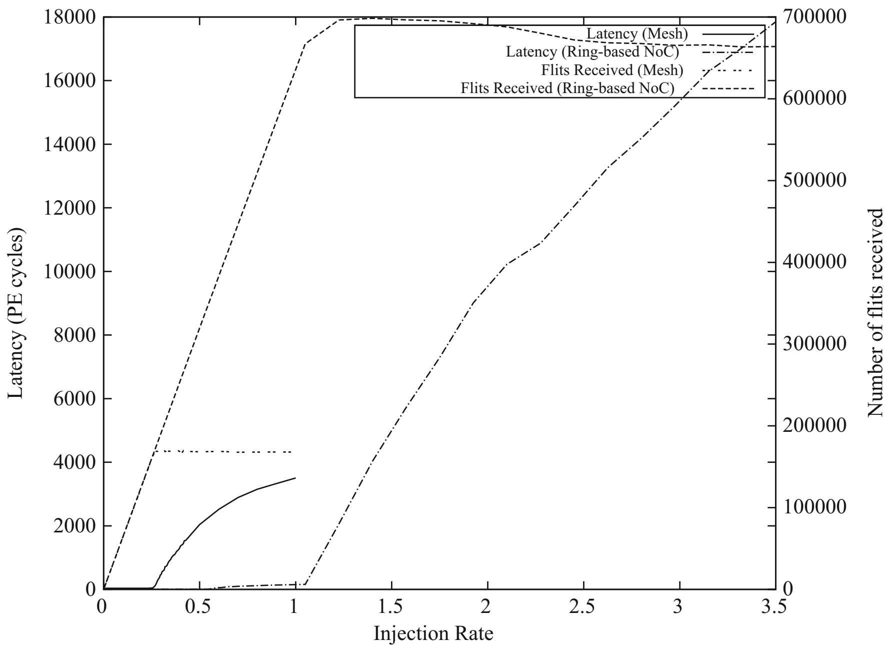


Fig. 3.13 Latency comparison for Uniform Traffic with scaled Y-axis

than the mesh. The fact that the ring-based NoC runs significantly faster than the mesh (by 7×) contributes to these improvements.

Figure 3.14 (Fig. 3.15) presents the latency and the number of flits delivered as a function of the flit injection rate for tornado (bit-complement) traffic, compared to the mesh. The ring-based NoC achieves on average 3× and 3.5× lower latency for tornado and bit-complement traffic respectively. Moreover, the ring-based NoC is able to sustain a maximum injection rate of 82 % (65 %) compared to 35 % (25 %) for the mesh, for tornado (bit-complement) traffic.

From Figs. 3.13, 3.14 and 3.15, we also observe that the ring-based NoC can deliver 2.8×, 3.3× and 2.6× more flits than the mesh-based NoC for uniform, tornado and bit-complement traffic respectively.

3.2.4.3 Effect on Virtual Channels

The number of virtual channels (VC) is a key aspect of NoC design. A virtual channel splits a single physical channel into two channels, virtually providing two paths for the flits to be routed. The use of VCs reduces the network latency at the expense of area, power consumption, and production cost of the NoC implementation. Figure 3.16 presents the latency of the ring-based NoC with two to eight virtual channels for uniform traffic. The x-axis represents the injection rate, while y-axis represents the

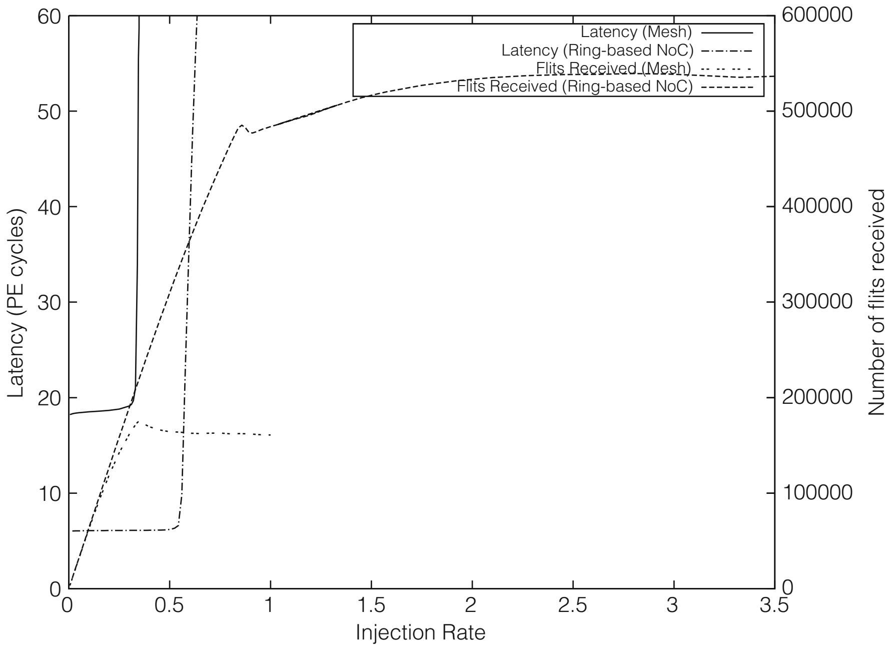


Fig. 3.14 Latency comparison for Tornado Traffic

average latency in terms of PE clock cycles. We observe that with 2 and 4 virtual channels, the maximum sustained injection rate is only 0.32 and 0.41 respectively. The maximum sustained injection rate increases to 0.55 and 1.1 for 6 and 8 virtual channels respectively. Since a state of the art mesh Michelogiannakis (2010) supports a maximum sustained injection rate of 0.38, we opt for 6 virtual channels in our ring-based NoC, to trade off complexity versus maximum sustained injection rate.

3.2.4.4 Effect on Adversarial Traffic Patterns

In our experiments, we use DOR-XY routing in the state of the art mesh. We also use DOR-XY while routing flits from one JS to another JS in our ring-based NoC. However, such a routing scheme is oblivious, which means given a source and a destination, the path is always fixed. We next implement path diversity for the baseline mesh topology and discuss the results.

We implement a path-diversity aware routing protocol called P-XY for the mesh. In P-XY, the flits always move along the shortest quadrant from a source to destination, making a turn (if possible) at every router with 50 % probability. In order to avoid deadlocks for P-XY, we split the virtual channels into two sets. If the destination is on the right of the source, we use the first set of virtual channels. If the destination is on the left of the source, we use the second set of virtual channels. If

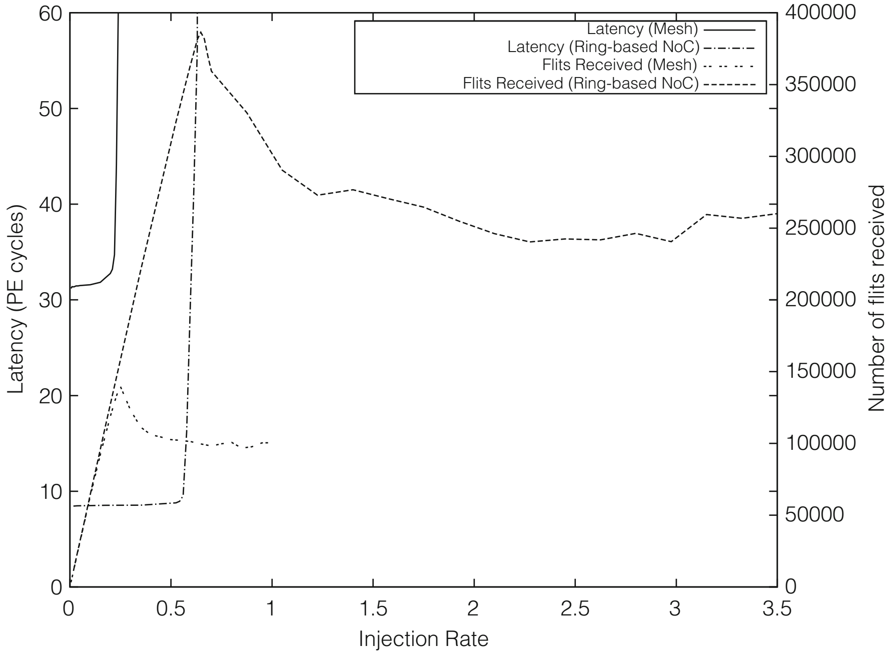


Fig. 3.15 Latency comparison for Bit-complement Traffic

the destination is on the same vertical line as that of the source then it can use any of the two sets. Figure 3.17, 3.18 and 3.19 compares the latency and number of flits delivered for P-XY and DOR-XY routing as a function of injection rate for uniform, tornado and bit-complement traffic respectively. We observe that for all of the above traffic patterns, the maximum sustained injection rate (left y-axis) and the number of flits delivered (right y-axis) are better for DOR-XY than P-XY. The reasons are as follows: (a) DOR-XY makes fewer turns (exactly one or zero). If more packets in the network make turns, they are likely to create more conflicts in the routers. (b) For P-XY, due to deadlock avoidance, we allocate two different virtual networks depending on the location of source-destination pairs. Hence this hard allocation introduces a negative impact on the available throughput. (c) For P-XY, more packets tend to go through the diagonal creating a congestion at the center. The above can be fixed by giving more weights to edges, but requires yet more complex hardware to calculate the non-uniform probabilities required to implement such a feature.

Figure 3.20 compares the latency (left y-axis) and flits delivered (right y-axis) of P-XY and DOR-XY routing as a function of flit injection rate for transpose traffic. In transpose traffic, for a $n \times n$ CMP, the destination is $(n - j, n - i)$ for a source (i, j). Let us consider all the source PEs in the column C_k of the $n \times n$ CMP. Under transpose traffic, all of the source PEs of column C_k will have their destinations in another column (say C_l). In case of DOR-XY, all the flits traveling from the column C_k to the column C_l will first traverse the links along the x-axis and then will

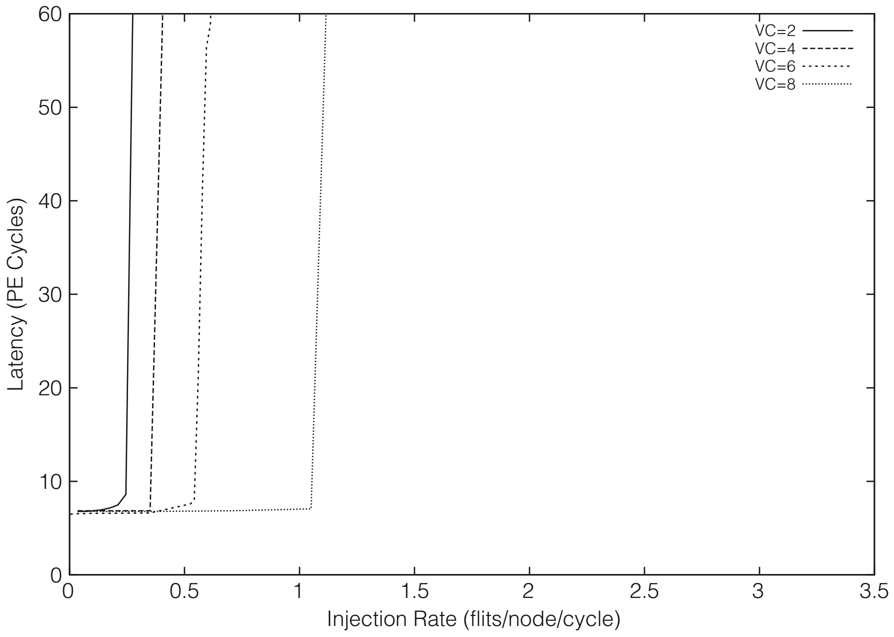


Fig. 3.16 Effect of VC on latency

traverse the y-axis links only in the column C_k. This will increase the link utilization of the center of the column C_k and hence will become a throughput bottleneck. Hence, transpose traffic is adversarial in nature for DOR-XY. We observe that for transpose traffic, DOR-XY performs worse than P-XY. In case of transpose traffic, DOR-XY heavily loads some specific links, where as P-XY distributes them equally. However, we argue that an adversarial traffic situation like transpose can be avoided at architectural level by ensuring that the source-destination pairs are placed in a way that they do not overload specific links. Since DOR-XY performs much better for other traffic patterns and is simpler to realize in hardware, we opt to use DOR-XY in our ring-based NoC design.

3.2.4.5 Benefit of Synchronous PEs

We have implemented our design in a multi-synchronous paradigm where the NoC and the PEs operate at two different clock domains. However, a multi-synchronous design suffers from the following disadvantages: (i) it requires more area for the asynchronous FIFOs present at clock crossing boundaries, (ii) it incurs an extra latency due to synchronization latencies and (iii) the synchronizers present at the clock crossing boundaries have a non-zero probability of failure due to metastability issues. Hence, a common synchronous clock for the NoC and the PEs is desirable. We achieve the above by implementing a common clock distribution scheme for the

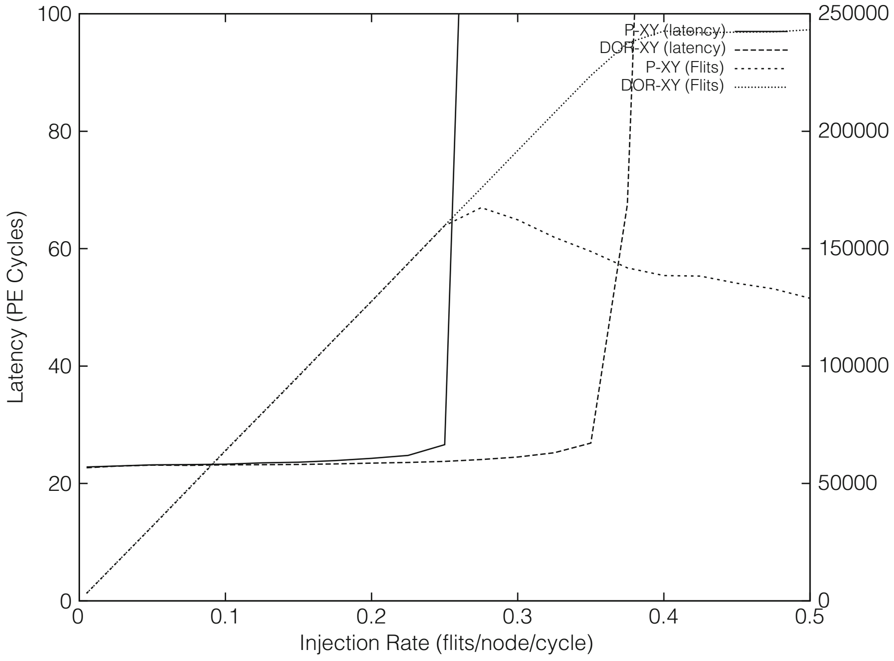


Fig. 3.17 P-XY and DOR-XY comparison for Uniform Traffic

NoC and the PEs as proposed in Sect. 2.2. In this scheme, the PEs can extract a low jitter clock from the high-speed ring clock by division. Next we analyze the impact of a synchronous design on the overall NoC performance. Figure 3.21 compares the latency (left y-axis) and the number of flits delivered (right y-axis) of the ring-based NoC with asynchronous and synchronous PEs, as a function of the flit injection rate for uniform traffic. We observe that both the designs are able to sustain the same maximum injection rate. However, the ring-based NoC with synchronous PEs achieves on average 30 % lower latency than the ring-based NoC with asynchronous PEs. This motivates us to design synchronous CMP where both the NoC and the PEs are synchronous. Hence in our subsequent NoC designs that we propose in this chapter, we utilize a common synchronous clock for both the NoC and the PEs.

3.2.4.6 Real Traffic Results

Realistic workload traces were captured for a 64-core CMP running a PARSEC benchmark Bienia et al. (2008). The traces were captured from a CMP composed of 64 in-order cores with 32-KB private L1 Instruction Cache and 32-KB private L1 Data Cache along with 16 MB of shared L2 cache. Coherence among the L1 caches was maintained using a MESI protocol. The cache line size was 144 bits. A single cache line comprised of a single-flit for mesh and 2 sub-flits for ring-based NoC.

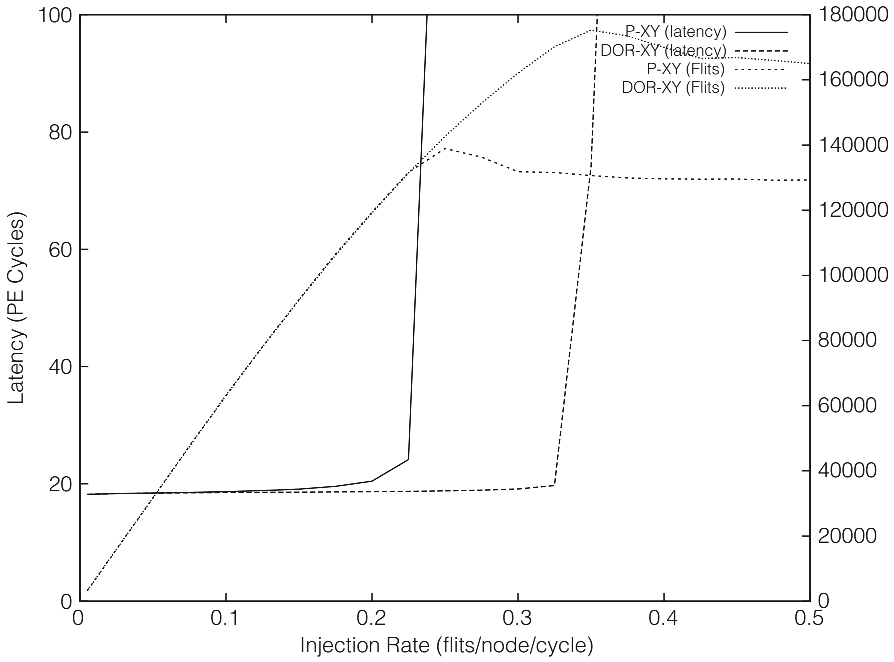


Fig. 3.18 P-XY and DOR-XY comparison for Tornado Traffic

A 100 million cycle segment of the PARSEC benchmark "region of interest" was simulated. The traffic comprised of miss requests, coherence traffic and cache line transfers.

Figure 3.22 shows the latency comparison of our ring-based NoC with mesh. The X-axis shows the PARSEC benchmarks while Y-axis shows the latency (in terms of PE cycles), normalized against the mesh. On average, the latency for our ring-based design is $\sim$ 78 % lower than that of a mesh. We observe a very low injection rate for all of these benchmarks (less than 10 %). The fact that our ring-based NoC runs 7$\times$ faster than the mesh contributes to this improvement.

3.2.5 Conclusion

Traditionally, Network-on-Chip (NoC) architectures are based on mesh interconnect structures. These existing NoCs operate at the same or lower clock speed as the PEs, hence slowing down the communication system. In Sect. 3.1, we propose a new source synchronous ring based NoC architecture, which runs significantly faster than the PEs and offers a high bandwidth and low contention free latency. In this section, we explore the architectural aspects of our fast ring-based NoC. Architectural simulations show that the baseline design suffers from deadlock. We avert the deadlock

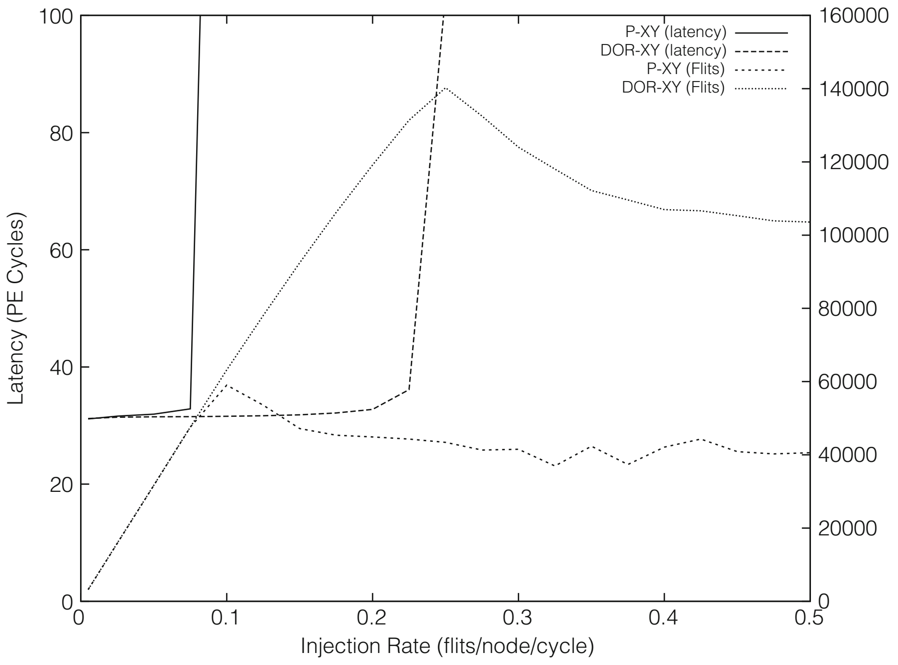


Fig. 3.19 P-XY and DOR-XY comparison for Bit-Complement Traffic

by using link ordering and virtual channels. This requires a redesign of the routers of the circuit of the ring-based NoC. We perform both circuit level and architectural simulations to validate our design. Architectural results obtained on synthetic traffic demonstrate that the ring-based NoC has a 3.5×, 3× and 3.5× lower latency and a 2.9×, 2.3× and 2.6× higher maximum sustained injection rate for uniform, tornado and bit-complement traffic respectively, compared with a state of the art mesh based NoC.

3.3 Source Synchronous H-tree based NoC

Most NoCs are implemented using the multi-synchronous paradigm. In Sect. 3.1 and Sect. 3.2, we explored the circuit and architectural aspects of a high-speed, source-synchronous NoC architecture, where the PEs communicate with the NoC using the multi-synchronous paradigm. In the following section, we propose a modified source-synchronous design where the PEs extract a low jitter clock directly from the high speed ring clock by division and hence are synchronous with the NoC. Using the above modified design, we propose a class of source-synchronous NoCs organized in an H-tree pattern which consume lower logic and wiring area while providing the same or better performance compared to a state of the art mesh.

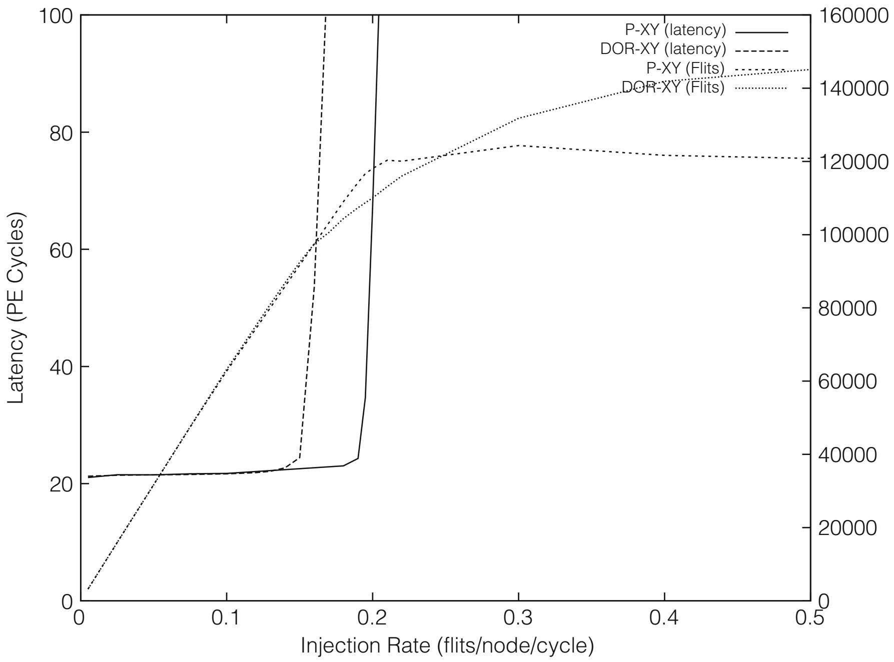


Fig. 3.20 P-XY and DOR-XY comparison for Transpose Traffic

3.3.1 Introduction

Most NoCs are implemented using the multi-synchronous paradigm. In Sect. 3.1 and Sect. 3.2, our high-speed, source-synchronous NoC architectures used the NoC using the multi-synchronous paradigm. However, a multi-synchronous design suffers from several disadvantages: i) it requires more area for the asynchronous FIFOs (present at clock crossing boundaries), ii) it incurs an extra latency due to synchronization and iii) the synchronizers present at the clock crossing boundaries have a non-zero probability of failure due to metastability issues. Hence, a common synchronous clock for the NoC and the PEs is desirable. In this section, we present a source synchronous NoC (laid out in an H-tree topology) with each link being routed in parallel with an SWO clock ring. The clock is implemented as a standing wave oscillator (SWO). Multiple clock rings are used (one for each H-tree segment), and each clock ring is injection locked with other clock rings that adjoin it. Hence the entire NoC is synchronous. The PEs can extract a low jitter clock from the high speed ring clock by clock division, making them synchronous with the NoC clock. We also show that by recursively duplicating links in the H-tree based source synchronous NoC (*Hnoc*) we can obtain new hybrid NoC designs. In the limit, in this manner, the H-tree based NoC morphs into the mesh-based source synchronous NoC (*Mnoc*). The performance of each such intermediate hybrid NoC structure is quantified. The

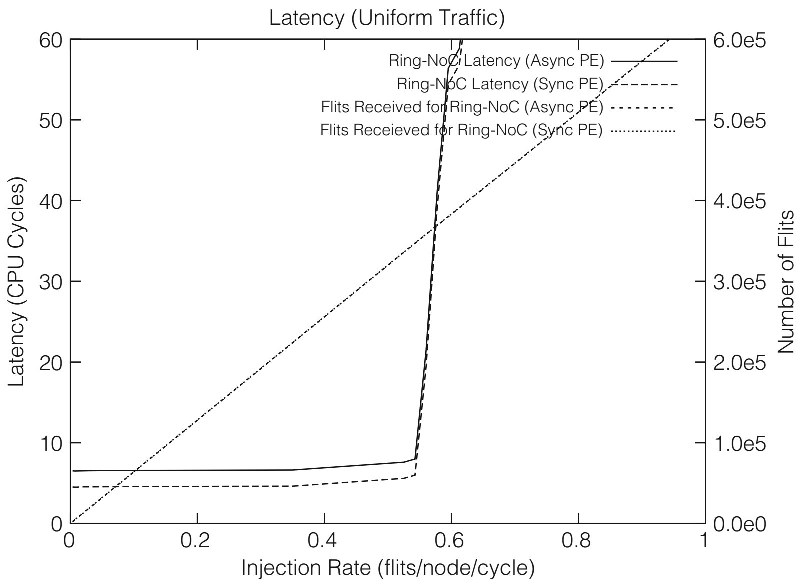


Fig. 3.21 Latency comparison with Synchronous PEs

Hnoc has the lowest area but worst performance. The *Mnoc* has largest area and best performance. Based on the performance requirements and the area budget, an NoC designer can select any intermediate hybrid NoC structure. All our designs are compared with a state-of-the art mesh based NoC Michelogiannakis (2010).

The rest of the section is organized as follows: Sect. 3.3.2 describes previous approaches in this area. Section 3.3.3 presents our approach, while Sect. 3.3.4 describes the experimental results which we performed to validate our approach. We conclude in Sect. 3.3.5.

3.3.2 Previous Work

The goal of an NoC is to provide a high performance, modular connectivity between PEs in a CMP. The NoC topology defines the way in which the PEs are connected, and directly impacts the area requirement (logic and wire) and performance of the NoC. NoC characteristics such as average hop count, routing protocol used, etc. affect the power consumption and average latency of the NoC. In terms of topology, the mesh (Dally and Towles 2001) and torus (Duato 2002) have received greatest attention due to their regular and modular structure, making them easy to implement in an IC. Application-specific topologies (Hu 2002) that can offer superior performance while minimizing area and energy consumption have been also proposed. In Tran et al. (2010), the authors implement a simple yet effective reconfigurable source-synchronous NoC, which can sustain a peak throughput of one word per cycle. In

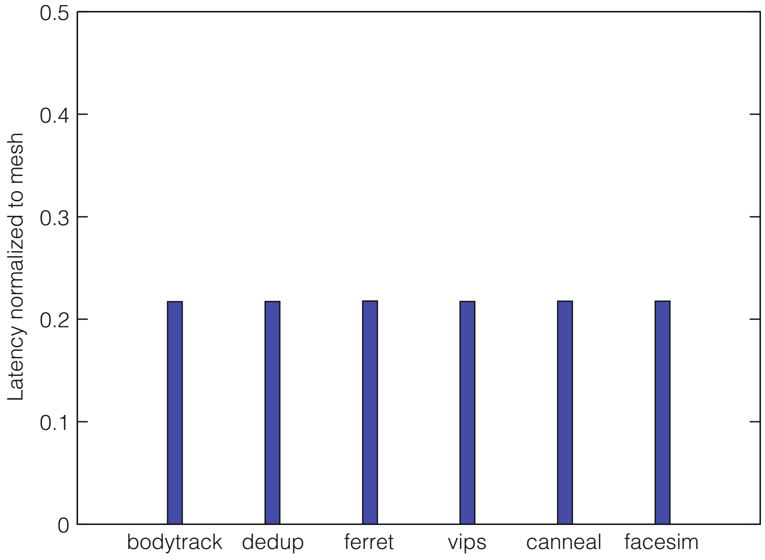


Fig. 3.22 Latency comparison for Real Traffic

Horak (2010), the authors have implemented a low-overhead asynchronous NoC, which has significantly lower latency and competitive throughput for mid-range injection rates, but suffers degradation at higher injection rates. There has been also work on asynchronous mesh NoC (Thonnart 2010), which yield a 30–50 % gain in speed and a 5× reduction in power, at the cost of 3× more area compared to a synchronous mesh.

In all the above implementations, design decisions are made based on the fact that the NoC runs at the same or lower frequency as the PEs. In Sect. 3.1 and Sect. 3.2 of this chapter, we present a fast source synchronous ring-based NoC architecture which runs significantly faster (by a factor of 7×) than the PEs' clocks. Data is driven in a source-synchronous manner along with a high speed resonant clock, providing significantly higher bisection bandwidth with narrower links (yielding a lower area and power for the same bisection bandwidth). The significantly lower latencies allow the NoC architecture to scale elegantly for larger CMPs. Architectural results obtained on synthetic traffic demonstrate that the ring-based NoC has up to 3.5× lower latency and up to 2.9× higher maximum sustained injection rate compared with a state of the art mesh-based NoC. However, in Sect. 3.2, we assume a mesh topology, where the ring clocks are synchronous amongst each other, and the PE clocks are also synchronous amongst each other. The ring and the PE clocks are asynchronous, thereby requiring asynchronous FIFOs for the communication between the PEs and the NoC, simplifying the design further. In this work, in contrast, we propose an unified synchronous clock distribution for the NoC and the PEs. In our design, the PEs can extract a low jitter clock directly from the high speed ring clock by division. Moreover, since the PE clocks are synchronous with the ring clocks, we eliminate the need for synchronizers and asynchronous FIFOs while communicating between the PEs and the NoC.

In Chap. 1.7, we have proposed various resonant clock distribution schemes. We propose a "comb" topology in Sect. 2.2 and a 2D tiled topology in Sect. 2.4. In contrast, we focus our attention on resonant clock distribution, which uses an H-tree topology. We rely on injection locking to synchronize the SWOs in any segment of the H-tree with its adjoining SWOs, with an aim of distributing clock to the leaves of the H-tree (where the PEs are located). We route the NoC datapath parallel to the resonant clock grid, both of which are arranged in an H-tree pattern. This enables the NoC logic blocks (routers) to derive a high speed, low jitter clock directly from the ring. Also, PEs can extract their clocks from the ring as well (after division).

3.3.3 *Our Approach*

In this section, we first introduce the H-tree based baseline NoC architecture used by our approach. Next, we discuss how the H-tree based source synchronous NoC (*Hnoc*) is modified by link duplication, resulting in various hybrid intermediate NoC structures, yielding a *Mnoc* in the limit. Finally, we discuss how deadlock-free routing is accomplished in these hybrid NoC designs.

3.3.3.1 Proposed Architecture

Figure 3.23 shows the baseline architecture of our proposed resonant clock rings (arranged in an H-tree pattern) and the overlaid datapath, for a three level H-tree serving 8 PEs. Each of the clock rings (dotted lines) oscillate at the same frequency, and they are injection locked with adjoining clock rings as shown in Fig. 3.23. This ensures that all rings oscillate with the same phase and frequency. A bidirectional link (solid line) is responsible for carrying the data for the NoC, and is routed parallel to the corresponding clock ring. The PEs of the CMP connect to the ring at the leaves of the H-tree through *Insertion-Extraction Stations* (IES'), which are labeled "$\times$" in Fig. 3.23. *Junction Stations* (JS') are present where the links intersect (JS' are labeled as "$\circ$" in Fig. 3.23) and are responsible for routing the flits. The IES' and JS' extract a high speed, low jitter clock directly from the clock rings. The PEs extract their clock by division from the ring clock. The design thus eliminates the need of synchronizers between the PEs and the IES', which improves the communication latency. Moreover, we avoid the need of extra hardware to maintain the read and write pointers of the asynchronous FIFOs (responsible for data transfer between the PEs and the IES') in two different clock domains. This is because the FIFOs are synchronous.

3.3.3.2 Baseline *Hnoc* Clocking

In this section, we discuss our approach towards the resonant clock distribution used in our design.

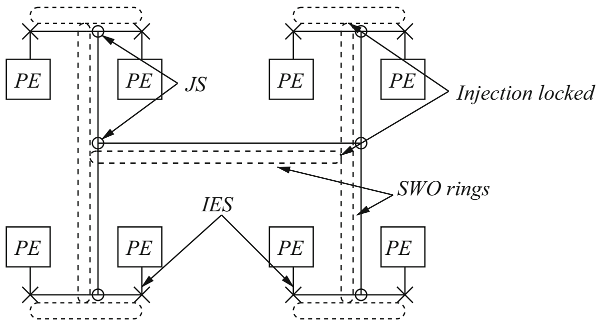


Fig. 3.23 Clock rings of *Hnoc* (dotted) with Datapath Overlaid (solid)

As discussed in Chap. 1.7, for an SWO with perimeter p, the total phase change is $\lambda/2$ while traversing the ring once. Since the different links of the H-tree have different lengths, we need to implement SWOs with different perimeter, but same frequency. Hence we increase the perimeter of the ring to $k \cdot p$ (where k is odd), so that the total phase change over a single traversal of the ring is $k \cdot \lambda/2$. Such an implementation will require p equally spaced inverter pairs, and an odd number (typically one) of mobius connections. Such a ring with perimeter $k \cdot p$ (where k is odd) oscillates at the same frequency as a ring with perimeter p. We choose the k value for each ring of Fig. 3.23 so that the ring has the desired length of the H-tree segment. Intersecting SWOs (of the same frequency) use injection locking to ensure identical phase across all the rings. Hence, we achieve a high-speed, synchronous clock distribution across the die, laid out in an H-tree topology. Note that from Fig. 3.23, we need a clock ring of perimeter $2x$ to serve a bidirectional link of length x. Based on HSPICE Inc Meta-Software (Inc Meta-Software) simulations in a 22 nm PTM (2013) process (discussed later), we obtain a 14 *GHz* clock frequency for the NoC. The required length of the metal wire is 2.75 *mm* (corresponding to $\lambda/2$) which can serve a bidirectional link of length 1.375 *mm*. For links with length greater than 1.375 *mm*, we use a ring perimeter of $k \cdot \lambda/2$, (where k is odd, and appropriately selected). The PEs (present at the leaves of the H-tree) can extract their clock by division of the SWO clocks. The above design suffers from a drawback that the clock provided to the PEs are not phase locked to the external reference. However, this is reconciled by observing that only IO PEs need a phase locked loop (PLL) in order to establish off-chip communication. Hence the IO PEs can have a separate PLL, and would require a single synchronizer while communicating with the NoC. The rest of PEs (which are not IO PEs) can derive their clock from the SWOs by division, without requiring synchronizers.

3.3.3.3 H-tree Based Source Synchronous Topologies

In this section, we discuss how the H-tree based source synchronous NoC (*Hnoc*) is modified by link duplication, resulting in various hybrid intermediate NoC structures, yielding a source-synchronous mesh NoC (*Mnoc*) in the limit. We consider an m

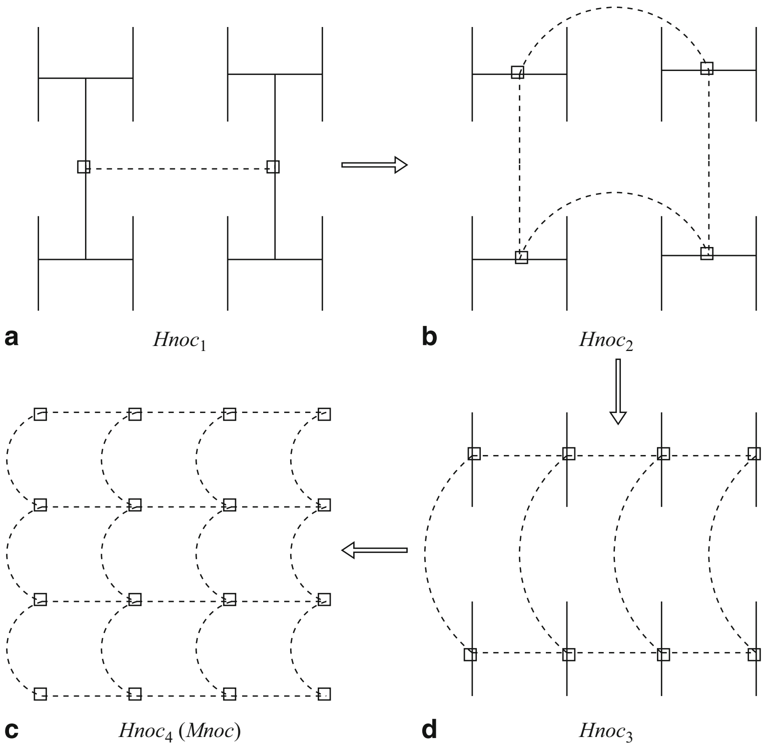


Fig. 3.24 Morphing of *Hnoc* into a *Mnoc*

level H-tree, serving $n = 2^m$ PEs. Figure 3.24 shows an example of the generation of hybrid NoC structures starting with $Hnoc_1$, for $m = 4$. In our baseline architecture, we create two $m - 1$ level H-tree based source synchronous NoCs to serve the n PEs (where $m = log_2(n)$). These two H-tree based source synchronous NoCs are connected (using dotted lines) to realize $Hnoc_1$, as shown in Fig. 3.24 a). The dotted connections are realized using a mesh based source synchronous NoC topology (as discussed in Sect. 3.1 and Sect. 3.2). In the second variant, we create four $m - 2$ level H-tree based source synchronous NoCs. These four H-tree based source synchronous NoCs are connected in form of a mesh (dotted lines) to realize the $Hnoc_2$ (Fig. 3.24b). In the third variant, we create eight 1-level H-tree based source synchronous NoCs and connect these eight nodes in the form of a mesh to realize $Hnoc_3$ (Fig. 3.24c). Finally we create sixteen 0-level H-tree based source synchronous NoCs and connect these sixteen nodes in the form of a mesh to realize $Hnoc_4$. Note that $Hnoc_4$ is same as *Mnoc* in our example. In general, at i-th step, we create 2^i $(m - i)$ level H-tree based source synchronous NoC. We connect the centers of these 2^i H-trees in the form of a mesh based source synchronous NoC, to realize $Hnoc_i$. Note that $Hnoc_m$ is identical to a *Mnoc*.

We compare the different hybrid NoC designs as constructed above, in terms of area, bisection bandwidth and contention-free latency. We also perform experimental simulations on synthetic traffic (uniform, tornado and bit-complement) as well as real traffic to quantify latency and maximum number of flits delivered (as a function of injection rate) for the various hybrid NoC topologies. In addition, we analyze the average link utilization of all the links in the NoC. This provides an insight into the bottlenecks that limit the maximum sustainable injection rate of any NoC topology. Based on this study, we propose and quantify additional NoC variants, as described in the sequel.

3.3.3.4 Deadlock Free Routing

In Dally and Seitz (1987), the authors prove that the necessary and sufficient condition for deadlock-free routing is the absence of cycles in the channel dependency graph. An H-tree by construction is acyclic. However, as we add additional links to get hybrid NoC designs, we introduce cycles in the channel dependency graph. We visualize each hybrid NoC design as a mesh with an H-tree rooted at each of the mesh nodes. The routing in the H-tree part of the topology is deadlock free (since H-tree is acyclic). We implement a deadlock free routing in the mesh by employing virtual channels along with dimension-ordered routing (DOR)-XY. This achieves deadlock free routing for all the hybrid NoC designs.

3.3.4 Experimental Results

We use a modified version of GEM5 (Binkert et al. 2011) for cycle-accurate micro-architectural NoC simulations. We simulate a network with 64 PEs. In other words, $m = 6$, and the baseline $Hnoc_1$ has 6 levels. The link width in any direction is assumed to be 18 bytes. We assume a single flit packet of 18 bytes. Note that, the above assumption simplifies the routing logic with the overhead of transmitting the header information in each packet. Since our NoCs have 64 PEs, the source and destination addresses together require 12 bits. This amounts to a minimal overhead of $\sim$8 % for a packet of 18 bytes. Each of the routers (IES' and JS') support 6 virtual channels per port and perform routing as discussed in Sect. 3.3.3.4. We compare our architectural results with a state of the art mesh with virtual channel buffered flow control (Michelogiannakis 2010). The mesh operates at 2 GHz. Although $Hnoc_6$ or *Mnoc* is topologically same as the mesh (Michelogiannakis 2010), $Hnoc_6$ uses high speed source synchronous links (operating at 14 GHz). The routers in the mesh of (Michelogiannakis 2010) are 3-stage pipelined supporting 6 virtual channels and perform a dimension-ordered (DOR)-XY routing. The routers in the mesh are capable of supporting multi-flit packet (unlike our work) by the virtue of wormhole routing, which requires complex logic. Moreover, the lack of availability of a faster on-chip clock restricts the mesh design (Michelogiannakis 2010) to operate at the same frequency (2 GHz) as the PEs.

Table 3.4 Area comparison for Hybrid NoC Topologies

Topology	Links	Wiring (mm)	Wiring (%)	Buffers	Buffers (%)
$Hnoc_1$	126	206.64	75	1500	86.8
$Hnoc_2$	124	216.48	78.6	1488	86.1
$Hnoc_3$	122	226.32	82.1	1464	84.7
$Hnoc_4$	120	236.16	85.7	1440	83.3
$Hnoc_5$	116	265.68	96.4	1392	80.6
$Hnoc_6$ ($Mnoc$)	112	275.52	100	1728	100
Mesh Michelogiannakis (2010)	112	275.52	100	1728	100

3.3.4.1 Area, Bandwidth and Contention-free Latency Comparison

Table 3.4 reports the number of links, wire length and the total number of buffers across all the routers for different hybrid NoC structures. The number of links (Column 2) is intended to provide a coarse measure of connectivity. Note that the links reported in Column 2 are not all of the same length. Hence, we report the total wire length of all the links in Column 3. Column 4 indicates the wiring length as a fraction of the wiring length of the state of the art mesh topology.

The buffers were shown to occupy 75 % of the total on-chip network area in the TRIPS chip Gratz et al. (2006) and hence we use the buffer count as an indicator of the global logical area requirement. Note that each router supports 6 virtual channels and hence the number of buffers is six times the number of ports. The number of buffers used is indicated in Column 5, while Column 6 presents the ratio of the number of buffers to that required for a mesh design.

Clearly $Hnoc_6$ (which has the same topology as a mesh) has the longest wire length and the highest number of buffers. Wire length is lowest for $Hnoc_1$ and increases as we duplicate links to create other hybrid NoC designs. In particular, $Hnoc_1$ and $Hnoc_5$ respectively have 25 % and 3.6 % lower wire length compared to a mesh. In contrast, the number of buffers reduces from $Hnoc_1$ to $Hnoc_5$. The reason behind this trend is that the number of routers decreases as we transform $Hnoc_i$ to $Hnoc_{i+1}$ (for $i \leq 4$), since JS routers are reused in this transformation, by adding an extra port. When transforming from $Hnoc_{m-1}$ to $Hnoc_m$, each of the IES requires an extra pair of ports, resulting in a steep increase in the number of required ports. In particular, $Hnoc_1$ and $Hnoc_5$ respectively have 13.2 % and 19.4 % fewer buffers compared to a mesh.

Table 3.5 compares the analytical network performance of various hybrid NoC structures for a 64 PE CMP. B_C corresponds to the bisection channel count and $\frac{B_C}{T_{clk}^{NoC}}$ defines the aggregate bandwidth (in flits/sec) available. We also report the average contention-free latency (T_0) incurred by a flit from source s to destination d which comprises: 1) the average hop count (H) from s to d, 2) router traversal latency (t_r) and 3) the average channel traversal latency (T_c), which is the latency induced by repeaters in long links. The average contention-free latency T_0 is calculated as:

$$T_0(s, d) = H(s, d) * t_r + T_c(s, d)$$

Table 3.5 Network performance for Hybrid NoC Topologies and Mesh

Topology	B_C	$\frac{B_C}{T_{clk}^{NoC}}$	H	$T_c(cycles)$	$T_0(cycles)$
$Hnoc_1$	2	14	9.68	1.66	5.81
$Hnoc_2$	4	28	8.92	1.48	5.30
$Hnoc_3$	6	42	7.90	1.27	4.66
$Hnoc_4$	8	56	7.22	0.89	3.98
$Hnoc_5$	12	84	6.75	0.82	3.71
$Hnoc_6$ (*Mnoc*)	16	112	6.25	0.75	3.42
Mesh Michelogiannakis (2010)	16	16	6.25	5.25	24

Note that we report results in Table 3.5 in terms of PE clock cycles. The bisection channel count increases from $Hnoc_1$ to $Hnoc_6$ since we duplicate links to morph the H-tree into a *Mnoc*. In particular, $Hnoc_1$ and $Hnoc_4$ respectively have $8\times$ and $2\times$ lower bisection channel count compared to a mesh. The clock frequency for the state of the art mesh was assumed to be same (2 GHz) as the PEs. Our source synchronous hybrid NoC designs ($Hnoc_1$ through $Hnoc_6$) run $7\times$ faster than the PEs. This helps us achieve a higher available aggregate bandwidth for $Hnoc_2$ through $Hnoc_5$ compared to a regular mesh, despite lower bisection channel counts. In particular, $Hnoc_2$ and $Hnoc_5$ respectively have $1.75\times$ and $5.25\times$ better available aggregate bandwidth compared to a mesh, with a $4\times$ and $1.33\times$ lower bisection channel count respectively. $Hnoc_6$, which has the same bisection channel count as the state of the art mesh, has a $7\times$ higher available aggregate bandwidth.

The router traversal latency (t_r) is 3 PE clock cycles for the mesh and 3/7 PE clock cycles for the hybrid NoC designs (since the NoC clock runs $7\times$ faster than the PEs). The average hop count decreases as we duplicate links in a H-tree to get hybrid structures. For link lengths greater the 0.615 mm, we introduce repeaters for both mesh and source synchronous NoC designs and this contributes to an increase in the average channel traversal latency (T_c). Note that only the links are buffered, not the SWO itself. Note that for both the mesh and the hybrid NoC designs, we report T_c in terms of PE clock cycles. We improve the average contention free latency (T_0) by $4.1\times$ for $Hnoc_1$ and by $7\times$ for $Hnoc_6$.

3.3.4.2 Link Utilization

We next analyze the utilization of the various links for different hybrid NoC designs. Figure 3.25 shows the histogram of the link utilization for various hybrid NoC designs, obtained through experiments using uniform traffic. All links are assumed to have the same width. Note that $Hnoc_1$ and $Hnoc_6$ are identical to an H-tree and a mesh respectively, in terms of topology. The x-axis represents the link utilization in units of flits/cycle and y-axis represents the number of links with a particular value of link utilization. The PEs inject traffic uniformly and the injection rate is chosen to ensure that the corresponding network is on the onset of saturation. Clearly for $Hnoc_1$, we observe a small number of links showing a very high link utilization. They

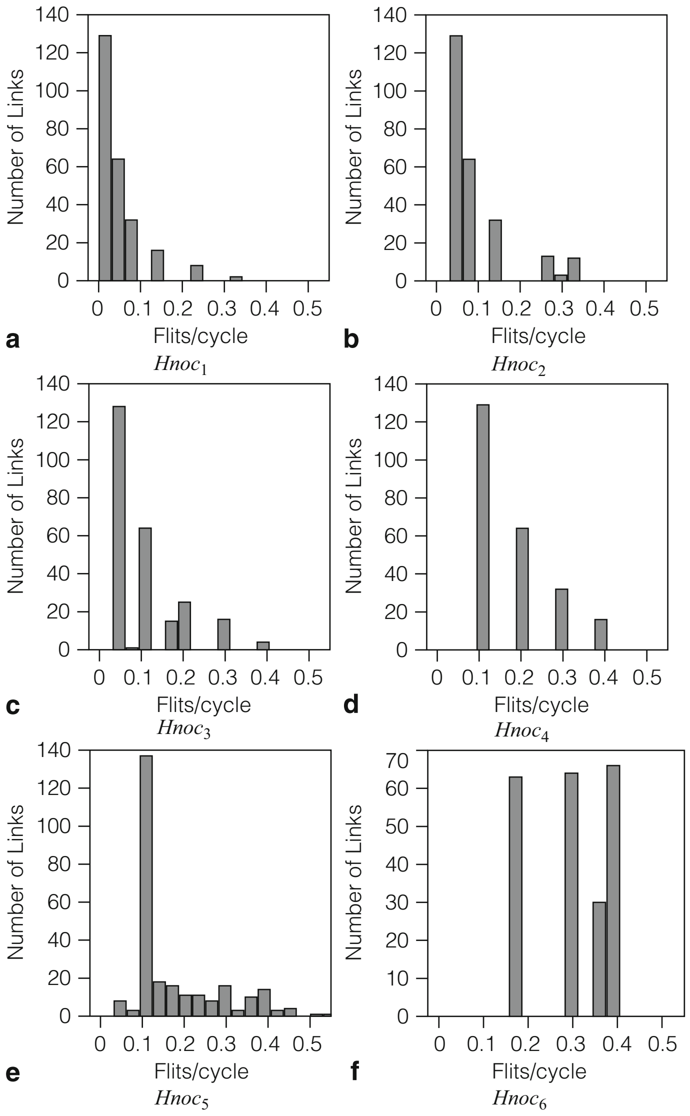


Fig. 3.25 Experimental link utilization for Hybrid NoC Topologies

correspond to the lowest level links of the H-tree (the level that is farthest from the leaves). We also observe a higher number of links with a very low utilization. They correspond to links which are present towards the leaves of the H-tree, servicing fewer PEs and hence exhibiting a lower link utilization. Next, we construct $Hnoc_2$ by removing the lowest level of H-tree. We observe more links with higher link utilization (corresponding to the mesh portion of the hybrid NoC design). As we keep

Table 3.6 Area Comparison for Hybrid NoC Topologies with Wide Links

Topology	Links	Wiring (mm)	Wiring (%)	Buffers	Buffers (%)
$Hnoc_{1,fat}$	132	221.40	80.3	1560	90.3
$Hnoc_{2,fat}$	128	226.32	82.1	1536	88.9
$Hnoc_{3,fat}$	132	295.20	107.1	1560	90.2
$Hnoc_{4,fat}$	144	354.68	128.7	1728	100
$Hnoc_{5,fat}$	168	521.52	189.3	2064	119
$Hnoc_{6,fat}$	224	551.04	200	3456	200
Mesh Michelogiannakis (2010)	112	275.52	100	1728	100

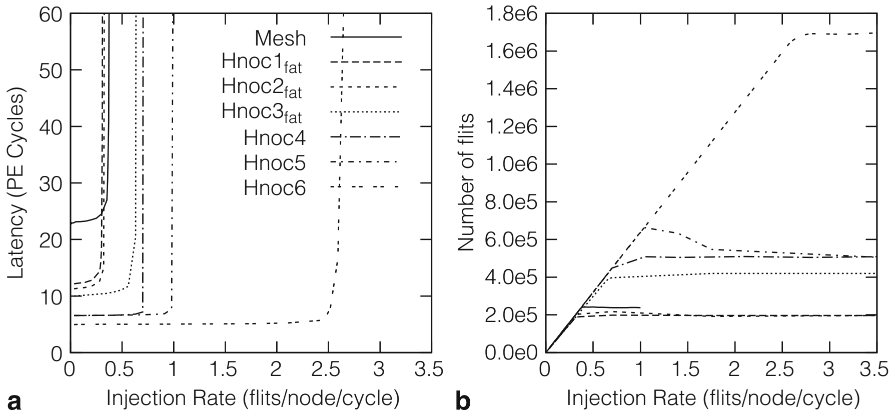
removing levels of the H-tree and growing the size of the mesh portion of the hybrid NoC, the link utilization is spread in the mesh core. The peaks in the hybrid NoC designs ($Hnoc_2$ through $Hnoc_5$) correspond to the links which are present towards the leaves of the H-tree. They serve fewer PEs and hence exhibit a lower utilization. As we morph from $Hnoc_5$ to $Hnoc_6$ (which is topologically same as the mesh), we observe a major change in pattern of the link utilization. This happens because we remove links that were responsible for serving a single PE in $Hnoc_5$ and instead create a grid of links. An ideal link utilization histogram needs to be unimodal with a very low variance. The bandwidth of a NoC design is limited by the highest utilized link. When there are a few links in the network with a high utilization, most of the links are being under-utilized when the network saturates. Such a design will not be able to sustain a high injection rate.

The purpose of the above experiment is to provide a systematic way to debug link utilization. Based on the results of this experiment, we selectively widen congested links. We widen the two lower level links (the ones farthest from the leaves) for $Hnoc_1$ and the mesh portion for the rest ($Hnoc_2$ to $Hnoc_6$) of the hybrid NoC designs by a factor of two. The wiring and buffer overheads are reported in Table 3.6. Each variant with widened links is referred to with a "*fat*" subscript. We observe that $Hnoc_{4,fat}$, $Hnoc_{5,fat}$ and $Hnoc_{6,fat}$ have significantly higher overheads compared to the baseline mesh topology. So we only present the results for $Hnoc_{1,fat}$, $Hnoc_{2,fat}$, $Hnoc_{3,fat}$, $Hnoc_4$, $Hnoc_5$ and $Hnoc_6$ in the next section.

3.3.4.3 Architectural Simulations on Synthetic Traffic

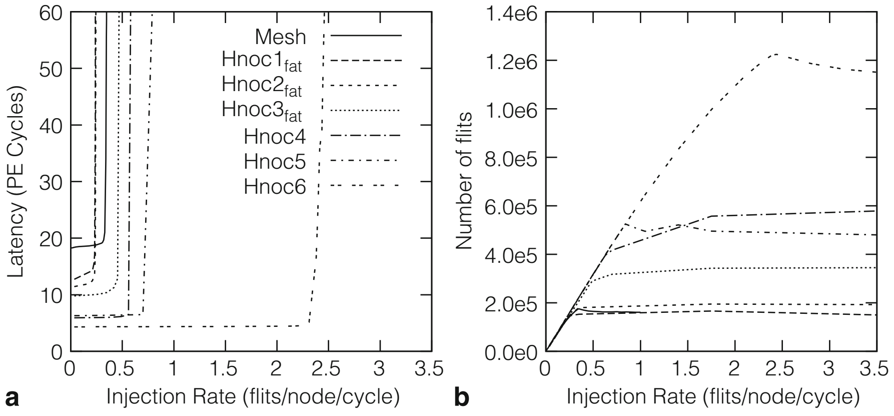
In this section, we compare the performance of different hybrid NoC structures described in Sect. 3.3.3.3 by running synthetic traffic (uniform, tornado and bit-complement). Figure 3.26a, 3.27a and 3.28a present the latency (in terms of PE cycles) as a function of injection rate, while Fig. 3.26b, 3.27b and 3.28b present the number of flits delivered over 10 K PE cycles as a function of flit injection rate for the three synthetic traffic patterns. Note that the legends in Fig. 3.26b, 3.27b and 3.28b corresponds to the legends in Fig. 3.26a, 3.27a and 3.28a respectively, and are omitted for clarity.

We summarize the data of Fig. 3.26, 3.27 and 3.28 (normalized to the mesh Michelogiannakis (2010)) in Table 3.7 and 3.8. Based on Table 3.7, we observe that all *Hnoc* design variants achieve a better latency than the state of the art mesh of



Latency Flits Delivered

Fig. 3.26 Uniform Traffic

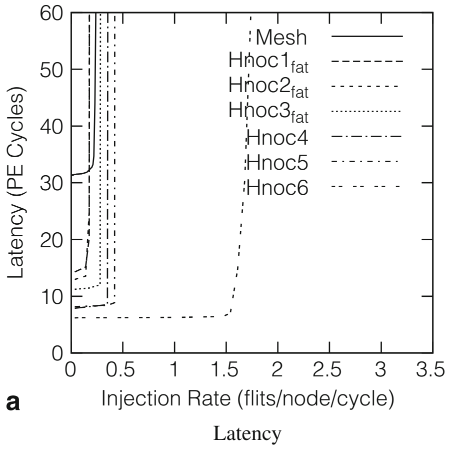


Latency Flits Delivered

Fig. 3.27 Tornado Traffic

Michelogiannakis (2010), for all three traffic types. The latency improvement varies from 1.4× to 5×. From Table 3.8, we note that $Hnoc_{3,fat}$, $Hnoc_4$, $Hnoc_5$ and $Hnoc_6$ achieve a higher number of flits than the mesh of Michelogiannakis (2010) for all traffic types. For tornado traffic, $Hnoc_{2,fat}$ improves over the mesh, while for bit-complement traffic, all variants beat the mesh in terms of numbers of flits delivered. The fact that the hybrid NoC designs run significantly faster than the mesh (by 7×) contributes to these improvements.

For uniform traffic, we observe that $Hnoc_{1,fat}$ and $Hnoc_{2,fat}$ provide 1.9× and 2× lower latency compared to the mesh of Michelogiannakis (2010). However, we observe that they can only sustain a maximum injection rate of 0.31 and 0.32 in



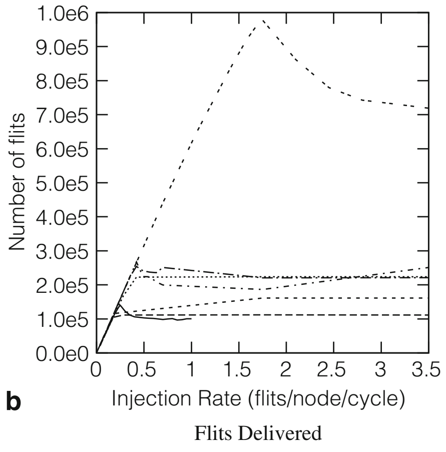


Fig. 3.28 Bit-complement Traffic

Table 3.7 Latency comparison

Topology	Uniform	Tornado	Bit-Complement
$Hnoc_{1,fat}$	1/1.9	1/1.4	1/2.3
$Hnoc_{2,fat}$	1/2.0	1/1.6	1/2.4
$Hnoc_{3,fat}$	1/2.3	1/1.9	1/2.8
$Hnoc_4$	1/3.5	1/3.1	1/3.9
$Hnoc_5$	1/3.5	1/2.9	1/3.8
$Hnoc_6$	1/4.6	1/4.2	1/5.0
Mesh Michelogiannakis (2010)	1	1	1

Table 3.8 Maximum flits delivered comparison

Topology	Uniform	Tornado	Bit-Complement
$Hnoc_{1,fat}$	0.8	0.9	1.1
$Hnoc_{2,fat}$	0.8	1.2	1.6
$Hnoc_{3,fat}$	1.8	2.1	2.2
$Hnoc_4$	2.1	3.6	2.2
$Hnoc_5$	2.1	3.0	2.5
$Hnoc_6$	7.0	7.0	7.0
Mesh Michelogiannakis (2010)	1	1	1

comparison to 0.38 for the mesh. $Hnoc_{3,fat}$, $Hnoc_4$, $Hnoc_5$ and $Hnoc_6$ provide 2.3×, 3.5×, 3.5× and 4.6× lower latency compared to a state of the art mesh and are able to sustain a maximum injection rate of 0.63, 0.7, 1 and 2.6 respectively.

For tornado traffic, we observe that $Hnoc_{1,fat}$ and $Hnoc_{2,fat}$ provide on average 1.4× and 1.6× lower latency compared to the mesh of Michelogiannakis (2010) and can only sustain a maximum injection rate of 0.23 and 0.24 in comparison to 0.35 for the mesh. $Hnoc_{3,fat}$, $Hnoc_4$, $Hnoc_5$ and $Hnoc_6$ provide a 1.9×, 3.1×, 2.9× and 4.2× lower latency compared to a mesh and are able to sustain better injection rates than the mesh of Michelogiannakis (2010) (0.45, 0.6, 0.8 and 2.4 respectively). The maximum number of flits delivered over 10 K PE cycles for $Hnoc_{2,fat}$, $Hnoc_{3,fat}$, $Hnoc_4$, $Hnoc_5$ and $Hnoc_6$ are between 1.2× and 7× higher compared to the mesh of Michelogiannakis (2010), as reported in Table 3.8. For bit-complement traffic, we observe that $Hnoc_{1,fat}$ and $Hnoc_{2,fat}$ provide 2.3× and 2.4× lower latency compared to the mesh of Michelogiannakis (2010) but can only sustain an injection rate of 0.18 and 0.19 respectively in comparison to 0.25 for the mesh. $Hnoc_{3,fat}$, $Hnoc_4$, $Hnoc_5$ and $Hnoc_6$ provide a 2.8×, 3.9×, 3.8× and 5× lower latency compared to the mesh of Michelogiannakis (2010) and are able to sustain a higher injection rate than the mesh (0.3, 0.37, 0.45 and 1.8 respectively). The maximum number of flits delivered over 10 K PE cycles for all *Hnoc* variants are between 1.1× and 7× higher than the mesh of Michelogiannakis (2010), as reported in Table 3.8.

Hence we conclude that $Hnoc_{1,fat}$, $Hnoc_{2,fat}$ have significantly lower latency and comparable maximum sustained injection rate in comparison to the state of the art mesh. Moreover, $Hnoc_{3,fat}$, $Hnoc_4$, $Hnoc_5$ and $Hnoc_6$ provide a better latency and are able to sustain a higher injection rate for all synthetic patterns in comparison to a state of the art mesh. Among them, $Hnoc_5$ has the lowest number of buffers and $Hnoc_4$ has the smallest wire length. By taking this tradeoff into account, a designer can choose a hybrid NoC design depending on the area-performance characteristics they wish to achieve.

3.3.4.4 Real Traffic Results

Realistic workload traces were captured for a 64-core CMP running a PARSEC benchmark (Bienia et al. 2008). The traces were captured from a CMP composed of 64 in-order cores with 32-KB private L1 Instruction Cache and 32-KB private L1 Data Cache along with 16 MB of shared L2 cache. The cache line size was 18 bytes. A single cache line comprises of a single flit in all the hybrid NoCs. Coherence among the L1 caches was maintained using a MESI protocol. A 100 million cycle segment of the PARSEC benchmark "region of interest" was simulated. The traffic comprised of miss requests, coherence traffic and cache line transfers.

Figures 3.29 and 3.30 shows the latency comparison of H-tree based NoCs and the state of the art mesh. The x-axis shows the PARSEC benchmarks while y-axis shows the latency (in terms of PE cycles), normalized against the state of the art mesh. From Fig. 3.29, we observe that the latency for $Hnoc_{1,fat}$, $Hnoc_{2,fat}$ and $Hnoc_{3,fat}$ are on average 46 %, 51 % and 55 % lower than that of a state of the art mesh. From Fig. 3.30, we observe that the latency for $Hnoc_4$, $Hnoc_5$ and $Hnoc_6$ are on average 70 %, 72 % and 78 % lower than that of a state of the art mesh. We observe a very low injection rate for all of these benchmarks (less than 10 %). The fact that these hybrid NoC designs run 7× faster than the mesh contributes to this improvement.

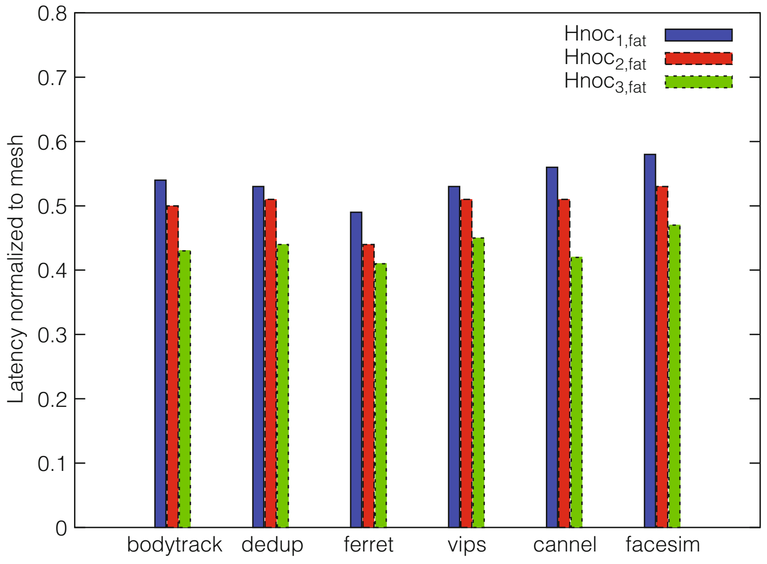


Fig. 3.29 Real Traffic comparison for *Hnoc*$_{1,fat}$, *Hnoc*$_{2,fat}$ and *Hnoc3*, *fat*

3.3.5 Conclusion

In this section, we present a family of high-speed source synchronous NoCs organized in an H-tree topology, with each data link being routed parallel to a clock ring. The clock is generated with the help of injection locked standing wave oscillators of varying length as required by the topology. PEs directly extract a low jitter clock from the high speed ring clock by division. We compare different hybrid NoC designs with a state of the art mesh (Michelogiannakis 2010) in terms of area, link utilization and contention-free latency. We also explore hybrid H-tree based source synchronous NoCs with selective link widening. We also perform experimental simulations on synthetic traffic (uniform, tornado and bit-complement) as well as real traffic to quantify latency and maximum number of flits delivered as a function of injection rate for various hybrid NoC structures. Our results demonstrate significant improvements over a state of the art mesh. The fact that the hybrid NoC designs run significantly faster than the traditional mesh (by 7×) contributes to these improvements. Using our results, a designer can choose any hybrid NoC design depending on the area-performance characteristics desired.

In the previous section, we proposed a modified source-synchronous design where the clock and data NoC are synchronous yielding a fast, robust design. In such a source-synchronous design, the PEs extract a low jitter clock directly from the high speed ring clock by division and hence are synchronous with the NoC. In the following section, we use the above idea to propose and evaluate two additional NoC topologies with significantly lower logic and wiring area with the same or better performance compared to a state of the art mesh.

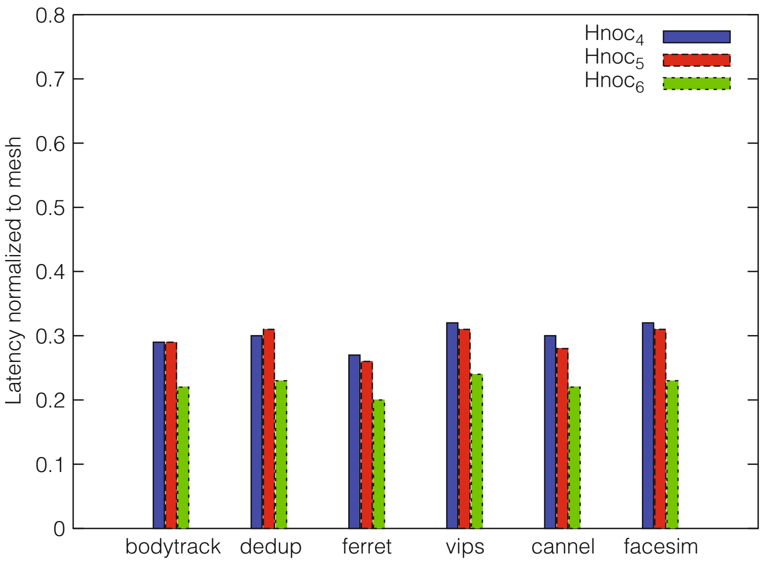


Fig. 3.30 Real Traffic comparison for $Hnoc_4$, $Hnoc_5$ and $Hnoc6$

3.4 Exploring Ring of Star and Spine with Ring Topology for NoC

3.4.1 Introduction

In Sect. 3.1 and Sect. 3.2, we introduced a high-speed, source-synchronous ring-based NoC architecture. We refer to the topology used in Sect. 3.2 as Mesh of Rings (MOR), which consumes the same buffer area as that of a state of the art mesh. In an NoC design, the topology determines the way in which the PEs are connected. In Sect. 3.2 and Sect. 3.3, we observe how topologies affect the bandwidth and latency of a network. Our proposed ring-based NoC designs run significantly faster than the PEs, yielding improved performance. In Sect. 3.3, we explore H-tree based topologies, which consume lower logic and wiring area compared to a state of the art mesh with the same link width. In this section, we explore two alternate source synchronous ring-based topologies called the ring of stars (ROS) and the spine with rings (SWR) which consume significantly lower logic and wiring area than the state of the art mesh, and are able to provide better performance in terms of communication latency compared to a state of the art mesh. The ROS topology is constructed by connecting multiple star networks in form of a ring. Each star network is hierarchically constructed by connecting smaller star networks. The smallest star network directly connects the PEs. The SWR topology consists of concentric rings with a horizontal and a vertical spine connecting all the rings. In both of the proposed topologies, the data NoC is being routed in parallel to a clock ring, which is driven by a fast resonant clock. We use standing wave resonant oscillators (SWOs) to implement the clock rings. Any SWO ring is injection locked with other SWO rings that adjoin it and hence the entire NoC is synchronous. Our design allows the PEs to extract a low jitter clock from the high speed ring clock by division, thus making the PEs synchronous with the ring clock. Hence we eliminate the need of synchronizers between the PEs and the NoC. We evaluate the area and performance of the two new ring-based NoC topologies and compare them with a state of the art mesh.

The rest of this section is organized as follows. Section 3.4.2 describes previous approaches in this area. Section 3.4.3 presents our approach, while Sect. 3.4.4 describes the experimental results which we obtained while validating our approach. We conclude in Sect. 3.4.5.

3.4.2 Previous Work

The previous work reported in Sect. 3.3.2 is applicable for this section as well. In 2010, Sanchez et. al. (2010) compared various network topologies of interconnection networks in terms of latency, throughput, and energy dissipation. The authors report that for a 64-core CMP, the total area utilization is lowest in case of the mesh topology. The two additional source synchronous ring-based topologies in this section consume much lower area than the mesh, and are able to provide better performance in terms of communication latency compared to a state of the art mesh.

3.4.3 Our Approach

In this section, we introduce the two alternate source synchronous ring-based topologies followed by their corresponding deadlock-free routing approach. In the following discussion, we assume a CMP with 64 PEs, for all NoC topologies.

3.4.3.1 The ROS Topology

- *Architecture:* Figure 3.31 shows our proposed ring of stars (ROS) topology. The smallest star network (level-1) connects 4 PEs and consists of 4 IES' (circles) and 1 JS (square). A level-2 star network is responsible for connecting 4 level-1 star networks. Hence, a level-2 star network consists of 5 JS' as shown. Finally, we have a ring which connects 4 level-2 stars to provide full connectivity for the 64-PE CMP. We highlight a portion of the NoC around the JS in the left of Fig. 3.31. The JS is present at the intersection of two clock rings. The two intersecting clock rings are injection locked and hence oscillate with the same frequency and phase. We also show the five bidirectional links connected to the JS that are responsible for carrying the data for the NoC. The JS extracts a high speed, low jitter clock directly from these clock rings.

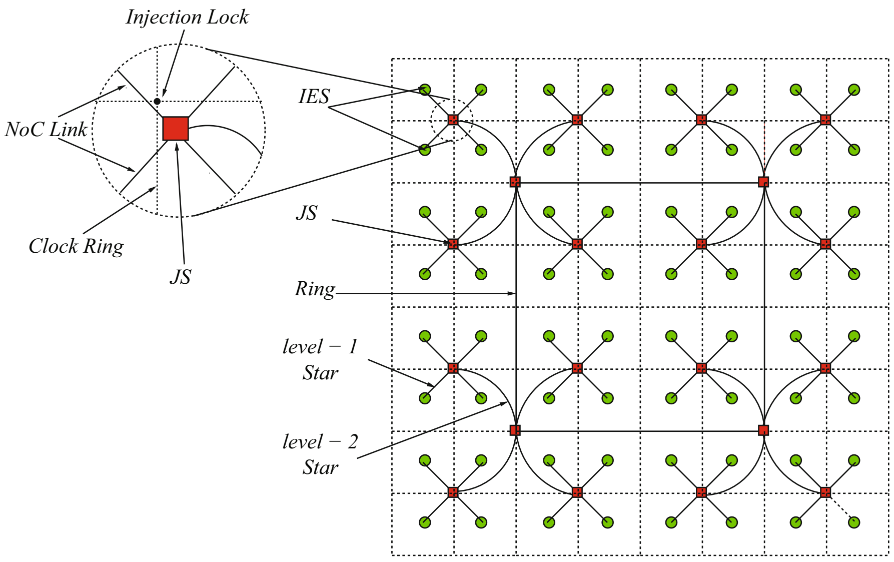


Fig. 3.31 ROS Topology

- *Deadlock-free Routing:* In Dally and Seitz (1987), the authors prove that the necessary and sufficient condition for deadlock-free routing is the absence of cycles in the channel dependency graph. A star-topology by construction is acyclic. However, as we introduce a ring to connect multiple stars, we introduce cycles in the channel dependency graph. We visualize the ROS as a 2×2 mesh with a star network rooted at each of the four mesh nodes. The routing in each of the star network segments is deadlock free (since the star network is acyclic). We establish a deadlock free route in the mesh by using virtual channels and dimension-ordered routing (DOR)-XY. This achieves a deadlock free route for the entire ROS.

3.4.3.2 The SWR Topology

- *Architecture:* Figure 3.32 shows our proposed spine with rings (SWR) topology. The topology consist of 4 concentric rings with a vertical and horizontal spine connecting the rings. The innermost ring is the smallest ring and connects only 4 PEs at the center using IES' (circles). The outermost ring is the largest and connects 28 PEs which are present the periphery of the CMP. One vertical and one horizontal spine connect all the 4 rings to provide full connectivity for the 64-PE CMP. IES's in Fig. 3.32 are shown as circles, while JS' are shown as squares. A portion of the NoC around the JS is highlighted in the left of Fig. 3.32. The two intersecting clock rings at the JS are injection locked, and hence oscillate with the same frequency and phase. We also show the four bidirectional links connected to the JS which are responsible for carrying the data for the NoC. The JS extracts a high speed, low jitter clock directly from these clock rings.

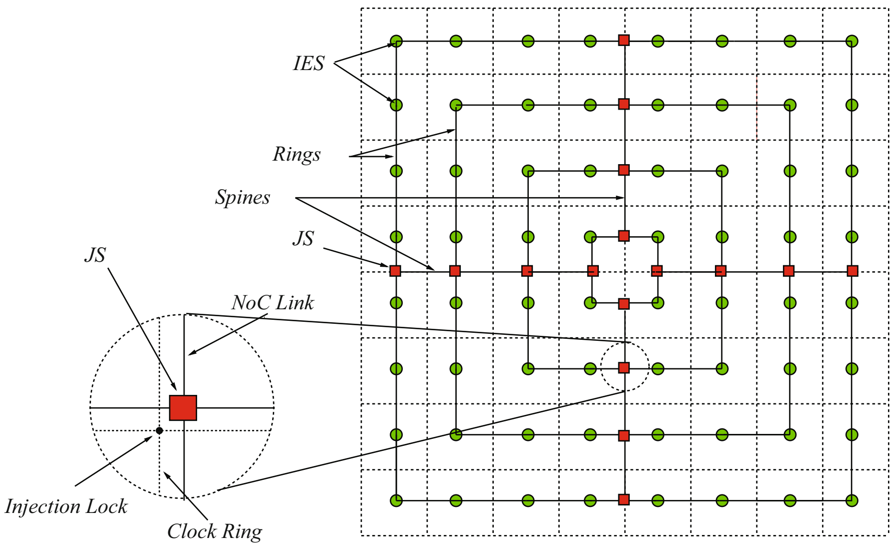


Fig. 3.32 SWR Topology

- *Deadlock-free Routing:* The SWR is composed of 4 concentric rings and a vertical and a horizontal spine connecting these concentric rings. Clearly the SWR has cycles in the channel dependency graph. Each router (IES and JS) in our design supports 6 virtual channels. We construct two virtual networks with the help of these 6 virtual channels. Three of the virtual channels are reserved for flits which travel from an outer ring to an inner ring. The other half of the virtual channels is reserved for flits traveling from an inner ring to outer ring. Without two virtual networks, deadlock can occur when traffic simultaneously is routed from inner rings to outer rings, as well as from outer rings to inner rings. With two virtual networks, the flits of these two virtual networks do not share resources and do not create a cycle in the channel dependency graph. This achieves a deadlock free route for SWR.

We compare the above ring-based NoC topologies in terms of area, average communication latency and maximum number of flits delivered. In addition, we analyze the average link utilization (theoretical and experimental) of the links in the ring-based NoCs, which provides an insight to the maximum injection rate sustainable by an NoC structure, and allows us to debug the link(s) that are responsible for congestion.

3.4.4 Experimental Results

We use a modified version of GEM5 (Binkert et al. 2011) for cycle-accurate micro-architectural NoC simulations. For all topologies, we simulate a network with 64 PEs. Each PE tile is assumed to be 1.229 $mm \times 1.229$ mm (using the estimates from Sect. 3.1.3.2). The link width in any direction is assumed to be 18 bytes. The routers

Table 3.9 Area comparison for various NoC Topology

Topology	Number of Links	Wire Length (mm)	Number of ports	Number of buffers
MOR	112	137.65	288	1728
ROS	84	206.10	168	1008
SWR	92	98.32	144	864
Mesh Michelogiannakis (2010)	112	275.30	288	1728

(IES' and JS') operate at 14 GHz and can drive a maximum link length of 0.615 mm. For links greater than 0.615 mm, we insert repeater(s) to ensure that largest link driven is 0.615 mm. We assume a clock frequency of 2 GHz for the PEs and a clock frequency of 14 GHz for the various ring-based NoC designs. We assume a single flit packet of 18 bytes. Each of the routers (IES' and JS') support 6 virtual channels. We compare our architectural results with a state of the art mesh with virtual channel buffered flow control Michelogiannakis (2010). The mesh operates at 2 GHz. The routers in the mesh are 3-stage pipelined supporting 6 virtual channels and perform a dimension-ordered (DOR)-XY routing.

3.4.4.1 Area Comparison

Table 3.9 reports the number of links, wire length, the total number of input ports and the total number of buffers across all the routers for the MOR, ROS, SWR and mesh Michelogiannakis (2010) topologies. The first column is intended to provide a coarse measure of connectivity. Note that the links reported in Column 1 are not all of the same length. Hence, we report the total wire length of all the links in the second column. The buffers were shown to occupy 75 % of the total on-chip network area Gratz et al. (2006) in the TRIPS chip and hence we use them as an indicator of the global logic area requirement. Note that since each router supports 6 virtual channels, the number of buffers is six times the number of ports.

Clearly the MOR and the state of the art mesh have the highest number of buffers. MOR has 50 % lower wire length as the link width is half compared to the state of the art mesh. The ROS and SWR have 25.2 % and 64.3 % lower wire length respectively compared to the state of the art mesh. Moreover, the ROS and the SWR have 41.7 % and 50 % fewer buffers respectively, compared to the mesh. In the rest of this section, we show that our ROS and SWR topologies leverage the benefit of a high speed NoC by reducing the wiring length and the number of buffers while still providing lower communication latency compared to a state of the art mesh.

3.4.4.2 Link Utilization

In this section, we analyze the utilization of the various links for different ring-based NoC designs. Figure 3.33 shows the histogram of the analytical and experimental link utilization for the ROS and SWR topologies. For analytical link utilization, we assume that each link is able to provide an unlimited bandwidth to the incoming flits.

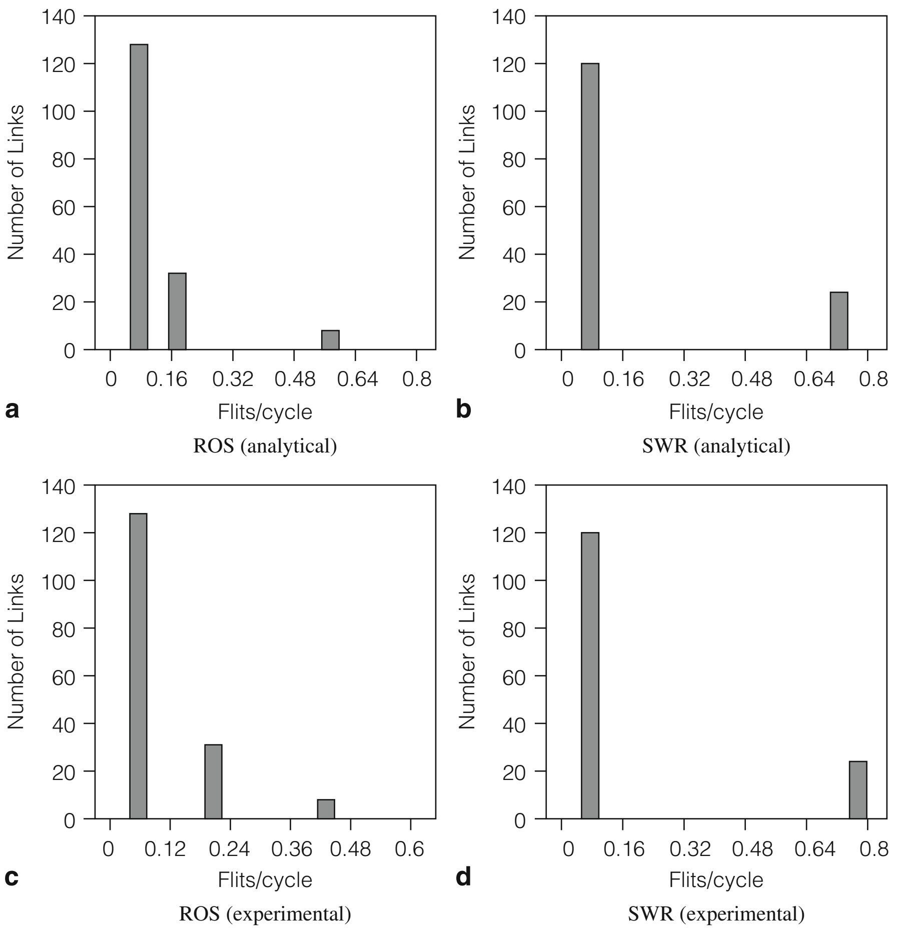
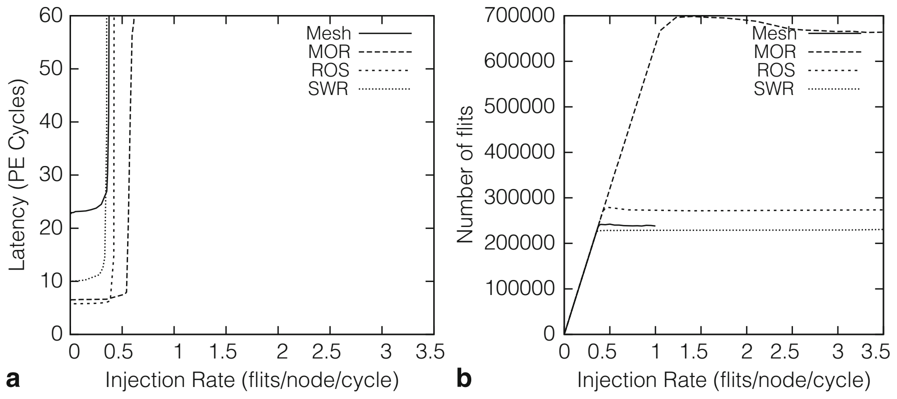


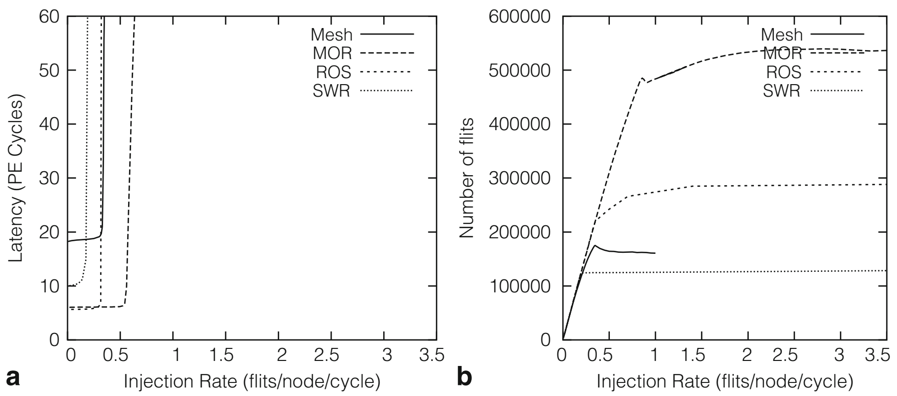
Fig. 3.33 Link utilization of ROR and SWR

For experimental link utilization, the PEs inject traffic uniformly and the injection rate is chosen to ensure that the corresponding network is at the onset of saturation. The injection rates used for both ROS and SWR were 0.35. The same injection rate was used for the analytical experiment as well. For the ROS topology, the level-1 star networks have the lowest utilization since each of them serves only 4 PEs. The utilization increases for the level-2 star network which serves 16 PEs. Finally, the ring which connects four level-2 star networks has the highest utilization. For the SWR topology, all the links present in the concentric rings have same utilization due to symmetry. However, the spines which are responsible for connecting all the concentric rings have the highest link utilization. From Fig. 3.33, we observe that there is a very close resemblance between the analytical and the experimental link utilization. Ideally, we prefer a link utilization distribution that is uniform across all the links. For a ROS topology, the link utilization is non-uniform due to its



Latency Flits Delivered

Fig. 3.34 Uniform Traffic



Latency Flits Delivered

Fig. 3.35 Tornado Traffic

hierarchical topology. For the SWR topology, the link utilization is uniform across all the links in the concentric rings but is highest for the links in the spine connecting the rings. The above study provides an insight to the network load of each topology and also gives an indication about the maximum sustained injection rate. The results of the above experiment can allow the NoC designer to determine which links need to be widened, and thereby fine-tune the performance versus the area trade off.

3.4.4.3 Synthetic Traffic Results

We compare the performance of different ring-based NoC designs with a state of the art mesh by running synthetic traffic (uniform, tornado and bit-complement) through these NoCs. Figures 3.34a, 3.35a and 3.36a presents the latency on the y-axis

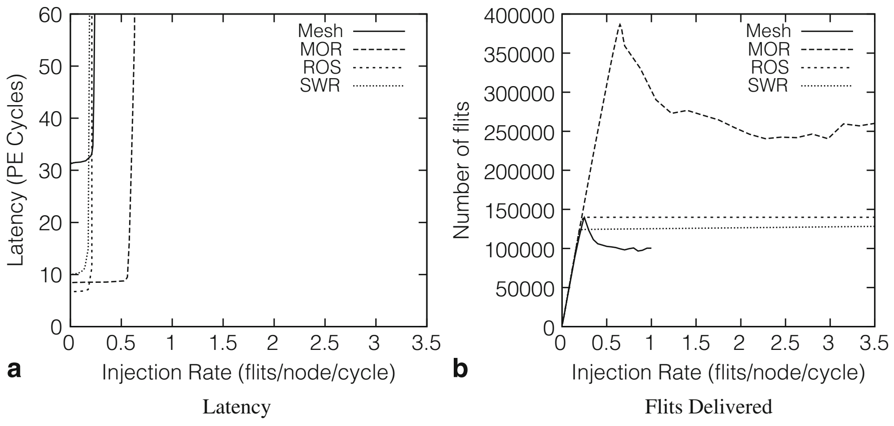


Fig. 3.36 Bit-complement Traffic

(in terms of PE cycles) as a function of flit injection rate on the x-axis. Figures 3.34b, 3.35b and 3.36b provides the number of flits delivered (y-axis) over 10 K PE cycles as a function of flit injection rate (x-axis) for the corresponding traffic patterns. For uniform traffic, we observe that MOR, ROS and SWR provide on average 3.6×, 3.9× and 2.3× lower latency compared to a state of the art mesh. Moreover, from Fig. 3.34b, we observe that MOR, ROS and SWR can sustain an injection rate of 1.1, 0.42 and 0.36 respectively in comparison to 0.38 for the mesh.

We summarize the data of Figs. 3.34, 3.35 and 3.36 (normalized to the state of the art mesh (Michelogiannakis 2010) in Tables 3.10 and 3.11. The maximum number of flits delivered over 10 K PE cycles for MOR, ROS and SWR are 5.5×, 1.2× and 1.9× compared to the state of the art mesh as reported in Table 3.11. The fact that the ring-based NoC designs run significantly faster than the mesh (by 7×) contributes to these improvements.

For tornado traffic, we observe that MOR, ROS and SWR provide on average 3×, 3× and 1.7× lower latency compared to a mesh and can sustain a maximum injection rate of 0.85, 0.38 and 0.2 respectively in comparison to 0.35 for the mesh. The maximum number of flits delivered over 10 K PE cycles for MOR, ROS and SWR are 6.6×, 1.8× and 1.6× compared to the mesh as reported in Table 3.11. Finally, for bit-complement traffic, we observe that MOR, ROS and SWR provide on average 3.7×, 4.6× and 3× lower latency compared to a mesh and can sustain a maximum injection rate of 0.66, 0.23 and 0.2 respectively in comparison to 0.23 for the mesh. The maximum number of flits delivered over 10 K PE cycles for MOR, ROS and SWR are 3.8×, 1.1× and 1.9× compared to the mesh as reported in Table 3.11.

Clearly, the MOR has the best performance in terms of latency and maximum sustained injection rate. However, we also observe that the ROS and the SWR provide a better latency (with a minimum of 1.7× better) for all synthetic patterns in comparison to a state of the art mesh. ROS is able to sustain the same or better maximum injection rate as that of the mesh across all synthetic traffic patterns. However,

Table 3.10 Latency comparison

Topology	Uniform	Tornado	Bit-Complement
Mesh Michelogiannakis (2010)	1	1	1
MOR	1/3.6	1/3	1/3.7
ROS	1/3.9	1/3	1/4.6
SWR	1/2.3	1/1.7	1/3

Table 3.11 Maximum flits delivered comparison

Topology	Uniform	Tornado	Bit-Complement
Mesh Michelogiannakis (2010)	1	1	1
MOR	5.5	6.6	3.8
ROS	1.2	1.8	1.1
SWR	1.9	1.6	1.9

SWR sustains a lower maximum injection rate than that of the mesh across all synthetic traffic patterns. In the SWR, as suggested by the link utilization discussion, the spine becomes the throughput bottleneck at higher injection rates. Both the SWR and ROS topologies use fewer buffers (upto 50 % less) and lower wire length (upto 64.3 % lower) compared to the mesh, and therefore are significant improvements over the state of the art in this respect. Figures 3.34, 3.35 and 3.36 indicates that both MOR and ROS are able to deliver a larger number of flits than the mesh, but with a significantly lower latency and area utilization.

3.4.4.4 Real Traffic Results

Realistic workload traces were captured for a 64-core CMP running a PARSEC benchmark Bienia et al. (2008). The traces were captured from a CMP composed of 64 in-order cores with 32-KB private L1 Instruction Cache and 32-KB private L1 Data Cache along with 16 MB of shared L2 cache. The cache line size was 18 bytes. A single cache line comprises of a single flit for the ring-based NoC designs. Coherence among the L1 caches was maintained using a MESI protocol. A 100 million cycle segment of the PARSEC benchmark "region of interest" was simulated. The traffic comprised of miss requests, coherence traffic and cache line transfers.

Figure 3.37 shows the latency comparison of ROS and SWR with respect to the state of the art mesh. The x-axis shows the PARSEC benchmarks while y-axis shows the latency (in terms of PE cycles), normalized against the state of the art mesh. We observe that the average latency for ROS is 49 % lower than that of the state of the art mesh. The latency for SWR is 45 % lower than that of the mesh. We observe a very low injection rate for all of these benchmarks (less than 10 %). The fact that our ring-based NoC runs 7× faster than the mesh contributes to this improvement.

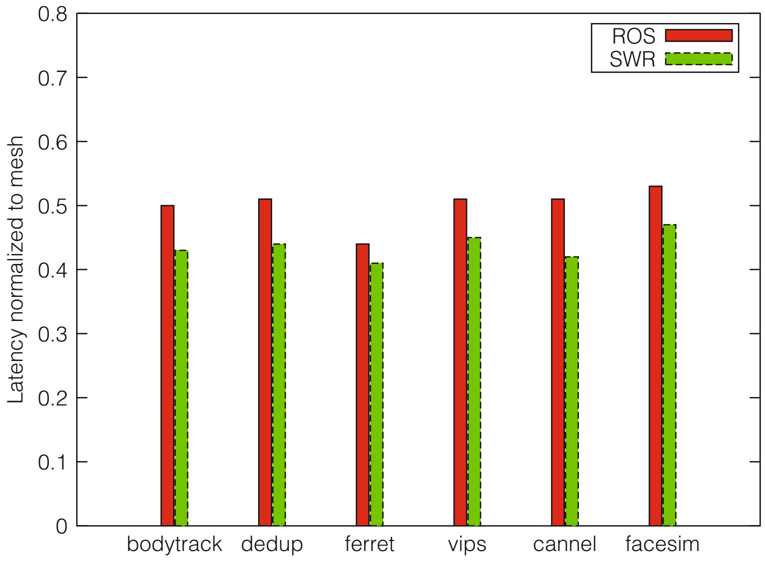


Fig. 3.37 Latency comparison for real Traffic

3.4.5 Conclusion

In this section, we evaluate two additional source synchronous ring-based NoC topologies which consume much lower area and are able to provide better performance in terms of communication latency compared to a state of the art mesh. In our proposed topologies, the clock and the data NoC are routed in parallel, yielding a fast, robust design. Our design allows the PEs to extract a low jitter clock from the high speed ring clock by clock division. The area and performance of these ring-based NoC topologies is quantified. Experimental results on synthetic traffic show that the new ring-based NoC designs can provide significantly lower latency (by upto 4.6×) compared to a state of the art mesh. The proposed topologies use fewer buffers (upto 50 % less) and lower wire length (upto 64.3 % lower) compared to a state of the art mesh. The proposed ROS topology is able to sustain the same or better injection rate as that of the mesh across all synthetic traffic patterns. However, the SWR topology sustains a lower injection rate for all synthetic traffic patterns compared to the mesh, but has a significantly lower latency.

3.5 Chapter Summary

Traditionally, Network-on-Chip (NoC) architectures are based on a mesh interconnection structures. In this chapter, we propose a ring based NoC architecture which is based on a source synchronous data transfer model over a ring. The source synchronous ring is clocked by a resonant clock which operates significantly faster than

Table 3.12 Latency comparison for various NoC Topologies

Topology	Uniform	Tornado	Bit-Complement
MOR	1/3.6	1/3	1/3.7
$Hnoc_{1,fat}$	1/1.9	1/1.4	1/2.3
$Hnoc_{2,fat}$	1/2.0	1/1.6	1/2.4
$Hnoc_{3,fat}$	1/2.3	1/1.9	1/2.8
$Hnoc_4$	1/3.5	1/3.1	1/3.9
$Hnoc_5$	1/3.5	1/2.9	1/3.8
$Hnoc_6$ (*Mnoc*)	1/4.6	1/4.2	1/5.0
ROS	1/3.9	1/3	1/4.6
SWR	1/2.3	1/1.7	1/3
Mesh Michelogiannakis (2010)	1	1	1

the individual processors that are served by the ring. This allows us to significantly reduce the area devoted to the NoC logic and wiring. We have validated the circuit-level design of our proposed NoC components using a 22 nm predictive process.

Next, we explore the architectural aspects of our fast ring-based NoC. We avert deadlock by using link ordering and virtual channels. Architectural results obtained on synthetic traffic demonstrate that the ring-based NoC has a significantly lower latency (upto 3.5×) a higher maximum sustained injection rate (upto 2.9×) for synthetic traffic, compared with a state of the art mesh based NoC. Next, we present a family of high-speed source synchronous NoCs organized in an H-tree topology, with each data link being routed parallel to a clock ring.

Our design allows the PEs to directly extract a low jitter clock from the high speed ring clock by division. Our baseline design resembles an H-tree, and we recursively duplicate links to arrive at a mesh topology in the limit. We compare these different hybrid NoC designs in terms of area, link utilization and contention-free latency. We also perform experimental simulations on synthetic traffic to quantify the latency and maximum sustained injection rate for the hybrid NoC structures.

Using our results, a designer can choose any hybrid NoC design depending on the area-performance characteristics desired. Finally, we evaluate two additional source synchronous ring-based NoC topologies which consume much lower area and are able to provide better performance in terms of communication latency compared to a state of the art mesh. Similar to our H-tree based NoC designs, the clock and the data NoC are routed in parallel, yielding a fast, robust design. The design also allows the PEs to extract a low jitter clock from the high speed ring clock by clock division. The area and performance of these ring-based NoC topologies are quantified.

The proposed topologies use fewer buffers (upto 50 % less) and lower wire length (upto 64.3 % lower) compared to a mesh. Experimental results on synthetic traffic show that the proposed topologies are able to sustain the same or better injection rate as that of the mesh across all synthetic traffic patterns but with a significantly lower latency (upto 4.6×).

We summarize the average latency and maximum number of flits delivered for various NoC topologies considered in this chapter (normalized to the mesh (Michelogiannakis 2010)) in Tables 3.12 and 3.13 respectively. In addition, we summarize

Table 3.13 Maximum flits delivered for various NoC Topologies

Topology	Uniform	Tornado	Bit-Complement
MOR	5.5	6.6	3.8
$Hnoc_{1,fat}$	0.8	0.9	1.1
$Hnoc_{2,fat}$	0.8	1.2	1.6
$Hnoc_{3,fat}$	1.8	2.1	2.2
$Hnoc_4$	2.1	3.6	2.2
$Hnoc_5$	2.1	3.0	2.5
$Hnoc_6$ (*Mnoc*)	7.0	7.0	7.0
ROS	1.2	1.8	1.1
SWR	1.9	1.6	1.9
Mesh Michelogiannakis (2010)	1	1	1

Table 3.14 Area comparison for various NoC Topology

Topology	Wiring (mm)	Wiring (%)	Buffers	Buffers (%)
MOR	137.65	50	1728	100
$Hnoc_{1,fat}$	221.40	80.3	1560	90.3
$Hnoc_{2,fat}$	226.32	82.1	1536	88.9
$Hnoc_{3,fat}$	295.20	107.1	1560	90.2
$Hnoc_4$	236.16	85.7	1440	83.3
$Hnoc_5$	265.68	96.4	1392	80.6
$Hnoc_6$ (*Mnoc*)	275.30	100	1728	100
ROS	206.10	74.8	1008	58.3
SWR	98.32	35.7	864	50
Mesh Michelogiannakis (2010)	275.30	100	1728	100

the wiring and buffer area for various NoC topologies considered in this chapter (normalized to the mesh Michelogiannakis (2010)) in Table 3.14.

Clearly, from Tables 3.12 and 3.13 we observe that $Hnoc_6$ or *Mnoc* which has the same topology as a state of the art mesh, but operates significantly faster than the state of the art mesh (7× faster), has the lowest latency and highest number of flits delivered across all traffic patterns. However, from Table 3.14, we observe that $Hnoc_6$ consumes the same wiring and buffer area as that of state of the art mesh. Among the other *Hnoc* topologies, all of them have significantly lower latency compared to the state of the art mesh while consuming comparable or lower wiring and buffer area. Among these, $Hnoc_{3,fat}$, $Hnoc_4$ and $Hnoc_5$ are able to deliver more flits compared to a state of the art mesh. MOR consumes the same buffer area as that of a state of the art mesh but only half the wiring area. MOR has significantly lower latency as well delivers significantly higher number of flits compared to a state of the art mesh. Also, we can further improve MOR performance by making the NoC clock and the PE clock synchronous as is the situation with hybrid H-tree topologies as well as SWR and ROS topology. Finally, we observe that both the ROS and SWR consume significantly lower wiring and buffer area and still have significantly lower latency compared with a state of the art mesh. The SWR topology is able to deliver more flits than the SWR topology, and both of these topologies deliver more flits than the state of the art mesh. Hence depending on the desired throughput in a CMP, one designer can select any of these NoC designs.

References

Rajeev Balasubramonian, Naveen Muralimanohar, Karthik Ramani, and Venkatanand Venkatachalapathy, “Microarchitectural Wire Management for Performance and Power in Partitioned Architectures,” in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, Washington, DC, USA, 2005, pp. 28–39, IEEE Computer Society.

James D. Balfour and William J. Dally, “Design tradeoffs for tiled CMP on-chip networks,” in *International Conference on Supercomputing*, 2006, pp. 187–198.

Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li, “The PARSEC benchmark suite: Characterization and architectural implications,” Tech. Rep., IN PRINCETON UNIVERSITY, 2008.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood, “The GEM5 simulator,” *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

T. Bjerregaard, “The MANGO clockless network-on-chip: Concepts and implementation,” 2005, Supervised by Assoc. Prof. Jens Sparsø, IMM.

L. Bononi, N. Concer, M. Grammatikakis, M. Coppola, and R. Locatelli, “NoC Topologies Exploration based on Mapping and Simulation Models,” in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, 2007, pp. 543–546.

T. Chelcea and S.M. Nowick, “A low-latency FIFO for mixed-clock systems,” in *VLSI, 2000. Proceedings. IEEE Computer Society Workshop on*, 2000, pp. 119–126.

D. M. Chiu, M. Kadansky, R. Perlman, J. Reynders, G. Steele, and M. Yuksel, “Deadlock-free routing based on ordered links,” in *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks*, Washington, DC, USA, 2002, LCN ’02, pp. 0062–, IEEE Computer Society.

E C Cummings and Peter Alfke, “Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons,” *Technical Report, Sunburst Design*, 2002.

W. J. Dally and C. L. Seitz, “The Torus Routing Chip,” *The Journal of Distributed Computing*, vol. 1(3), pp. 187–196, 1986.

W. J. Dally and C. L. Seitz, “Deadlock-free message routing in multiprocessor interconnection networks,” *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, May 1987.

W J Dally and J W Poulton, *Digital Systems Engineering*, Cambridge University Press, 1998.

W.J. Dally and B. Towles, “Route packets, not wires: on-chip interconnection networks,” in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 684–689.

Jose Duato, Sudhakar Yalamanchili, and Ni Lionel, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

G. Gerosa, S. Curtis, M. D’Addeo, Bo Jiang, B. Kuttanna, F. Merchant, B. Patel, M.H. Taufique, and H. Samarchi, “A Sub-2W Low Power IA Processor for Mobile Internet Devices in 45 nm High-k Metal Gate CMOS,” *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 1, pp. 73–82, 2009.

P. Gratz, Changkyu Kim, R. McDonald, S.W. Keckler, and D. Burger, “Implementation and Evaluation of On-Chip Network Architectures,” in *Computer Design, 2006. ICCD 2006. International Conference on*, Oct 2006, pp. 477–484.

M.N. Horak, S.M. Nowick, M. Carlberg, and U. Vishkin, “A Low-Overhead Asynchronous Interconnection Network for GALS Chip Multiprocessors,” in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, May 2010, pp. 43–50.

Jingcao Hu, Yangdong Deng, and Radu Marculescu, “System-level point-to-point communication synthesis using floorplanning information,” in *Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, Washington, DC, USA, 2002, ASP-DAC ’02, pp. 573–, IEEE Computer Society.

Inc Meta-Software, "HSPICE user's manual," Campbell, CA.

F. Karim, A. Nguyen, and S. Dey, "An interconnect architecture for networking systems on chips," *Micro, IEEE*, vol. 22, no. 5, pp. 36–45, Sep/Oct 2002.

J. Kim, J. Balfour, and W.J. Dally, "Flattened butterfly topology for on-chip networks," *Computer Architecture Letters*, vol. 6, no. 2, pp. 37–40, Feb. 2007.

M.M. Kim, J.D. Davis, M. Oskin, and T. Austin, "Polymorphic On-Chip Networks," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, June 2008, pp. 101–112.

Charles E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, pp. 892–901, October 1985.

Daniele Ludovici, Alessandro Strano, Georgi N. Gaydadjiev, and Davide Bertozzi, "Mesochronous NoC technology for power-efficient GALS MPSoCs," in *Proceedings of the Fifth International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*, New York, NY, USA, 2011, INA-OCMC '11, pp. 27–30, ACM.

George Michelogiannakis, Daniel Sanchez, William J. Dally, and Christos Kozyrakis, "Evaluating bufferless flow control for on-chip networks," in *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, Washington, DC, USA, 2010, NOCS '10, pp. 9–16, IEEE Computer Society.

U Nawathe, "Design and implementation of Sun's Niagara2 processor," *Technical Report, Sun Microsystems*, 2007.

L Peh H Wang and S Malik, "Power-driven design of router microarchitectures in on-chip networks," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, dec. 2003, pp. 105–116.

"PTM website," http://www.eas.asu.edu/~ptm (Accessed April 22, 2013).

"Raphael Interconnect Analysis Tool: User's Guide," .

H. Samuelsson and S. Kumar, "Ring Road NoC architecture," in *Norchip*, 2004, pp. 16–19.

Daniel Sanchez, George Michelogiannakis, and Christos Kozyrakis, "An analysis of on-chip interconnection networks for large-scale chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 7, pp. 4:1–4:28, May 2010.

Yvain Thonnart, Pascal Vivet, and Fabien Clermidy, "A fully-asynchronous low-power framework for GALS NoC integration," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 3001 Leuven, Belgium, Belgium, 2010, DATE '10, pp. 33–38, European Design and Automation Association.

Sergio Tota, Mario R. Casu, and Luca Macchiarulo, "Implementation analysis of NoC: a MPSoC trace-driven approach," in *Proceedings of the 16th ACM Great Lakes symposium on VLSI.* 2006, GLSVLSI '06, pp. 204–209, ACM.

Anh Thien Tran, Dean Nguyen Truong, and B. Baas, "A Reconfigurable Source-Synchronous On-Chip Network for GALS Many-Core Platforms," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 6, pp. 897–910, June 2010.

Chapter 4
Fast On-Chip Data Transfer Using Sinusoid Signals

Abstract In Chapter 2, we demonstrate that resonant clocking can be used as an ultra high-speed, low-jitter, low-power, stable on-chip clock generation and distribution scheme. In Chapter 3, we use such a clock to design a high speed source-synchronous ring-based NoC architecture. This helped us achieve inter-processor communication with minimal latency. In this chapter, we investigate an alternate design for high speed on-chip data transfer, which utilizes resonant oscillators. Traditional (pulse-based) on-chip data transfer achieves a maximum data transfer rate of one bit per wire per clock cycle. In this work, we explore the use of sinusoidal signals (generated using SWOs) of different frequencies as information carriers for on-chip data transfer. The advantage of our method is the ability to superimpose such sinusoids and thereby effectively send multiple logic values along the same wire in a clock cycle. Initial experimental results show that for the same throughput as a traditional scheme, we require 50 % fewer wires. This technique can be employed for off-chip data transfer as well.

4.1 Introduction

There is a significant incentive to improve the speed of on-chip data communication. It is well known that ICs have substantially followed Moore's Law, doubling their complexity approximately every 2 years. In contrast, data communication rates between the processing cores within an IC have improved less dramatically, proving to be a bottleneck in the the quest for faster computing. The fundamental reason for this is that data is communicated as a sequence of pulses. With such a choice, noise and signal are parallel vectors, making fast, reliable data transfer difficult.

In this work, we investigate the transmission of data as an additive superposition of several sinusoidal tones. Two significant gains are obtained with this choice. First, since the information is contained in the frequency of the sinusoidal tones, noise is orthogonal to the signal, yielding an extremely robust communication scheme. A second key feature of our scheme is that the data rate grows as more sinusoidal tones are utilized. Our initial experiments using 4 sinusoids suggest that our proposed scheme would yield a data communication rate which is at least 2× better than that achieved by the traditional pulse based signaling, for the same wiring area. Currently

DOI 10.1007/978-1-4614-9405-8_4, © Springer Science+Business Media New York 2014

our experiments focus on on-chip data transfer. However, the idea can be applied for off-chip data transfer as well. Although our current experiments utilize 4 sinusoids, we anticipate the ability to use significantly more sinusoids. As a consequence, this scheme is expected to achieve much higher data rates for on-chip as well as off-chip data transfer schemes.

The rest of this section is organized as follows: Sect. 4.2 describes previous approaches in this area. Sect. 4.3 presents our approach, while Sect. 4.4 describes the results of experiments which we performed to validate our approach. In Sect. 4.5, we draw conclusions.

4.2 Previous Work

Transmission lines (Beckmann and Wood 2003) were proposed to implement a low latency interconnect between the L2 cache banks and the cache controllers. The authors of (Beckmann and Wood 2003) have outlined CMP floorplans optimized for less complex circuitry, where the cache banks are located along the edges of the chip and cache controllers reside at the center of the chip. While transmission lines provide low-latency interconnects, they lack the capability of transmitting multiple logic values on a single wire. In Ogras et al. (2006), the authors explored application-specific long-range links between pairs of frequently communicating cores. They implement their long-range links using a higher-latency point-to-point pipelined bus. The application-specific nature of these long-range links makes them unsuitable for use in a general-purpose architecture, and no algorithms are presented to adapt their use to changing communication conditions. Moreover a higher-latency link can degrade the performance of the cores which are communicating. In Kirman et al. (2006), the authors employed optical technology to design a low-latency, high-bandwidth shared bus. While their design does take advantage of low-latency and high bandwidth via simultaneous transmission on different wavelengths, they examine optical interconnect to augment a shared bus topology.

Radio Frequency Interconnect (RF-I) (Chang et al. 2008) was proposed as a scheme for high aggregate bandwidth, low latency communication for off-chip as well as on-chip data transfer. On-chip RF-I is realized by transmitting baseband data on a carrier wave using amplitude and/or phase modulation. The modulation of the baseband signal in (Chang et al. 2008) involves up-conversion with a Gilbert Cell. The addition of multiple frequency bands along the same wire is performed with the help of unwieldy on-chip inductors. In contrast, we use a CMOS common source amplifier to add various sinusoidal tones with different phases. Moreover the RF-I schemes use large passive devices to filter and process the baseband data at the receiver. In contrast, we use a mixer and two stages of differential amplifiers to recover the transmitted signal. RF-I requires multiple carrier frequencies in the mm-wave range, which are typically generated on-chip using sub-harmonic injection locking VCOs. These VCOs are LC-tuned cross-coupled pair with current control.

In contrast, we use injection locked standing wave oscillators (SWOs) (Cordero and Khatri 2008) to generate the sinusoidal tones. Recent research on such oscillators suggests that they yield stable, low power, sinusoidal signal generators.

4.3 Our Approach

In this section, we first introduce our sinusoid-based technique of achieving on-chip data transfer. Next, we briefly discuss our proposed transmitter and receiver design.

4.3.1 Overview

Our approach utilizes n sinusoidal tones as the carriers of information. We use standing wave oscillators (SWOs) to generate these sinusoid signals. We generate the n sinusoidal tones by using n SWOs, which are all injection locked to a common SWO with a $pilot$ frequency F_p. Each SWO generates two sinusoidal signals with 0° and 180° phase. Using n oscillators, and by selecting either phase for each oscillator, we obtain $2n$ basis tones. We send these tones in a differential manner along two wires. If the phase of any sinusoid is 0° (180°) on the first wire, then it is 180° (0°) on the second. A new set of n superposed sinusoidal tones are sent during each period of the $pilot$ frequency F_p. From the basis tones on any wire, we can perform the superpositions to yield a hyperspace of 2^n basis symbols. In effect, this means that each differential pair of wires in our approach is equivalent to n wires in a traditional pulse based data transfer scheme. The resulting signal, which is the sum of all n sinusoids is transmitted differentially. The $pilot$ signal is also transmitted source-synchronously (for synchronization) at the receiver. The receiver also has n SWOs, which are all injection locked to the incoming $pilot$ signal. The received signal (sum of n sinusoids) is separately mixed with the n sinusoidal tones to recover the transmitted symbols.

4.3.2 Transmitter

Figure 4.1 shows our proposed transmitter circuit driving two differential on-chip wires. On the left, we show n SWOs of frequencies F_1, F_2, ... F_n respectively. All the SWOs are injection locked to a common SWO with the pilot frequency F_p. Two common source adders (CSAs) are used to add either a 0° or a 180° phase of each of the n sinusoidal tones obtained from the SWOs. Two pass gates connect each of the SWOs with the CSAs. The pass gate from the SWO with frequency F_i is controlled by the complementary signals C_i and $\overline{C_i}$. When C_i is high, the 0° phase of frequency F_i is send to CSA-1 and 180° phase of frequency F_i is send to CSA-2, and vice versa.

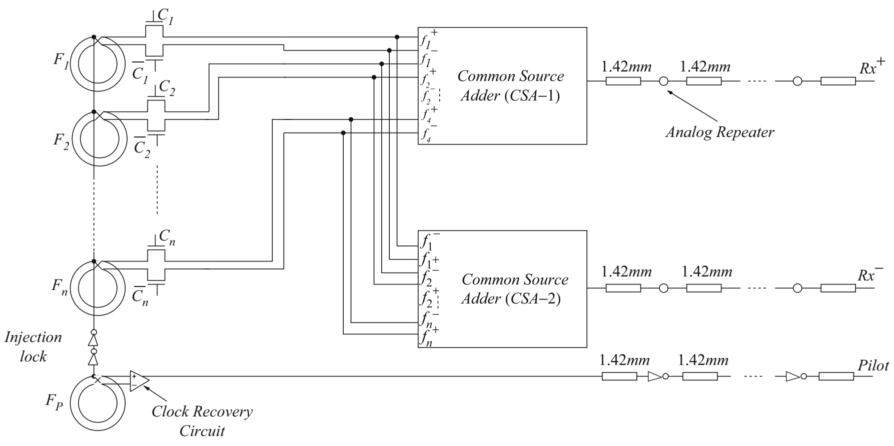


Fig. 4.1 Transmitter Circuit

Fig. 4.2 Common Source Amplifier

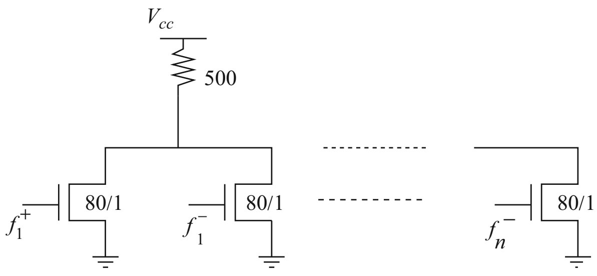


Fig. 4.3 Analog Repeaters

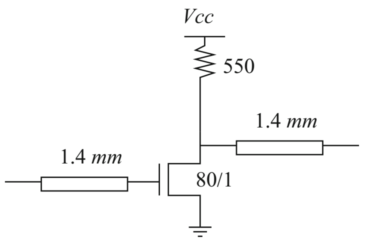


Figure 4.2 shows the internal circuit of the CSA. It has $2n$ parallel common source stages pulled up by a resistor. Exactly n out of the $2n$ inputs are active during one *pilot* clock cycle. The CSA performs the addition of these n sinusoids and drives a wire of length 1.42 mm. We introduce analog repeaters (as shown in Fig. 4.3) every 1.42 mm of the wire to amplify the signal. This distance is maximum that the CSA and repeaters can drive the signals, while guaranteeing that the data can be recovered. The *pilot* signal is also transmitted in a source synchronous manner, as shown in Fig. 4.1. We use regular inverters to regenerate the drive strength of the *pilot* signal, since it is a digital signal. The inverters are inserted every 1.42 mm to ensure that the

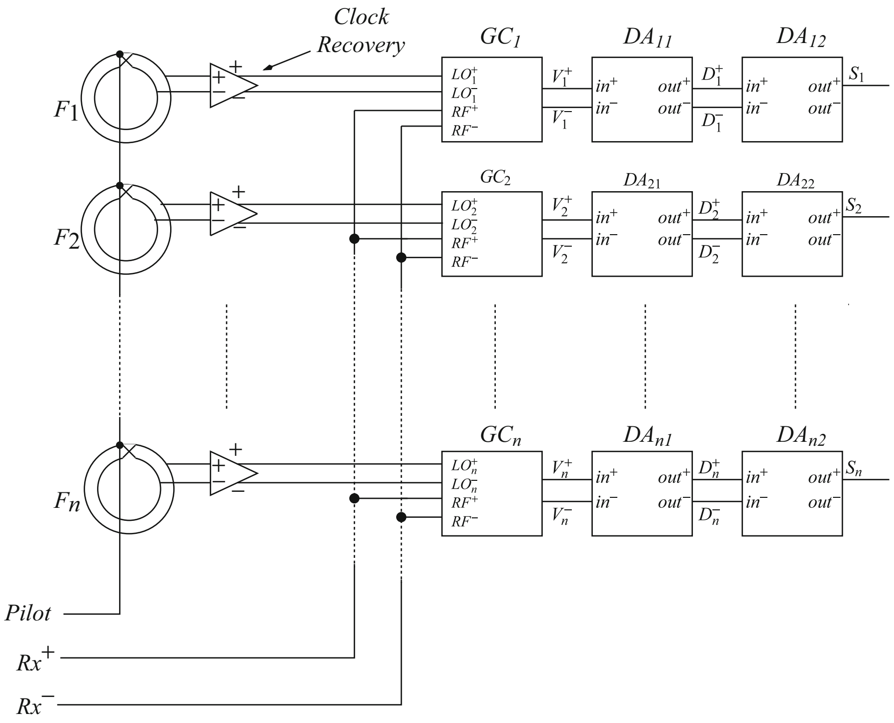


Fig. 4.4 Receiver Circuit

delay of the $pilot$ and the two differential signals Rx^+ and Rx^- are identical while traveling from the transmitter to the receiver.

4.3.3 Receiver

Figure 4.4 shows our proposed receiver circuit which is used to correlate the two differential input signals (Rx^+ and Rx^-) with the basis sinusoidal tones. We use a double balanced Gilbert cell (labeled as GC_i in Fig. 4.4) and two stages of differential amplifiers (labeled as DA_j in Fig. 4.4) to recover the transmitted symbol. The circuit for the Gilbert Cell and the differential amplifiers are shown in Figs. 4.5 and 4.6 respectively. The Gilbert Cell uses a linear, time-varying circuit to perform time domain multiplication. The Gilbert Cell multiplies the time domain input signal RF by a square wave signal at the LO frequency. Let us consider two time domain signals:

$$x(t) = Acos(\omega_{RF}t) \tag{4.1}$$

$$y(t) = Bcos(\omega_{LO}t) \tag{4.2}$$

Fig. 4.5 Gilbert Cell

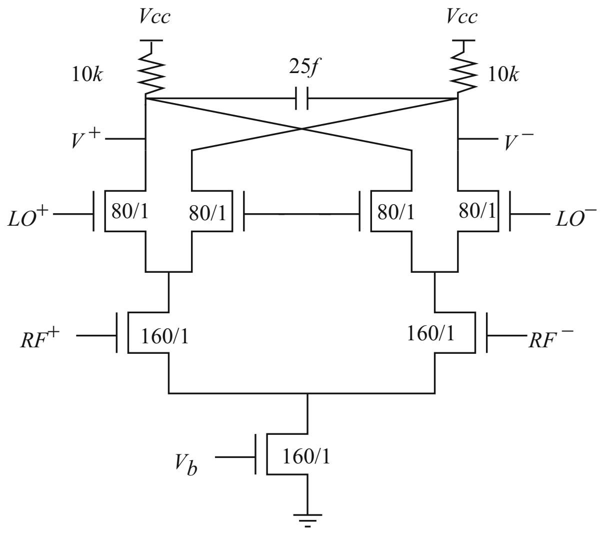
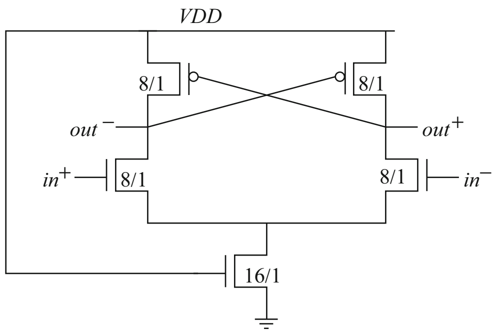


Fig. 4.6 Differential Amplifier



When we multiply these two signals in the time domain, we get the product as

$$x(t) \times y(t) = \frac{AB}{2} cos(\omega_{RF} - \omega_{LO})t + \frac{AB}{2} cos(\omega_{RF} + \omega_{LO})t \tag{4.3}$$

Observe that when $\omega_{RF} = \omega_{LO}$, we get a DC offset along with a high frequency term (corresponding to $2\omega_{RF}$). For $\omega_{RF} \neq \omega_{LO}$, we obtain the sum and difference of the two frequencies whose DC value is zero. The output is subjected to a low-pass filter to remove any high-frequency component. By measuring the DC offset, we can detect if ω_{RF} and ω_{LO} are identical in frequency.

Similar to the transmitter, we have n SWOs which are injection locked to the incoming $pilot$ signal. The 0° and 180° phase of the sinusoids obtained from the SWOs are fed to a clock recovery circuit to extract differential square wave signals, which are used as the local oscillator (LO) inputs for the Gilbert cell. The two received differential input signals (Rx^+ and Rx^-) are used as the RF inputs to the Gilbert cell. For a LO frequency F_i, we get two differential output signals V_i^+ and

V^i- from the Gilbert Cell, which are then fed to two differential amplifier stages to finally recover the transmitted symbol S_i. The Gilbert Cell produces a significant *DC offset* for its two differential outputs. Hence we designed the differential amplifiers at twice the rated power supply. In order to get the model cards for the transistors operating at twice the rated power supply, we modify the gate-oxide thickness of the CMOS. Such devices with thick gate-oxide are available on-chip near the IO pins and are typically rated to operated at a significantly higher power supply, in order to tolerate higher IO voltage spikes without puncturing the gate oxide.

4.4 Experiment

We implemented our design in the 22 nm (PTM 2013) technology, with $VDD = 0.8$ *V*. All simulations were conducted in HSPICE (Inc). RLC parasitics for all the wires were extracted using Raphael (Raphael Interconnect Analysis Tool: User's Guide). Wires are implemented in Metal 9. The width of the wire was taken to be $0.45u$ and the spacing between the adjacent wires was taken to be $0.45u$.

We assume a distance of 2 *cm* between the transmitter and the receiver. For a square die of 1 *cm* × 1 *cm*, the longest distance between the two corners is 2 *cm*, hence this choice. We select 4 sinusoidal tones of frequencies 10 *GHz*, 12 *GHz*, 14 *GHz* and 16 *GHz*. All the SWOs are injection locked to a common SWO of *pilot* frequency 2 *GHz*. Figure 4.7 shows the spice waveforms. The top plot is the common *pilot* signal at frequency F_p at the transmitter. As mentioned earlier, a new set of sinusoidal tones is transmitted every *pilot* cycle. The second plot shows the complimentary signals C_1 and $\overline{C_1}$. We keep $C2 = C3 = C4 = 1$ throughput our simulation. Hence effectively we are transmitting alternate "1" and "0" every *pilot* cycle with the tone F_i. The third plot shows the output of CSA-1 which is the sum of 4 sinusoids. The last two plots show the output of the two differential amplifier stages at the receiver. The output of the second stage differential amplifier is registered every *pilot* cycle to recover the transmitted symbol. We have verified correct operation for all possible combinations of transmitted symbol tones.

We compare our results with a state-of-the-art pulse based data transfer operating at 2 *GHz*. For pulse based data transfer, we assume repeaters every 2 *mm* and a buffer size of 256× minimum size to regenerate the drive strength. We assume 64 parallel wires for the pulse based scheme and 32 parallel wires for our sinusoidal scheme. The above choice ensures identical throughputs for both the schemes. From Table 4.1, we observe that for the same throughput, our sinusoid based scheme consumes 48.7 % less wiring area and 55.3 % less circuit area compared to the pulse based data transfer scheme. Our scheme consumes 25.3 % more power due to high speed operation and active analog components. The time of flight from the transmitter to the receiver for the pulse based signals is 542 *ps* and only 323.2 *ps* for the sinusoid signals. Assuming the system clock to be 2 *GHz* (time period = 500 *ps*), the latency of the pulse based scheme is 2 cycles compared to 1 cycle for our sinusoid based scheme.

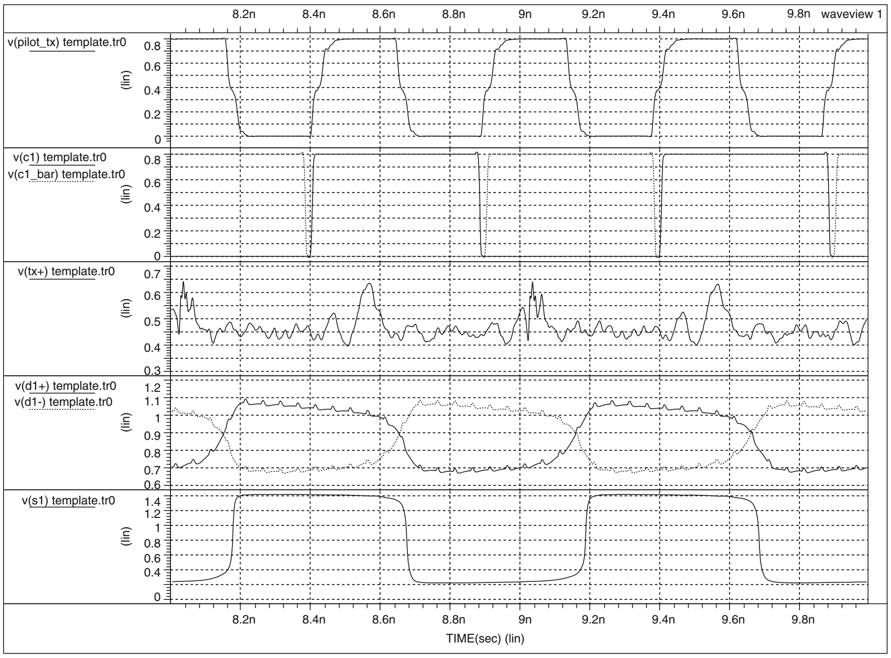


Fig. 4.7 Simulated Signal Waveforms

Table 4.1 Comparison between Pulse-based and Sinusoid Data Transfer

Scheme	Wiring Area	Circuit Area	Power	Latency	Time of Flight	Throughput
Pulse	1.15 mm^2	79.29 μ^2	275.8 mW	2 $cycles$	542 ps	128 $Gbps$
Sinusoid	0.59 mm^2	43.88 μ^2	345.73 mW	1 $cycles$	323.2 ps	128 $Gbps$

4.5 Conclusion

Traditional pulse-based on-chip data transfer achieves a maximum data transfer rate of one bit per wire per clock cycle. In this work, we explore the use of sinusoidal signals of different frequencies as information carriers for on-chip data transfer. The advantage of our method is the ability to superimpose such sinusoids and thereby effectively send multiple logic values along the same wire in a clock cycle. Initial experimental results show that for the same throughput as a traditional scheme, we require 50 % fewer wires. Using more sinusoids would further reduce the number of wires required.

References

James D. Balfour and William J. Dally, “Design tradeoffs for tiled CMP on-chip networks,” in *International Conference on Supercomputing*, 2006, pp. 187-198.

Bradford M. Beckmann and David A. Wood, “Tlc: Transmission line caches,” in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2003, MICRO 36, pp. 43–, IEEE Computer Society.

M.-C. Frank Chang, Eran Socher, Sai-Wang Tam, Jason Cong, and Glenn Reinman, “RF Interconnects for Communications On-chip,” in *Proceedings of the 2008 international symposium on Physical design*, New York, NY, USA, 2008, ISPD ’08, pp. 78–83, ACM.

Victor H. Cordero and Sunil P Khatri, “Clock distribution scheme using coplanar transmission lines,” in *Proceedings of the conference on Design, automation and test in Europe*, New York, NY, USA, 2008, DATE ’08, pp. 985–990, ACM.

Inc Meta-Software, “HSPICE user’s manual,” Campbell, CA.

Nevin Kirman, Meyrem Kirman, Rajeev K. Dokania, Jose F. Martinez, Alyssa B. Apsel, Matthew A. Watkins, and David H. Albonesi, Matthew A. Watkins, and David H. Albonesi, “Leveraging optical technology in future bus-based chip multiprocessors,” in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2006, MICRO 39, pp. 492–503, IEEE Computer Society.

Umit Y. Ogras, Student Member, and Radu Marculescu, “Its a small world after all: NoC performance optimization via long-range link insertion,” *IEEE Trans. Very Large Scale Integration Systems*, vol. 14, pp. 693–706, 2006.

“PTM website, ” http://www.eas.asu.edu/˜ptm (Accessed April 22, 2013).

“Raphael Interconnect Analysis Tool: User’s Guide,”.

Daniel Sanchez, George Michelogiannakis, and Christos Kozyrakis, “An analysis of on-chip interconnection networks for large-scale chip multiprocessors,” *ACM Trans. Archit. Code Optim.*, vol. 7, pp. 4:1–4:28, May 2010.

Chapter 5
Conclusion and Future Work

Abstract In this work, we proposed a high-speed, low power, low jitter chip-wide clock distribution. Using such a clock distribution network, we proposed a fast source-synchronous ring-based NoC responsible for the communication among the cores on a CMP. In the last part of this work, we explored an alternate scheme of achieving high-speed on-chip data transfer using sinusoid signals generated by standing wave oscillators. In this section, we discuss avenues for our future work.

5.1 Future Work on Resonant Clocking

This work addresses the issue of distributing a high-speed, low power, low jitter clock with the aim of serving the NoC and the PEs in a CMP. In our SWO-based clock generation and distribution scheme, we use the wiring resources to control the frequency. Alternate means (like lumped LC tanks) could be explored to achieve this goal. Such an implementation will trade-off the circuit area with the wiring area. Since our rings operate at very high speed, one useful extension to our SWO-based clock distribution scheme would be to run it in the sub-threshold mode of operation. Such a design would provide a significant power improvement while achieving comparable speeds as a state of the art clock distribution scheme.

5.2 Future Work on Fast Network-on-Chip

In this work, we proposed a ring based NoC architecture which is based on source synchronous data transfer over a ring. The source synchronous ring is clocked by a resonant clock which operates significantly faster than the individual processors that are served by the ring. This allows us to significantly reduce the area devoted to the NoC logic and wiring. We have validated the circuit-level design of our proposed NoC components using a 22nm predictive process. Possible enhancements to our NoC fall under the following categories:

- **Handling Wormhole Routing:** In our current routing scheme, each flit of a single packet is routed independently. Wormhole routing allows the transmission of a packet in a pipelined fashion, flit by flit, which improves the NoC performance.

The routing decision is made upon reception of the header flit, and the remaining flits follow the path taken by the header flit. Such an implementation would require a more complex way of allocating virtual channels.

- **Traffic Classes:** In our current design, we support virtual channels. This allows us the flexibility to support traffic classes. Currently, our implementation does not handle different traffic classes. Traffic classes effectively realize separate virtual networks in a time-multiplexed way in the NoC. Cache coherence protocols require the support of traffic classes for sending different message types in order to avoid protocol deadlocks.
- **Quality of Service (QoS):** Our current implementation does not support Quality of Service (QoS). We can also use the virtual channels to support QoS. Since the NoC is a shared resource, it needs to implement support for QoS. The essence of QoS is the ability to offer predictable system behavior (in terms of latency or throughput) to designers, and is crucial for optimum system performance.
- **Path Diversity Routing:** Currently, we use source routing. In this scheme, given a source and destination, the route is always fixed. Such a routing scheme may suffer from congestion under certain adversarial traffic pattern as well as due to certain node/link failures. A useful avenue for future work would be to implement an efficient path diversity aware routing with an aim of evenly distributing the network load.

5.3 Future Work on Fast On-Chip Data Transfer

In this work, we explore the use of sinusoidal signals of different frequencies as information carriers for on-chip data transfer. Initial experimental results show that for the same throughput as a traditional scheme, we require 50% fewer wires. Future work could optimize the design to incorporate more sinusoids, which would further reduce the number of wires required. One option is to use a different *pilot* frequency. Our current scheme uses only two phases (0° and 180°) for each sinusoidal tones. We could also explore the possibility of including quadrature phases and even continuous phases which would enable the transmission of more symbols along a single wire. Currently, we have designed only the transmitter and receiver circuit for on-chip link transfer. A possible extension of this work is to integrate our scheme in an NoC and validate its performance. Finally, the work can be extended to achieve fast off-chip data transfer as well.

Index