TM NoC: A Lightweight XY-Routed Mesh Network-on-Chip Architecture

Theodoros Pontzouktzidis
Department of Computer Science and Engineering
Technical University of Denmark
Denmark
s250239@student.dtu.dk

Dimitrios Vlachos
Department of Computer Science and Engineering
Technical University of Denmark
Denmark
s243192@student.dtu.dk

***Abstract*—Networks-on-Chip (NoCs) have emerged as a promising solution to meet the increasing power, performance, and scalability demands of next-generation Systems-on-Chip (SoCs) [1]. Today, NoCs are widely adopted as a fundamental communication backbone in complex integrated circuits, providing efficient and scalable data exchange between numerous processing elements. In parallel, multiprocessor architectures and platforms have been introduced to extend the applicability of Moore's law, relying heavily on concurrency and synchronization in both software and hardware to enhance system performance and design productivity [2]. In this paper, we focus on the routing component of NoCs, detailing the design and construction of network routers. We further evaluate their performance by measuring key metrics such as latency, throughput, and reliability, using traffic generators to simulate diverse network conditions.**

***Index Terms*—Networks-on-Chip (NoC), routing, switching, scheduling, topologies, interconnection networks, multiprocessor systems, scalability, NoC performance.**

I. Introduction

The increasing complexity and integration density of Systems-on-Chip (SoCs) have established Networks-on-Chip (NoCs) as a fundamental communication infrastructure. NoCs provide scalable, efficient, and structured interconnect solutions that are essential for managing the communication demands of many-core systems. However, as semiconductor technology advances into deep sub-micron regimes, with gate lengths in the 60–90 nm range, the impact of non-scalable wire delays becomes a critical concern [2]. In this context, routing and flow control emerge as key components in maintaining NoC performance, directly influencing latency, throughput, and reliability.

This work focuses exclusively on the routing subsystem of NoCs, deliberately excluding the design and analysis of Network Interface (NI) modules. We implement a network of routers interconnected to facilitate packet exchange, independently managing the flow of data across the chip. Each router is locally connected to traffic generators, which introduce controlled packet loads to the network, allowing for a systematic evaluation of router behavior under varying traffic conditions.

To comprehensively assess the performance of the designed NoC, we analyze critical metrics including throughput, latency, and communication reliability. By employing synthetic traffic patterns generated at each router, we create a controlled environment to stress and measure the routing mechanisms. The results of this study provide valuable insights into the design trade-offs and performance characteristics of routing architectures within NoCs.

II. Background and Related Work

Networks-on-Chip (NoCs) have become essential in modern multiprocessor systems, providing scalable, efficient communication infrastructures to support the growing number of integrated components. Various topologies have been proposed to organize the interconnection network, each with distinct advantages. Common NoC topologies include the mesh, torus, ring and butterfly, as well as irregular interconnection structures [3]. Mesh topologies are favored for their simplicity and scalability, while torus networks reduce average communication distance by wrapping edges. Ring networks offer minimal complexity but suffer from scalability limitations. Butterfly and octagon topologies provide efficient communication patterns for specific traffic types but can be less flexible. Irregular networks allow for customization based on application-specific requirements, often optimizing performance or area.

Router architecture is a critical component of NoC design, influencing overall network performance. Buffers can be placed at the input ports, output ports, or both. Dally and Towles proposed a classic packet-switched router architecture, where data packets are temporarily stored in buffers as they move through the network. This buffering mitigates congestion and helps maintain high throughput under heavy traffic conditions. [4]

Routing protocols determine the path packets take through the network and can be classified by their implementation. Lookup-table-based routing stores predefined paths, providing fast decision-making at the cost of memory usage. Finite State Machine (FSM)-based routing enables more flexible and adaptive behaviors. Routing strategies can further be categorized into unicast routing, which delivers packets to a single destination, and multicast routing, which distributes packets to multiple destinations simultaneously [2].

Switching techniques significantly impact network behavior. In circuit-switched networks, a physical path is reserved before any data transmission occurs, ensuring guaranteed bandwidth but at the cost of setup delay. Conversely, packet-switched

networks transmit packets independently without reserving a dedicated path. Some NoCs, like the Argo network, use time-division-multiplexed (TDM) switching to guarantee real-time communication with predictable latency by statically scheduling packet transmissions [5].

Flow control mechanisms manage how packets progress through the network to prevent buffer overflows and data loss. Buffered flow control allows intermediate routers to store incoming data temporarily. In contrast, bufferless designs drop or deflect packets when congestion occurs. Credit-based flow control keeps track of available buffer space, permitting data transmission only when sufficient credits are available [6]. Handshaking signal-based flow control, such as ACK/NACK protocols, ensures reliability by keeping copies of transmitted data until receipt acknowledgments are confirmed.

The Network Interface (NI) bridges the processing elements and the NoC fabric. It is responsible for packetizing outgoing data and depacketizing incoming data, converting between on-chip communication formats and network protocols [3]. We will not focus on the NI in this paper.

Finally, arbitration techniques are essential when multiple packets contend for the same output port. Common arbitration policies include Round Robin (RR), which grants access in a cyclic order; First Come First Serve (FCFS), which prioritizes packets based on arrival time; and Priority-Based (PB) arbitration, where packets are scheduled based on predefined priority levels [3].

Several works have explored NoC design, covering topology, routing, and flow control. This work focuses on evaluating routing performance under different traffic patterns using custom routers and local traffic generators.

III. Specifications and Design

In this work, we focused exclusively on designing and implementing the router component of a Network-on-Chip (NoC) system, leaving the Network Interface (NI) out of scope. Our design choices aim to balance simplicity, deadlock avoidance [3], and efficiency while enabling scalable and accurate performance evaluation. The scope of this work can be seen in Fig. 1.

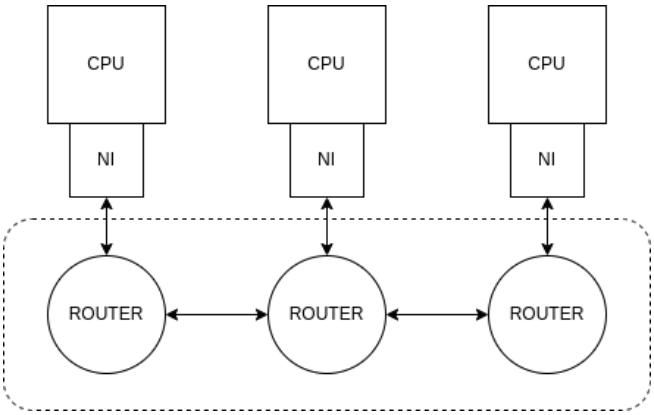

Fig. 1. Overview of the project scope (dashed rectangle).

To achieve a straightforward and deadlock-free design, we adopted a two-dimensional mesh topology with simple (x, y) routing (also known as dimension-order routing). This topology offers natural scalability, minimal hardware complexity, and predictable routing behavior. Dimension-order routing forwards packets first along the x-axis (east or west) until the correct column is reached, and then along the y-axis (north or south) until the packet arrives at its destination [7]. This structured path ensures deadlock freedom without the need for complex resource ordering or virtual channels [3].

A. Router Structure

Each router in our design is equipped with five bidirectional ports: **East**, **West**, **North**, **South**, and **Local**. The local port connects to a traffic generator module, responsible for injecting and consuming packets to and from the NoC. The remaining four directional ports interconnect with neighboring routers, with ports aligned such that, for example, the east output port of one router connects to the west input port of the adjacent router to the east.

A high-level block diagram of the router structure is shown in Fig. 2

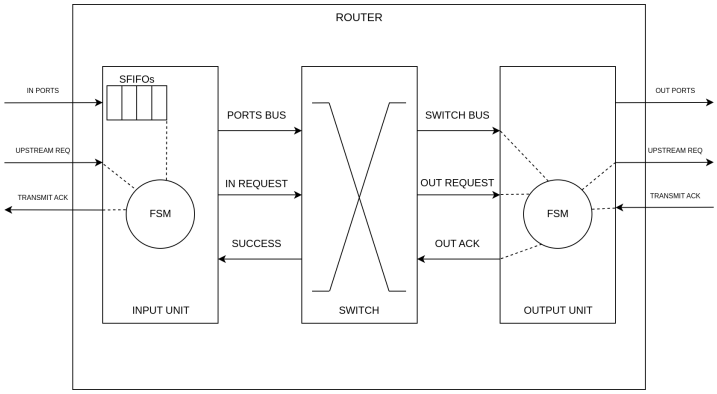

Fig. 2. Overview of a router.

B. Input Unit

Each input port includes a synchronous FIFO buffer that temporarily stores incoming packets. Flow control is achieved through a finite state machine (FSM) that governs the behavior of the input unit. Upon receiving a packet, the FSM decodes the destination address, computes the appropriate output port using (x, y) routing logic, and issues a connection request to the internal switch fabric.

C. Switch and Arbitration

The switch module forms the crossbar connection between input and output ports. When multiple input units request the same output port, a round-robin arbitration mechanism ensures fair and orderly access. Once an input unit is granted access, the packet is transferred from the input FIFO across the switch to the selected output unit.

D. Output Unit

Each output port has an associated finite state machine (FSM) managing the flow of packets toward downstream routers. The output unit listens for incoming requests from the switch and also actively requests transmission permission from the connected downstream router or traffic sink. When permission is granted, the output unit pushes the packet to the next router through direct connection, completing the transfer.

Overall, the router operates in a pipelined manner, enabling concurrent handling of incoming, switching, and outgoing operations to maximize throughput.

IV. Implementation

The proposed router architecture was implemented using Vivado as the primary design and synthesis tool. The implementation follows a modular and pipelined design to achieve efficient routing, flow control, and packet forwarding.

A. Packet Format

In our NoC design, data is transmitted using packets composed of three flits: a head flit, a body flit, and a tail flit. Each flit is 19 bits wide and carries specific information required for routing and data transmission.

- **Head Flit:**
  - 1 bit: Valid bit indicating if the flit is active.
  - 2 bits: Type field identifying the flit as a head, body, or tail.
  - Remaining 16 bits: Destination address, split equally into 8 bits for the X-coordinate and 8 bits for the Y-coordinate.
- **Body Flit:**
  - 1 bit: Valid bit.
  - 2 bits: Type field.
  - Remaining 16 bits: Data payload.
- **Tail Flit:**
  - 1 bit: Valid bit.
  - 2 bits: Type field.
  - Remaining 16 bits: Data payload.

We chose this packet structure (head, body, tail flits of fixed small size) to achieve a lower header-to-payload ratio, minimize bandwidth overhead, and avoid the need for large space-consuming buffers within the router architecture [6] [8].

Head Flit

Valid	Type	Destination (X)	Destination (Y)

Body Flit

Valid	Type	Data

Tail Flit

Valid	Type	Data

Fig. 3. Packet structure composed of head, body, and tail flits (each 19 bits wide).

B. Input Pipeline Stages

The input pipeline manages the reception, buffering, and initial routing of incoming packets. Two key vectors maintain the status of the pipeline:

- **gstate vector:** Tracks the port status with values *PORT_FREE*, *PORT_OCCUPIED*, and *PACKET_RECEIVED*.
- **buffer_status vector:** Tracks the buffer state with values *PACKET_EMPTY*, *PACKET_FILLING*, and *PACKET_RECEIVED*.

The input pipeline operates in two main stages:

- **Buffer Filling Stage:** Initially, the input FIFO is empty. Upon receiving a request from the downstream router, the *gstate* transitions from *PORT_FREE* to *PORT_OCCUPIED*, and the *buffer_status* transitions from *PACKET_EMPTY* to *PACKET_FILLING*. When the tail flit of the packet arrives, the *buffer_status* changes to *PACKET_RECEIVED*, and FIFO writing is halted.
- **Routing Stage:** Once the packet is fully buffered, the input finite state machine (FSM) retrieves the head flit, transitioning from *IDLE* to *ROUTING* state. During routing, the input FSM applies backpressure to the input port to prevent further incoming flits. The FSM computes the destination output port based on XY-routing and requests access to the switch. Upon successful acknowledgment from the switch, the FSM transitions to *ACTIVE* and transmits the flits until the tail flit is sent. After completion, the *gstate* resets to *PORT_FREE*, allowing new packets to be received.

The input finite state machine (FSM) can be seen in Fig. 4

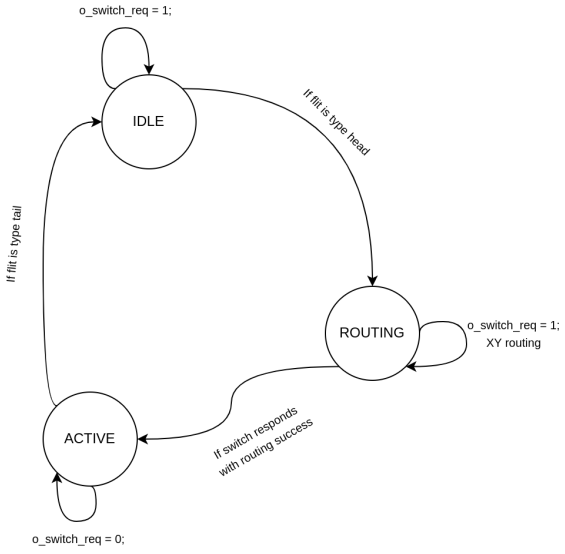

Fig. 4. Input finite state machine (FSM).

C. Switch Architecture

The switch module interconnects input and output ports based on routing decisions. It maintains two sets of port status registers:

- **Port Status:** Each input and output port can be in *P_IDLE* or *P_ACTIVE* states.

When an input port sends a request to the switch, a *request_en* flag is asserted if the target output port is free (*P_IDLE*). A round-robin arbiter grants one input port access and the switch requests to the out port. Upon receiving acknowledgment from the output unit, the switch sets both the source and target port statuses to *P_ACTIVE*. These statuses are maintained until the tail flit is forwarded, at which point they return to *P_IDLE*.

D. Output Unit

The output unit manages communication with downstream routers using a finite state machine (FSM). The output finite state machine (FSM) can be seen in Fig. 5

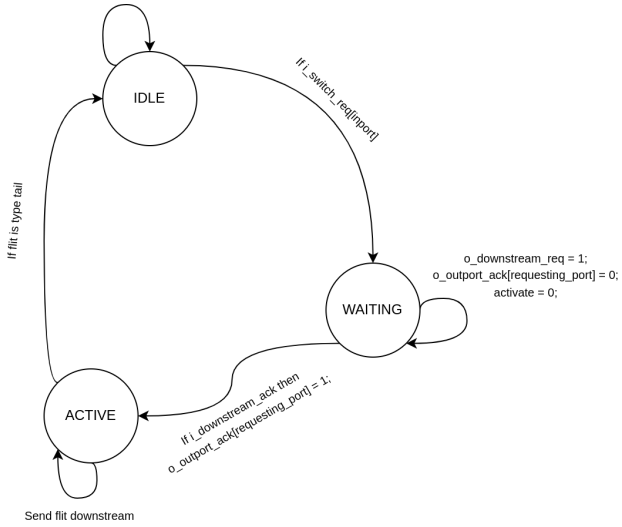

Fig. 5. Output finite state machine (FSM).

Each output port maintains a status that can be:

- *IDLE:* Ready to accept switch requests.
- *WAITING:* Waiting for ack from the downstream input.
- *ACTIVE:* Actively transmitting a packet.

When a switch requests an output port, the output unit transitions from *IDLE* to *WAITING*, sending a request to the connected downstream router. Upon receiving an acknowledgment, the status moves to *ACTIVE*, and the output unit sends an acknowledgment back to the switch, completing the connection. Once the tail flit is transmitted, the status resets to *IDLE*, marking the end of the packet transmission cycle.

E. Traffic Generator

The traffic generator module is responsible for generating and receiving packets within the network. It is designed with two separate FIFOs: one for incoming traffic and one for outgoing traffic. The module operates in two modes, **send** and **receive**, which can occur simultaneously.

In *send mode*, the traffic generator first assembles a packet by filling the output FIFO with the corresponding flits: head, body, and tail. After the packet is ready, it requests access to the downstream port. Once an acknowledgment (ACK) is received, the traffic generator transmits each flit sequentially from the output FIFO onto the network.

In *receive mode*, the traffic generator listens for incoming requests from the network. Upon receiving a request, it sends an ACK back and begins capturing the incoming flits into the input FIFO. Once a complete packet is received (head, body, and tail flits), the traffic generator has the ability to modify the data if necessary. After processing, it can immediately transition into send mode and forward the modified packet to another destination.

F. Key Challenges

During implementation, ensuring synchronized handshaking between input, switch, and output units was critical to maintaining data integrity. The integration of backpressure mechanisms and acknowledgment-based flow control was essential to correctly coordinate packet forwarding in all stages of the pipeline.

V. Results and Evaluation

In this section, we evaluate the performance of our NoC router implementation through latency, throughput, and execution time measurements.

Test Name	Packet Destination	Description
1_lat	Neighbor router (1 hop)	Measure round-trip time to neighbor and back
2_lat	Two routers ahead (2 hops)	Measure round-trip time over two routers
3_lat	Three routers ahead (3 hops)	Measure round-trip time over three routers

TABLE I
Latency Workload Setup

Test Name	Traffic Pattern	Description
base_thr	Single destination	1000 packets to a single neighbor
glob_thr	Global random traffic	1000 packets to random routers from all nodes

TABLE II
Throughput Workload Setup

Workload	Traffic Pattern	Description
edge_w	Edge-targeted traffic	1000 packets to an edge router from each router
URT_w	Uniform Random Traffic (URT)	1000 packets to random destinations from each router
A2A_w	All-to-All traffic	1000 packets from each router to all others
A2A_w (10K)	All-to-All traffic	10000 packets from each router to all others

TABLE III
Execution Time Workload Setup

All experiments were conducted on a simulated 4×4 mesh NoC, running at a 20 ns clock period in Vivado.

The latency evaluation shows a linear increase with the number of hops. In the 1_lat test, sending a packet to the neighboring router and receiving it back took 49 cycles. Extending the distance to two routers ahead in the 2_lat test resulted in a round-trip time of 98 cycles, while in the 3_lat test, the round-trip time reached 147 cycles. These results are presented in Figure 6. The proportional growth in latency with the number of hops confirms that the router maintains consistent per-hop delays without introducing excessive buffering or unpredictable stalls.

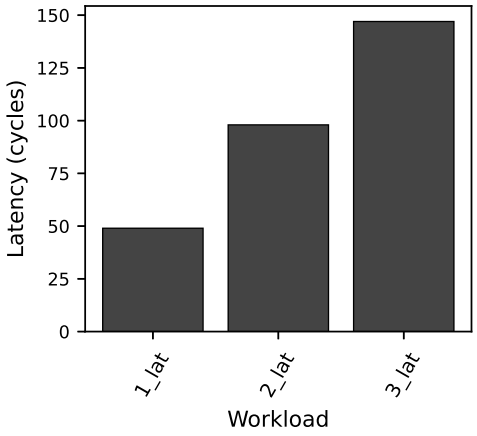

Fig. 6. Measured Round-trip Latencies

Throughput evaluation was performed under two traffic patterns. In the base_thr workload, where 1000 packets were sent to a single router, the system achieved a throughput of 22.24 MB/s over 16019 cycles. In contrast, under the glob_thr workload where all nodes sent 1000 packets to random destinations, the throughput decreased to 12.45 MB/s over 28605 cycles. These results are shown in Figure 7. As expected, throughput is higher when traffic is directed towards a single target and lower under random traffic due to increased contention in the network.

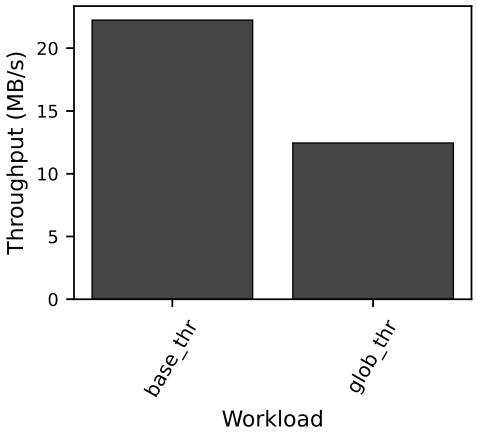

Fig. 7. Measured Throughput under Different Traffic Patterns

Execution time measurements under different workloads provide insights into the network's performance under varying traffic patterns. In the `edge_w` workload, where each router sends 1000 packets to an edge destination, the network required 202014 cycles to complete, reflecting congestion due to localized traffic patterns. This large value stems from the fact that edge destinations tend to concentrate traffic toward specific areas of the mesh, creating congestion and serialization effects as many packets compete for the same output links, ultimately slowing down packet delivery. In the `URT_w` workload, based on Uniform Random Traffic, the execution time dropped significantly to 28605 cycles, indicating improved load distribution. The `A2A_w` workload, where nodes communicate with all others, achieved 24073 cycles, benefiting from balanced traffic. Scaling the `A2A_w` workload to 10000 packets resulted in 244447 cycles, adding more stress to the network with more packets.

Overall, these results, illustrated in Figure 8, confirm that the network performs better when the traffic is spread uniformly or follows structured high-density patterns, while targeted traffic to specific locations (such as edges) introduces bottlenecks that significantly degrade performance.

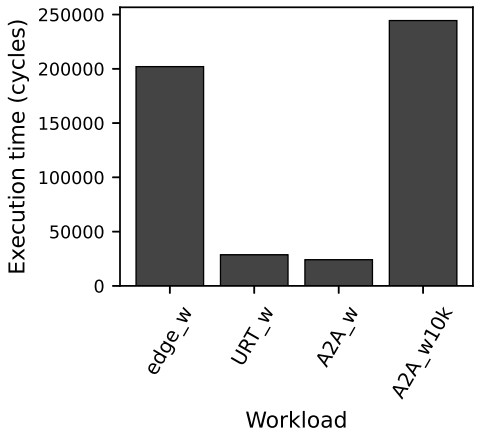

Fig. 8. Measured Execution Times under Various Workloads

VI. Conclusion

In this paper, we presented the design and evaluation of a router for a 2D mesh Network-on-Chip (NoC) architecture. Our work focused solely on the router microarchitecture, excluding the network interface (NI), and employed a deterministic XY routing strategy to ensure simplicity and deadlock avoidance. The router features five bi-directional ports (north, south, east, west, local) and supports packet-based communication using a lightweight flit structure composed of head, body, and tail flits, each 19 bits wide. This structure was chosen to minimize header-to-payload ratio and reduce memory and bandwidth overhead.

The router pipeline was implemented with detailed FSM-based flow control in both input and output stages. It includes mechanisms for buffering, arbitration through round-robin, and synchronized handshaking for flit transfers. A traffic generator

module was developed to emulate diverse communication scenarios, operating in both transmit and receive modes.

We implemented and simulated our design using Xilinx Vivado, targeting a 4x4 NoC at a 20 ns clock period. Evaluation focused on zero-load latency, throughput, and execution time under several synthetic workloads. Results showed expected linear latency increases with hop distance, and throughput performance that scaled with traffic balance. Execution time was minimized under uniform random traffic due to better distribution of load across the network, while workloads targeting edge destinations exhibited higher congestion and longer execution time.

Future work can explore enhancements such as adaptive routing algorithms for dynamic traffic optimization, virtual channels to alleviate head-of-line (HOL) blocking, expanded buffering to accommodate bursty traffic, and the integration of a network interface (NI) module. These improvements would make the router more scalable and robust for larger and more dynamic NoC systems.

REFERENCES

[1] P. K., "Network on-chip and its research challenges," *ICTACT Journal on Microelectronics*, vol. 01, pp. 83–87, 07 2015.

[2] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (noc) architectures & contributions," *Journal of engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[3] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[4] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 2001, pp. 684–689.

[5] E. Kasapaki and J. Sparsø, "The argo noc: Combining tdm and gals," in *2015 European Conference on Circuit Theory and Design (ECCTD)*, 2015, pp. 1–4.

[6] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, p. 1–es, Jun. 2006. [Online]. Available: https://doi.org/10.1145/1132952.1132953

[7] S. D. Chawade, M. A. Gaikwad, and R. M. Patrikar, "Review of xy routing algorithm for network-on-chip architecture," *International Journal of Computer Applications*, vol. 43, no. 21, pp. 975–8887, 2012.

[8] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.