Design of a Network-on-Chip Adaptive Routing Algorithm for Multi-Core Computing

Final Report

Danny Ly

Supervisor: Dr. Philip Watts

Second Assessor: Dr. Ioannis Andreopoulos

March 2013

Abstract

To meet the growing demands of computing and communication, chips with ever increasing transistor density and diminishing size are being continually fabricated. To accommodate this development, the emergence of Multi-Processor System-on-Chip (MPSoC) have become progressively dominant in present day technology. However, current designs incorporate traditional bus-based communication infrastructures which are ultimately not scalable to larger number of processing cores. This concern has been addressed by the Network-on-Chip (NoC), which has become an increasingly popular solution to relieve data bottleneck while also providing power-efficient operation as well as minimised communication contention.

Industrialised chip fabrication processes have progressively revealed an increased number of defects, variations and failure rates as transistors become ever smaller due to Moore's Law. Subsequent to manufacturing, there exists a possibility of permanent faults induced by ageing effects namely, electromigration. Therefore, to provide a plausible solution to the financial burden and prolong the lifetime of chips, a fault tolerant adaptive routing algorithm will be investigated. The project will characterise its latency performance and present its potential suitability to overcome physical faults and thus, maximise production yield.

A scalable SystemVerilog Network-on-Chip utilising 5-port routers has been integrated with a network simulation testbench and has also been modified to facilitate adaptive packet routing decisions based on local network traffic. The design has been configured to operate as a 16-core mesh network to allow a performance evaluation of two distinct types of routing algorithms; deterministic and adaptive. The designs are simulated with Mentor Graphics Modelsim utilising random synthetic traffic of various patterns (e.g uniform, hotspot, streams) and also using trace traffic from real multi-core processors. Synthetic traffic latency results show that the deterministic algorithm outperforms in uniform traffic workloads but, the adaptive algorithm dominates in centralised 30% hotspot and streaming tests. For application-driven workloads both algorithms produce identical latency results.

DECLARATION

I have read and understood the College and Department's statements and guidelines concerning plagiarism.

I declare that all material described in this report is all my own work except where explicitly and individually indicated in the text. This includes ideas described in the text, figures and computer programs.

Name: .....

Signature: .....

Date: .....

Contents

1 Introduction

With the development of multi-core technology continuing [1-2], researchers in the field have exposed critical scalability issues and limiting bandwidth performance of traditional bus-based interconnect architectures and point-to-point links [3-4]. This major flaw has led Network-on-Chip (NoC) interconnects to become an increasingly popular solution. To understand the origins of the NoC, the evolution of processing core technology will be highlighted along with the consequent problems which will lead onto discussion of why NoCs are considered to be a viable solution to accommodate the ever growing communication demands of future multi-core systems.

1.1 The Multi-Core Processor & Multi-Processor System-on-Chip Era

The continued increase in the transistor count due to Moore's Law has enabled increased number of components to be embedded onto a single silicon die allowing increased chip complexity and thus, resulting in the System-on-Chip (SoC) era [5]. Further technological advances coupled with amplified demand for application convergent computing devices such as mobile devices offering real time applications have been responsible for the emergence of multiprocessors and Multi-Processor System-on-Chips (MPSoCs). For these applications uniprocessors fail to deliver satisfactory performance in terms of area and power [6-8].

With present CMOS technology advancing towards the 22nm scale [9], projections from the International Technology Roadmap for Semiconductors (ITRS) indicate a continued trend of multi-core technology; particularly in the networking domain [10]. Therefore, as chip architectures evolve with rising core count it is logical that the underlying interconnect of these systems must adapt in consequence. A background study is presented in Section 2 and explains the need for development of scalable NoCs.

1.2 Problem Background

Integrated circuits are prone to errors both in fabrication and utilisation, this equates to potential profit loss for companies and expensive costs for consumers. It is forecasted that next generation chips will incorporate NoCs [11] therefore, an existing baseline NoC will undergo modification to enable a potential solution to failures.

1.2.1 Fabrication Failures

A continued reduction in chip feature sizes have made the fabrication process increasingly difficult to perfect since process variations have become magnified; leading to greater defect rates and significantly poor yield [12]. The integrated circuit (IC) manufacturing process is an extremely delicate process that involves several stages (oxidation, photolithography, etching etc.) each of which must achieve minimal contamination. Working in the scale of tens of nanometres has lowered the precision of control and can result in imprecise impurity deposition and non-uniform fields in the lithography process, thus leading to transistor malfunction [13]. A report by the ITRS predicts percentage variation in 22nm technology to be three times greater than compared to 45 - 65nm [14] at 18% compared to 5%.

1.2.2 Electromigration

The effect of operation time on chips causes a fundamental ageing phenomenon named electromigration (EM). The EM process gradually diffuses metal ions in wires during current flow and causes them to thin, increasing their local resistance. Eventually, the resistance becomes too large to accommodate and results in EM link failure. With present day chip designs housing a greater number of interconnections per chip, the likelihood of EM failures are becoming more probable [15].

1.2.3 Study of Proposed Solutions

Shamshiri and Cheng [16] implemented additional redundant hardware into designs to cope with failures. It was shown that including extra wires and cores in the initial design allowed repair during manufacture and also post-production. They revealed that with one spare wire per link the percentage of chips ready for market could be increased by up to 10%. Even though redundancy does improve yield, the increased manufacturing costs of extra area overhead can outweigh the yield savings.

A software based solution which uses packet flooding algorithms was investigated by Pirretti and Link [17]. In these algorithms, injected packets are sent to all neighbours with a probability ‘p’ and are dropped with probability ‘1-p’. The value of ‘p’ must be carefully selected to achieve both, near flooding performance and minimise transmission of redundancies. Ultimately, these solutions are probabilistic therefore, packets are not guaranteed to reach the destination. The final verdict for flooding methods is that they are limited to low injection rates due to a large communication overhead.

1.2.4 Solution Choice

There exists a wide range of adaptive algorithms varying from partially to fully adaptive and have the potential for fault tolerance. Authors in the field of NoCs have labelled the routing algorithm as a critical attribute in characterising system performance [18-20]. Lin and Tang [19] developed a bufferless routing algorithm and have shown significant improvements of up to; 10% for average latency, 9% for power consumption and 80% with regards to area overhead when compared to older designs. Therefore, due to the diversity of algorithm types and the possibility of system performance optimisation, the routing algorithm of a NoC will be the focus of this project.

1.3 Project Aim

The aim of this project is to design and implement a potential fault-tolerant unique fully-adaptive routing algorithm characterised by a unique combination of routing, deadlock and livelock techniques into a SystemVerilog baseline mesh NoC. And, interface a testbench that is able to provide real application traffic and randomised synthetic traffic for latency performance characterisation of the algorithm to allow evaluation of its feasibility for fault tolerance.

1.4 Project Objectives

- Interface an existing traffic simulator testbench with a baseline NoC and scale to a 16-core network
- Enhance the synthetic data testbench to enable injection of application recorded trace data into the on-chip network
- Research and implement a unique combination of adaptive routing algorithm characteristics
- Characterise the latency performance of the adaptive routing algorithm using different synthetic and real application-driven traffic patterns
- Characterise the latency performance of a pre-provided deterministic routing algorithm to allow a comparison to be made with the adaptive routing algorithm

1.5 Report Structure

The structure of the remaining report is as follows. Section 2: 'Background' discusses the evolution of interconnections. Section 3: 'Available Resources' overviews the functionality of available resources to be developed. Section 4: 'Justification of Design' summarises research findings and the resulting design choices. Section 5: 'Implementation' presents a summary of design implementation. Section 6: 'Test Method' describes the test plan and method. Section 7: 'Testbench Implementation' outlines the test facility developed. Section 8: 'Results' presents test results from simulations of a 16-core network and evaluations. Section 9: 'Conclusion' summarises the work done, its importance and proposes future improvements. Section 10: 'Appendix' includes partial SystemVerilog and Matlab source code developed and described in the report. The complete set of programs developed has been provided on a CD attached to this report.

2 Background

This section places NoCs into context by detailing the characteristics of current interconnects and leads onto why a new paradigm is essential for sustainability.

2.1 Evolution of On-Chip Interconnections

The early designs of SoCs utilised single shared bus-based architectures due to their simplistic concept [21] and, were adopted into first generation MPSoCs because of their success. However, complications began to arise in large scale MPSoCs regarding; scalability, bandwidth and energy performance prompting the requirement for alternative interconnects.

2.1.1 Single Bus

Traditional bus architectures form broadcast mediums in which a single shared backbone enables intrachip communication, however, there is often a need for dedicated point-to-point links for critical signals which, when scaled can result in an exponential increase of wiring and prolonged design cycles. Therefore, wiring can contribute to excessive power consumption and extremely poor scalability. Furthermore, with multiple processors there are significant increases in latency due to contention [22] (see Fig. 2.1).

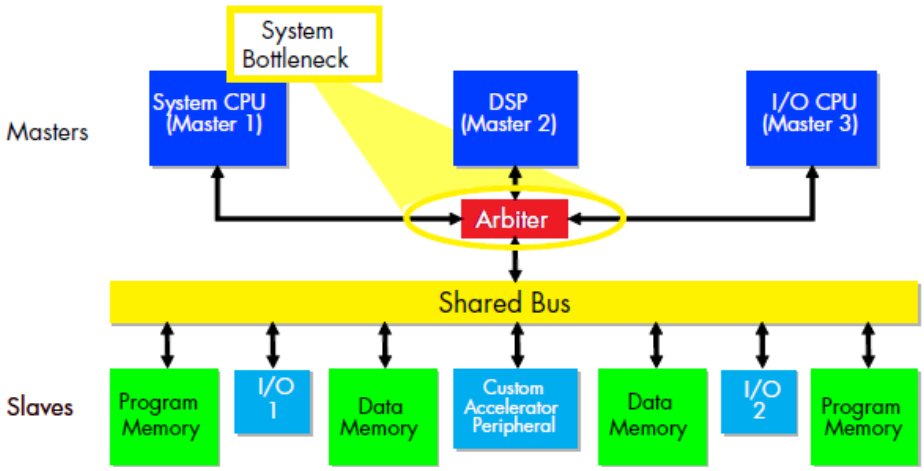


Figure 2.1[1]**:** A bus architecture depicting system bottleneck due to a single point arbitration scheme. These designs are only efficient when accommodating fewer than five cores. [23-24]

2.1.2 Hierarchical/Segmented Bus

To alleviate arbitration bottleneck, hierarchical bus structures were adopted to partition communication domains into layers that operate at different frequencies. This also optimises power usage because buses at the bottom of the hierarchy operate at low frequencies and connect high latency modules [6]. Examples of advanced bus architectures are the ARM AMBA and the IBM CoreConnect which are widely used today [24]. Effective consideration of power is important because increasing number of components means increased capacitive

[1] Altera - Comparing IP Integration Approaches for FPGA Implementation [ONLINE]: http://www.altera.com/literature/wp/wp-01032.pdf [Accessed 12 May 2012]

load [5]. Overall, optimisation is limited since bus architectures are broadcast networks and so data must reach all bus-connected modules [23].

2.1.3 Crossbar

Buses were developed further by incorporating crossbars to form a switch fabric (see Fig 2.2). This overcame the problem of increased non-parallelism by enabling simultaneous transactions to occur through simulation of point-to-point interconnections [6]. Although this facilitates bandwidth improvements, the number of links increase exponentially with rising core count [21] adding to the ever restricting area overhead and power consumption. This translates to the complexity of crossbars scaling as the number of ports squared. Crossbars can also suffer from cross-chip wire delays that limit clock frequencies due to large RC delays and quadratic scaling with distance [25], this eventually forces cross-chip communication latencies up to tens of cycles in sub-hundred nanometre technology [26]. These conclusions show that, although crossbars overcome limitations of the bus design they ultimately have poor scalability.

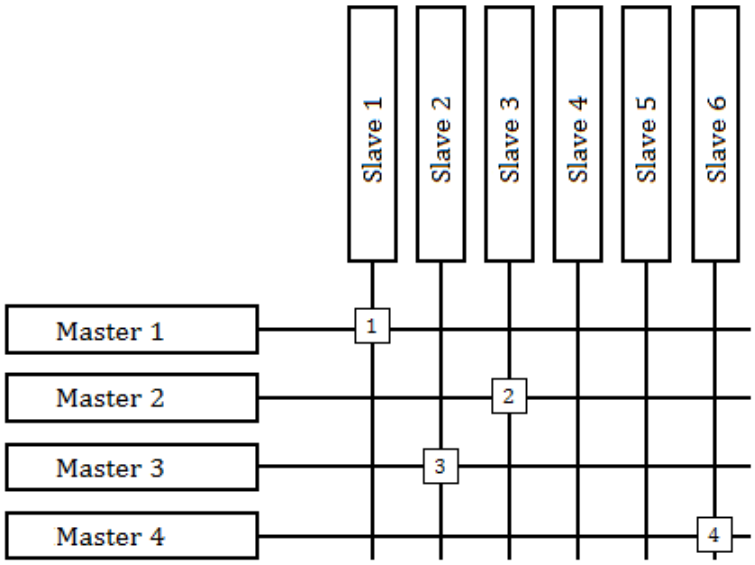


Figure 2.2: Crossbar interconnect displaying simultaneous transactions between master and slave modules.

Buses and crossbars have been the trusted communication providers for multiprocessor technology since their introduction [6] but, with new generations of complex designs appearing there is a requirement for a future sustainable interconnection fabric. A change that has been generally accepted is, the transition towards a shared and segmented global communication architecture since it can provide flexibility and scalability [21]. This structure is described as a data-routing network; a NoC.

2.2 Network-on-Chip

NoCs first began to appear over a decade ago [21] with Hemani (2000) presenting the need for a network structure to effectively accommodate the billion transistor era and, Dally and Towles (2001) introducing a simple design of an on-chip network together with its benefits. Today, NoCs continue to gain recognition such that they are becoming commercialised [27].

Due to the successful scaling of computer networks to meet the demands of the internet, the idea of packet-switching has been adopted to form the basis of on-chip communication [28-29]. NoC architectures contain shared communication resources consisting of multiple routers connected via short universal wires thus, removing unpredictable critical paths and gives predictable electrical parameters from which architects can optimise design layouts [30]. Each router has an explicit connection to a local terminal node (e.g. IP core, cache, peripheral) interfaced by a network interface (NI) module (see Fig 2.3). The client nodes generate data which is packetised by the NI. The packets traverse the network on a hop-by-hop manner determined by the routing scheme until eventually arriving at the destination, where they are ejected to the local node. Packetisation of data provides an effective enforcement of error control and recovery since, errors are contained within packet boundaries and recovery methods can be applied on a packet-by-packet basis [24].

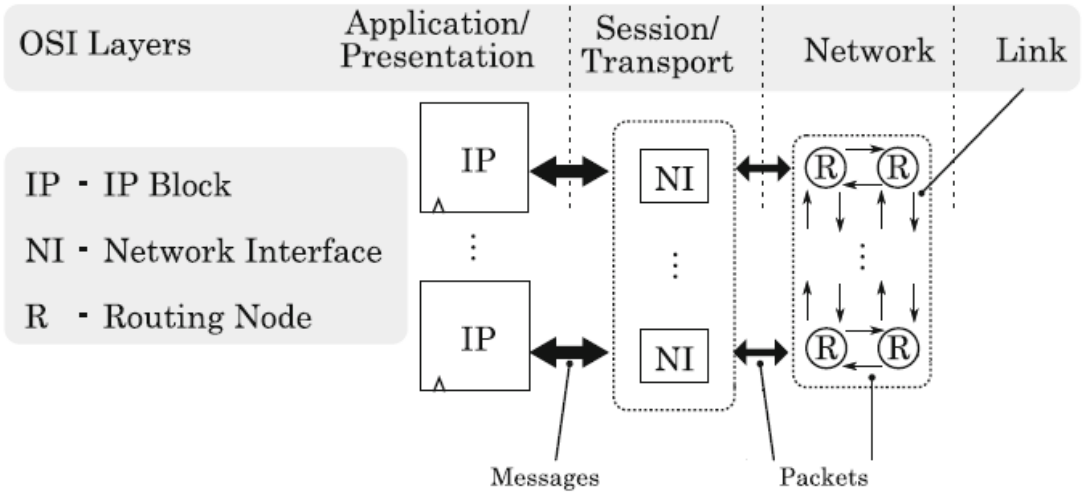


Figure 2.3: Mapping of the computing OSI layers into the NoC domain, adapted from [31]. Local nodes (IP Blocks) communicate with the router network through a network interface.

Various topology configurations are possible and choice depends on the available packaging technology utilised in the realised design, system requirements and module sizes [32]. Due to the repetitive nature of typical NoC topologies such as the city-block [33], design standardisation and reuse is possible which potentially minimises chip design time and time to market [34]. Arteris have shown reductions of up to 40% in the total number of wires required therefore, dramatically reducing design sizes along with faster back-end convergence [35].

3 Available Resources

This section outlines the SystemVerilog modules available for development provided at the start of the project.

3.1 Baseline Network-on-Chip

The baseline NoC consists of 5-port routers that can form either a mesh or torus topology each utilising dimension-ordered XY routing and containing; input buffers, an arbiter and a switch fabric.

3.1.1 Topology

The torus topology will not be considered in this project since it contains across-network links that can introduce significant delays and limit system clock frequencies [36]. In a mesh network routers have up to five connections, one to address the local processing element and four to connect neighbouring routers (see Fig 3.1).

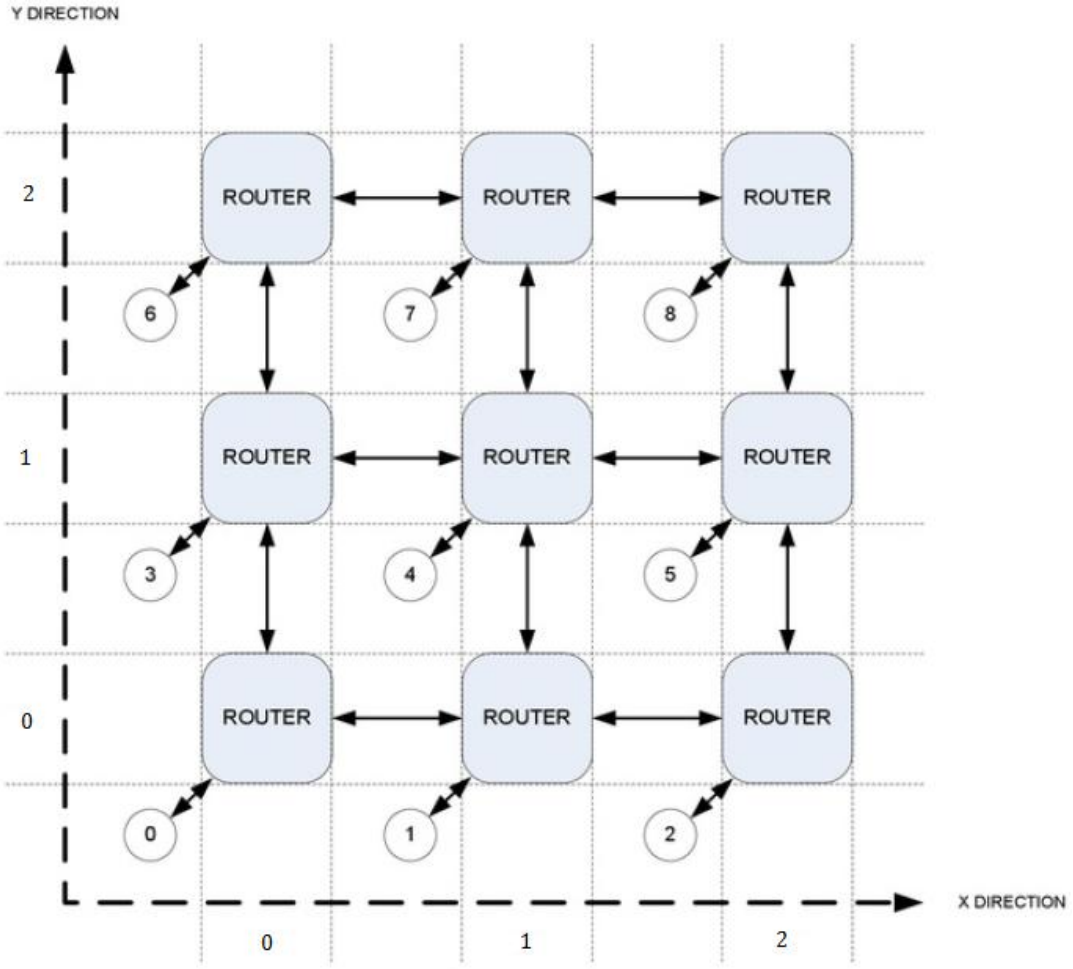


Figure 3.1 [37]**:** A 3x3 core 2D mesh network.

3.1.2 Routing

A deterministic XY dimension-ordered routing (DOR-XY) algorithm, also a shortest path algorithm, is implemented and operates by first routing packets along the x-direction until they reach the x-destination address and then likewise for the y-dimension. This means that packets always traverse the same path between a particular set of source-destination nodes.

3.1.3 Microrouter

Each router provides five fixed length input FIFO buffers to accommodate the local processing core and the neighbouring routers. No output buffering is provided. Write asserted flits are sampled every clock cycle and are written into input buffers. Routing logic produces output port requests for each buffer depending on the routing information contained within the head-of-line flit and are sent to a switch allocator, which configures the switch fabric to connect the input buffers to the output ports. A cyclic iterative priority arbiter employing round-robin scheduling is used for decision making. Fig 3.2 shows the router structure described.

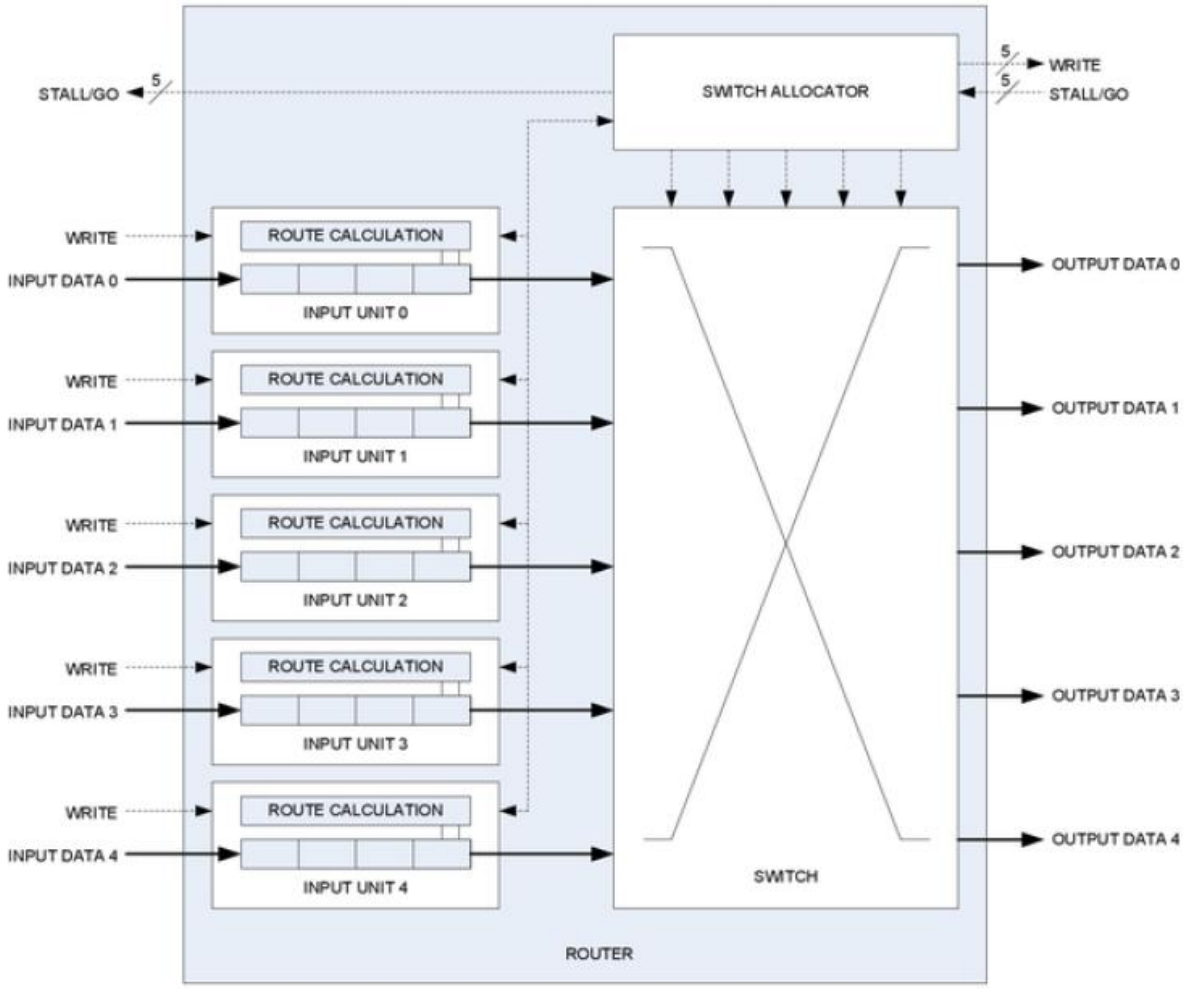


Figure 3.2 [37]**:** Router mircoarchitecture.

3.2 SystemVerilog Testbench

A generic network testbench provided enables random packet generation and measurement of ejected packets (see Fig 3.3). This allows NoC functionality simulation and verification.

3.2.1 Packet Source

For each core in the network a source is instantiated which uses linear feedback shift registers to produce random packet injection times and destinations. Incremental count functionality is also included to show the number of packets injected by each core into the network. Generated packets reside in buffers before injection into the network to allow realistic latency measurements.

3.2.2 Packet Sink

For each core, a sink records the total number of valid packet arrivals and their latency in a cumulative manner. Packet errors are also counted and increments whenever a packet is received at the wrong destination.

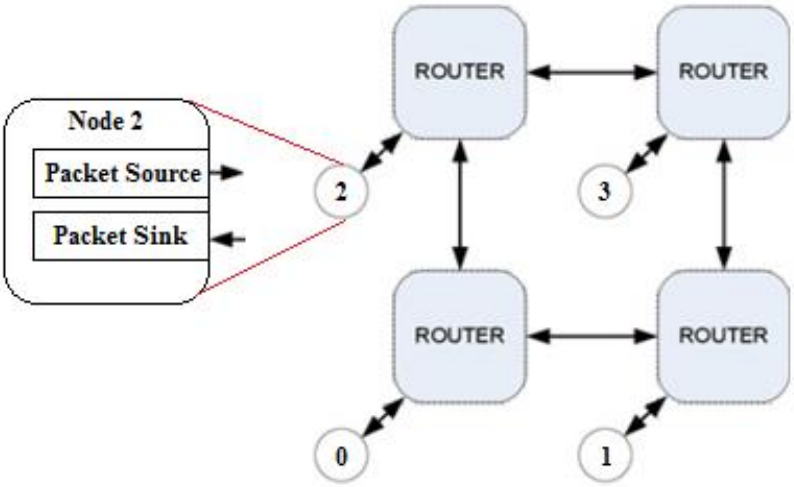


Figure 3.3: An example showing how the testbench interacts with the NoC. Each node contains a method of packet injection and retrieval.

4 Justification of Design

Having placed the NoC into context, the design decisions based on a literature review to address physical failures described in the introduction will now be presented. The deterministic dimension-ordered XY (DOR-XY) routing methodology results in packet loss if faults are present in the network therefore, an adaptive algorithm will be the main focus. The ability to avoid a link malfunction can be considered in the same sense as avoiding a congested path [38] and this proposal has been used for this project.

4.1 Routing Algorithm

DOR-XY routing algorithms are the most commonly used routing algorithms for on-chip networks due to their simplicity [4] and, a literature study has shown many adaptations of this to form adaptive algorithms.

4.1.1 Study of Fully-Adaptive Algorithms

A variant of the deterministic DOR-XY is the 'XYX' which uses DOR-XY and DOR-YX shown by Patooghy and Miremadi [39]. This algorithm works by forcing source nodes to inject redundant copies of every message, with a header flag bit to indicate whether the message is the original or the copy. Original messages are routed using XY while, copies follow YX routing. The results show that overall, the traffic distribution is close to uniform. The problem with this technique is that the addition of redundancy increases the amount of traffic on the network and limits the maximum injection rate for acceptable performance.

Li *et al.* [40] introduced the 'dyXY' algorithm which is both, deadlock and a livelock free. The algorithm, again based on the deterministic XY, routes packets to neighbouring routers towards the destination with the lowest traffic provided that the current routers x or y coordinates are not equal to the destination. Otherwise, normal deterministic DOR-XY routing is utilised. Their results show improved balance in load distribution when compared to the original DOR-XY algorithm.

Another derivative is presented by Nickray *et al.* (Adaptive-XY) [41] where, routers dynamically operate between deterministic and adaptive routing; deterministic when there is little or no congestion otherwise, adaptive. The technique incorporates a 2-bit identifier at each router port excluding the local port, which has unique values to depict different router congestion statuses. The benefit of this algorithm, similarly for dyXY, is that it evenly distributes the load traffic.

4.1.2 Design Choice Justification

Due to the fully adaptive properties and desirable load balancing of 'dyXY' and 'Adaptive-XY', a routing algorithm with a complexity that resides between these will be implemented. The 'dyXY' has a weakness that limits adaptive functionality since it is operates as a shortest path algorithm. Consider the example shown in Fig 4.1, source router (0, 0) compares the west queue of (1, 0) with the south queue of (0, 1) and forwards packets on the minimal

length queue. If router (0, 1) is chosen, the complete route has been determined and hence, the next hops are inevitably (1, 1) and (2, 1). If there is congestion along this path, the algorithm has failed to select an effective path.

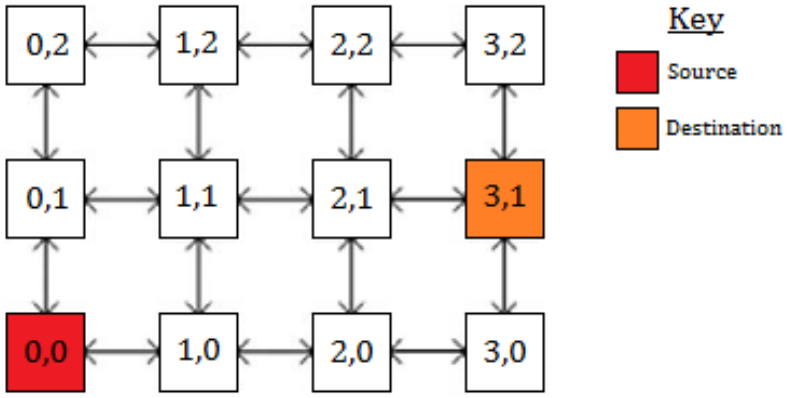


Figure 4.1: 4x3 mesh NoC used to show the drawback of 'dyXY'.

The chosen routing algorithm design has been selected from Dally and Towles [30] and removes the DOR-XY functionality however, retains the next-hop queue length comparison feature. A compact description of the algorithm is; for a certain packet, if a productive output port has a queue length less than some predetermined threshold, it is routed on to the port with the shortest queue length. Otherwise, it is routed to the shortest queue length regardless of being productive or unproductive.

4.1.3 Design Outline

The routing algorithm comprises of two main functions. One relates to gathering of resources and the other is, making a decision based on the information collected. The design to allow resource gathering will be presented first. The basis of this function is to quantify next-hop buffers and to implement this, two critical signals are required (see Fig 4.2).

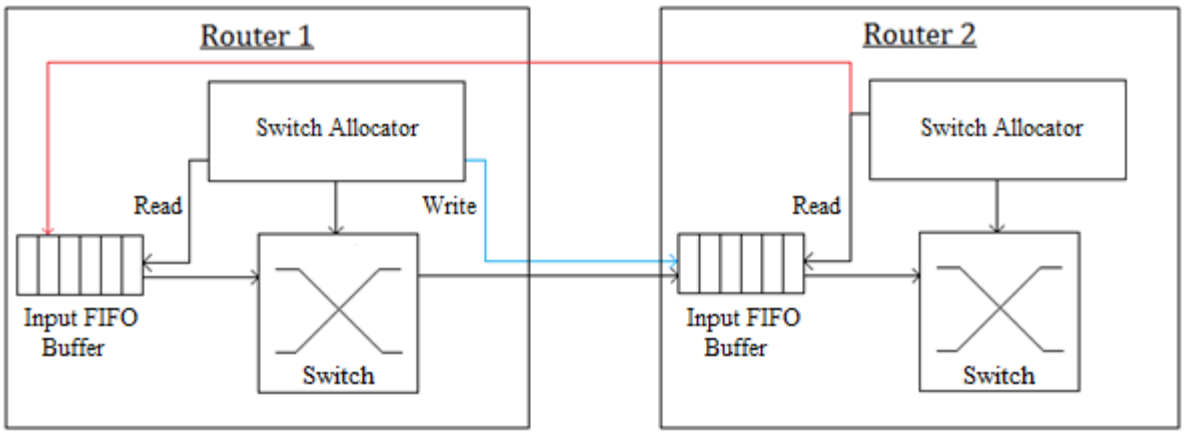


Figure 4.2: Simplified block diagram of two routers with buffer queue count determination signals highlighted. The red line is an additional connection requiring implementation into the baseline network and depends on the network topology.

From the perspective of router 1, the link connects the next-hop switch allocator to a local FIFO; which encapsulates route calculation logic. The buffer count works as follows, from the perspective of router 1, if the local write signal is logic high, the count increments and, if the read enable signal from router 2 goes high, the count decrements. In the example of Fig 4.2, router 1 records the status of its east output port; the west FIFO of router 2. This functionality is repeated for all cardinal directions.

The main routing algorithm utilises the buffer count functionality and will contain conditional statements to check for a productive path less than a threshold. If unavailable, it checks all unproductive cases with unique boundary conditions to find the smallest queue. Packets are routed productively regardless of comparison to the threshold if no single smallest path exists. Boundary conditions have to be considered because, depending on the position of the router, the possible next-hop paths to compare and route along are different. Fig 4.3 shows an example set of boundary condition for the case where the current xy location is less than the destination xy address.

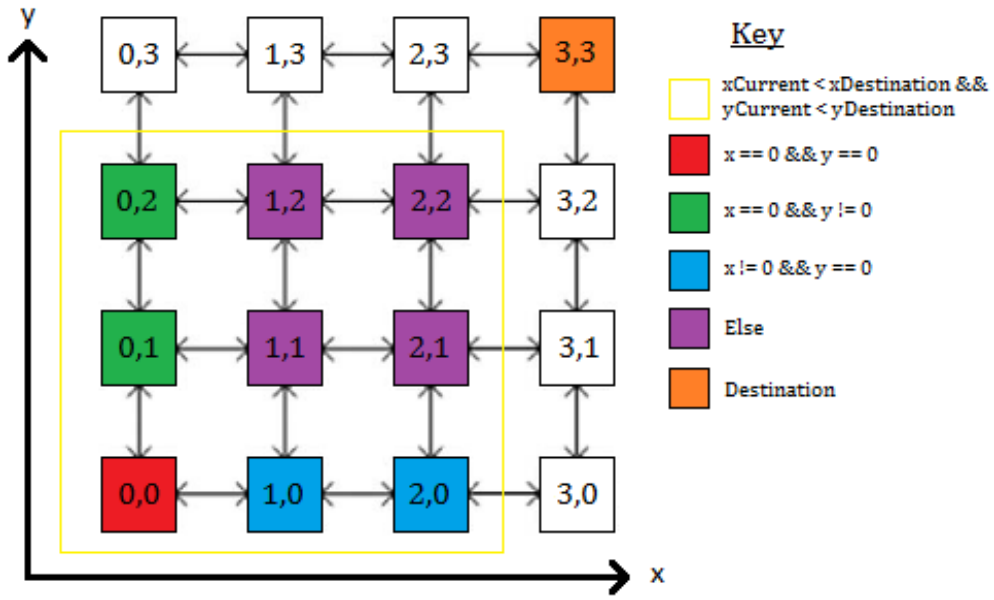


Figure 4.3: Router boundary conditions highlighted for the specific case that the current xy location of a packet is less than the xy destination (enclosed within the yellow box).

The complete routing algorithm is produced by considering all possible packet locations with regards to the destination, and by using the boundary methodology shown in Fig 4.3 for each case. Fig 4.3 describes one particular case; the remaining cases are shown in Appendix 10.1 for completeness. Fig 4.4 presents the algorithm of the specific case in more detail as a flow diagram. The productive paths in this circumstance are to route either north or east.

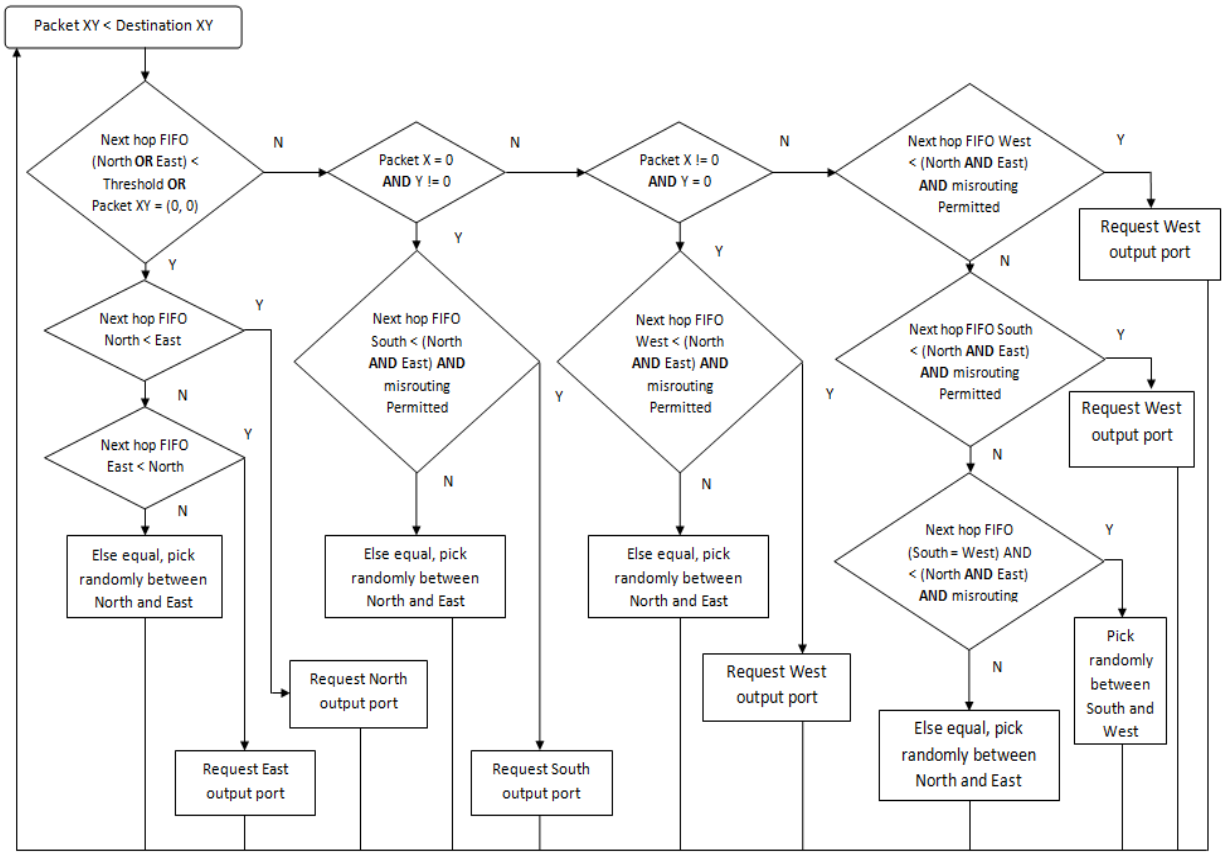


Figure 4.4: Flow diagram of the algorithm for the specific case that the current xy location of a packet is less than the xy destination.

A simplified overview of the complete decision algorithm is presented in Fig. 4.5.

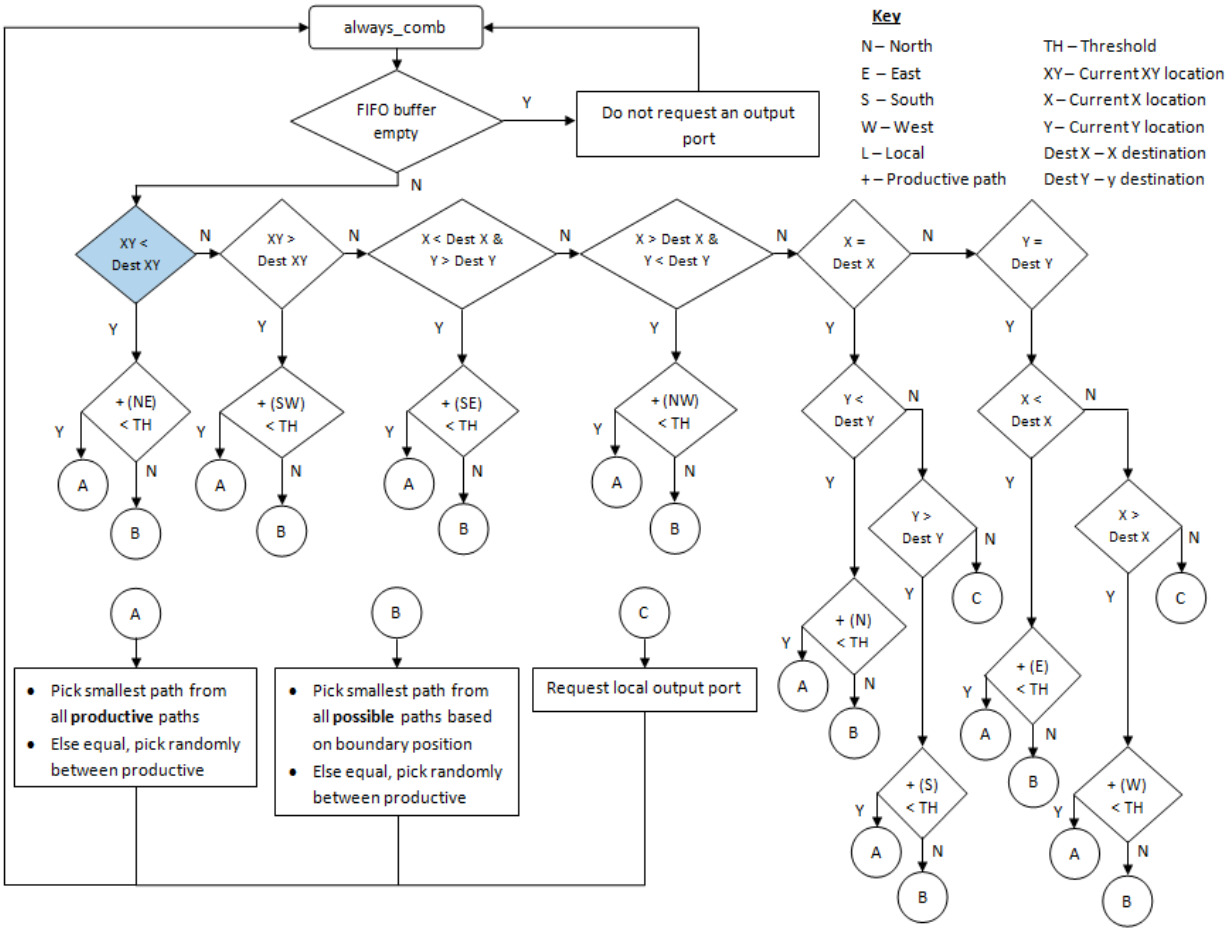


Figure 4.5: Flow diagram of the complete decision part of the adaptive routing algorithm showing all possible cases of current position with regards to destination. The highlighted decision box incorporates the logic of Fig 4.4.

The flow diagram shows that next-hop buffer queues can be equal therefore, the routing algorithm must provide a fair way of choosing between these. To allow this, linear feedback shift registers (LFSR) will be implemented into every router. LFSRs are dividers that produce random noise sequences based on a logical combination of storage elements [42]. The total number of unique sequences is given by:

$$Number\ of\ Sequences = 2^n - 1,\ where\ n\ is\ the\ number\ of\ bits \quad (1)$$

The circuit design shown in Fig 4.6 will be implemented. It is minimal size to minimise area overhead and the feedback path utilises XOR logic for three registers (3-bits) with two taps for maximal-length sequence [43]. The circuit requires an adequate seed to trigger different sequences and for XOR logic, taps with inverted bits are essential. Tapping of different combinations of registers affects only the order in which unique sequences appear therefore, bit zero and one are chosen. The seed ‘001’ has been selected. A single seed used in all routers will mean that they will all generate the same sequence every clock cycle.

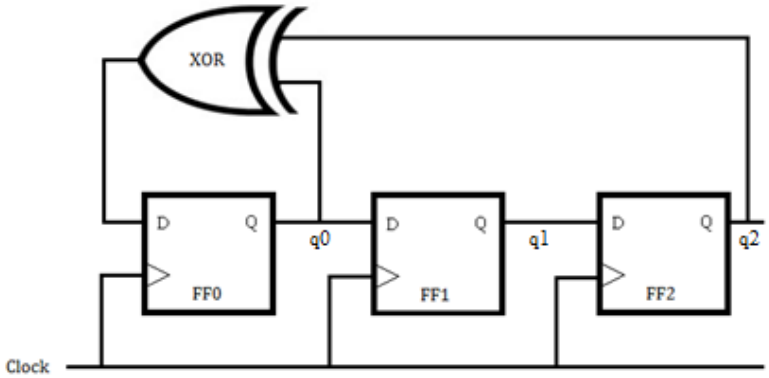


Figure 4.6: A LFSR utilising 3-bits and 2-taps from the MSB and LSB.

The circuit produces pseudorandom sequences and allows test repeatability when the same seed is utilised. This will be done to verify the algorithms functionality.

4.2 Deadlock

Deadlock consideration is vital in adaptive routing due to the large freedom of allowed paths creating the unavoidable outcome of locked cyclic dependencies.

4.2.1 Study of Deadlock Solutions

A widely used deadlock solution is the implementation of virtual channels (VC) [4], which incorporate extra buffers into routers to share the physical links by multiplexing. The idea is that when one VC is blocked, other packets are able to traverse the link through other VCs as escape channels thus, preventing a stalled state.

Dally and Seitz [44] detailed the requirements for deadlock-free routing and showed that by using two VCs, deadlock could be evaded. Their method was to route packets at node numbers smaller than the destination on one channel and those at nodes larger than the destination on another. Duato [45] showed that deadlock-free fully adaptive routing could be achieved by allowing routing sub-functions using two channels, one escape channel; supporting base algorithm and the other; allowing adaptive routing. Gomez *et al.* [46] proposed a similar idea using a two-phase routing scheme where an intermediate node is chosen to allow fault avoidance. From the source to the intermediate node one virtual channel is utilised and from intermediate to destination another is used.

Another technique to prevent deadlock is to prohibit at least one turn in all possible routing cycles [47]. Examples of algorithms that use this are the turn models by Glass and Ni [48] and odd-even model by Chiu [49]. These constitute as partially-adaptive algorithms will not be considered.

4.2.2 Design Choice Justification

Virtual channels will not be used due to their increased hardware costs, complexity and significant routing latency [50]. There exists a trade-off between the number of VCs required and the restrictiveness of the routing algorithm; less VCs are desired but impose a limit on

algorithm diversity. Therefore, a different recovery method involving the idea of deadlock detection timeout will be implemented due to its low logic complexity and minimal hardware requirements when compared to virtual channels [47]. The timeout will be a chosen parameter that depicts the maximum packet idle time for no deadlock; Agrawal and Ravikumar [51] give details on how to optimise the timeout value. Upon reaching this limit the packet will be dropped. It will therefore be the responsibility of higher level networking layers to guarantee correct delivery of the packet to allow reduced complexity in the transaction layer.

4.2.2 Design Outline

Each input buffer will contain logic that increments a count signal on every rising edge of the clock, provided that a packet remains idle at the front of the queue. Fig 4.7 describes the design.

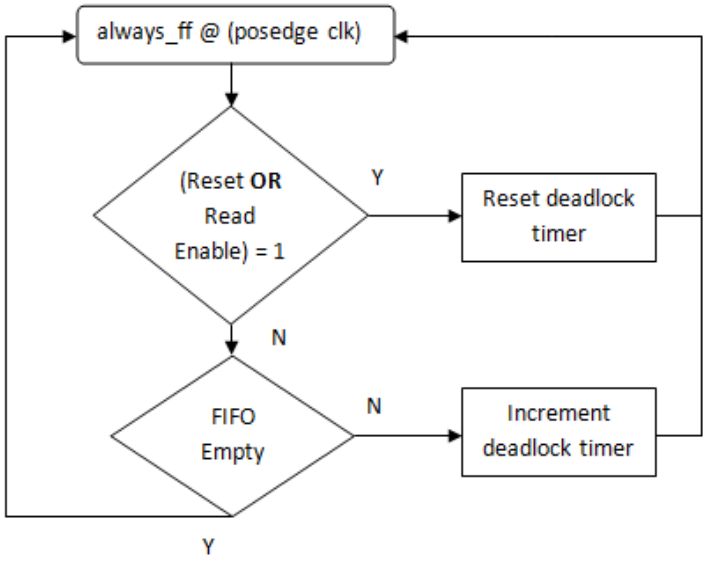


Figure 4.7: Flow diagram of deadlock recovery logic within input buffers.

The count signal will be monitored and a read enable signal for the buffer will trigger once a pre-chosen constant has been exceeded. This drops the packet at the front of the corresponding buffer.

4.3 Livelock

Livelock presents another potential performance limitation of adaptive routing algorithms because, packets face the possibility of being continuously routed around the network and therefore, never arrive at their destination.

4.3.1 Study of Livelock Solutions

Moscibroda and Mutlu [52] offer a solution to bufferless routers by giving packets a timestamp and, upon arbitration the oldest packet has the highest priority and is always forwarded on a productive path. Fallin *et al.* [53] imposed a similar method that promotes packets but removes timestamp comparison assignment and globally priorities individual

packets instead until delivered. A completely different approach adopted from computer networking is described by Chawade *et al.* [54] where a time-to-live (TTL) field is embedded into packets and upon timer expiration the corresponding packet is dropped.

4.3.2 Design Choice Justification

Livelock freedom is not guaranteed if prioritised packets techniques are implemented with networks incorporating wormhole flow control [19] and will therefore be ignored. Following the packet dropping scheme chosen to solve deadlock in Section 4.2.2, the TTL ejection method for livelock would mean potential increase in loss of packets therefore a different method will be adopted. The chosen solution is to impose a limit on the number times unproductive misroutes can occur and to route productively towards the destination once reached.

4.3.3 Design Outline

All packets will contain a misroute constant chosen before simulation. The constant is decremented when an output port request for a packet is unproductive and when the request is grant by the arbiter. Fig 4.8 summarises this functionality into an algorithm.

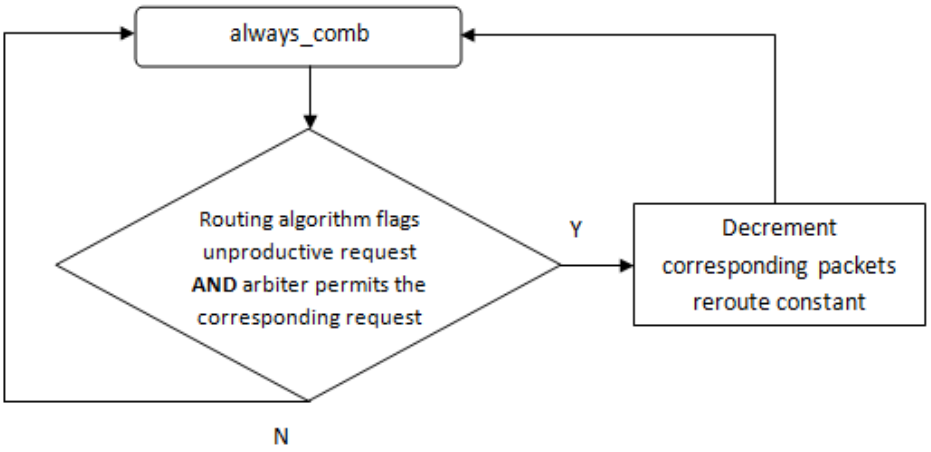


Figure 4.8: Flow diagram of the chosen livelock solution.

Once the misroute constant reaches zero the packet cannot be routed unproductively, this ensures that it will eventually arrive at its intended destination. To enforce this livelock technique extra functionality has to be implemented into the routing algorithm module to flag output port requests as productive or unproductive.

4.4 Testing Facility

Ever since the proposal of NoC appeared, simulation tools to test them have been growing in demand [55] and, are equally important because they allow tuning of design characteristics.

4.4.1 Study of Traffic Simulators

From a literature study of research papers and books, it was discovered that many network simulators have been developed that allow network simulation as well as generation of

different traffic patterns [56]. However, the traffic from these simulators are models of computer communication networks and are not dedicated for on-chip applications. Due to the scarcity of NoC simulators, these have been painstakingly adapted for use [55].

Sun *et al.* [57] utilised Ns-2, a commercial simulator to simulate their NoC and test their design. The Ns-2 testing facility is capable of producing traffic that is; random, that appears in bursts or is favourable towards a certain destination. Their results are described as theoretically reasonable and so the simulator enabled tests that were worthwhile. They also concluded that modelling synthetic traffic to imitate real traffic is a difficult problem to solve.

Another simulator, OPNET, was used by Xu and Wolf [58] and Bolotin *et al.* [59] in place of Ns-2 because it operates with improved speed and stability. The simulator allowed Xu and Wolf to show that their application-specific NoC (ASNoC) design had 39% less power, 59% less silicon overhead and 74% less metal area than compared to a 2D mesh NoC.

4.4.2 Design Choice Justification

Synthetic traffic generators are widely used for analysis of network designs [60]. However, strong dependence on these imposes the problem that the designs will not be able to reflect well when they face unpredictable data from real applications [10]. This means complete characterisation is not achieved and a biased result is portrayed. To offer a solution to this, a traffic generator capable of injecting packet data from network trace files produced by real-life applications will be implemented into the testbench described in Section 3.2 to complement the synthetic random traffic simulator.

4.4.3 Design Outline

Semicolon delimited traffic data will be formatted using a file handling script written in Matlab to produce a text file for each core and, will contain times of packet injection formatted such that they can be read in by the SystemVerilog testbench. Fig 4.9.1 shows the processing steps required to inject trace data into the network.

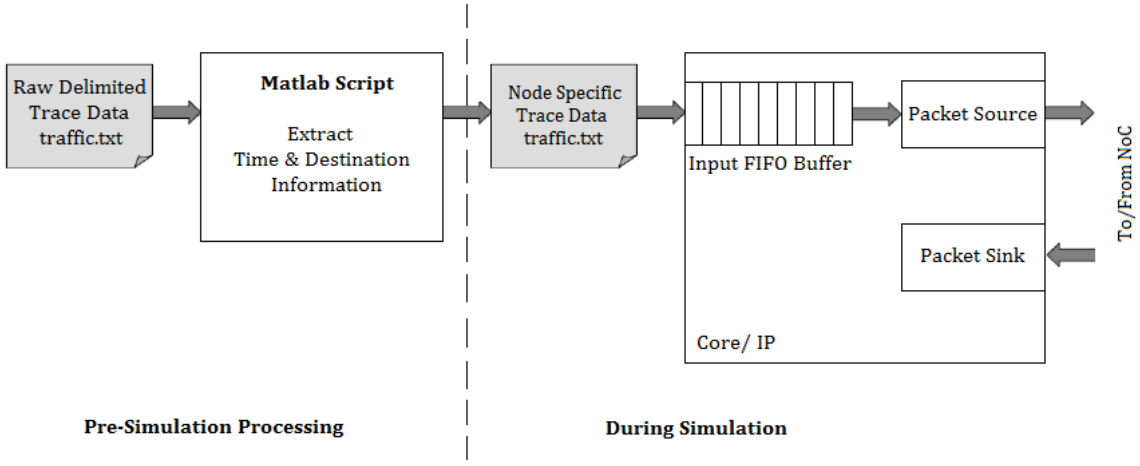
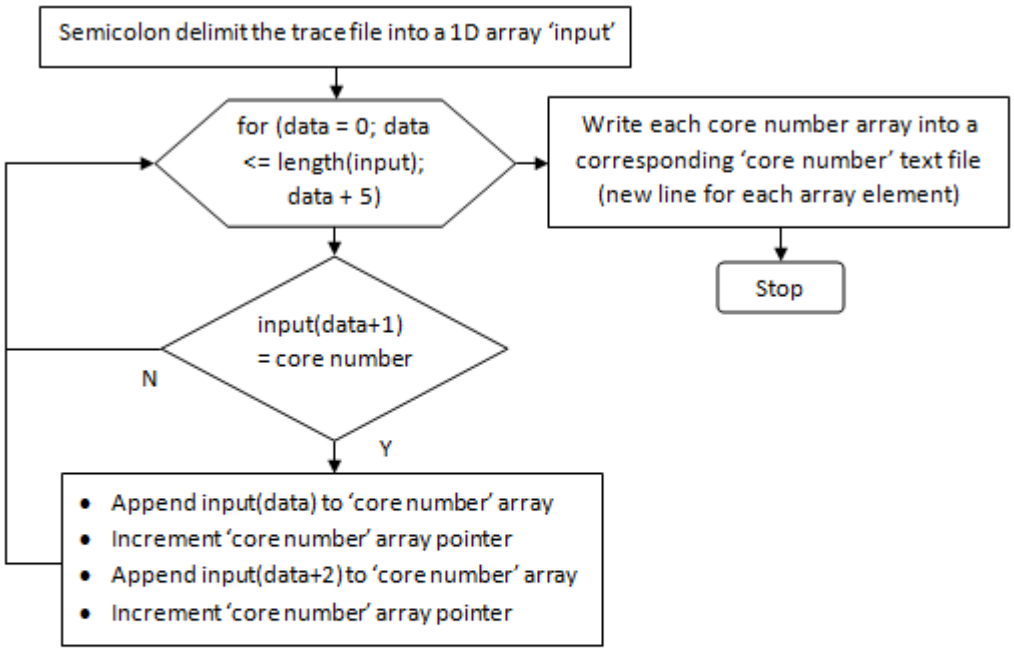


Figure 4.9.1: A block diagram showing how a raw trace data file connects to the testbench.

File handling functionality will be added to the packet source module (Section 3.2.1) to allow the correct text files to be read and generate packets with the corresponding time and destination information. The Matlab script will only require a single run and its output can be reused for simulations. An outline of the algorithm to be implemented in Matlab is shown in Fig 4.9.2 with an example trace file snippet for clarity.




```
Time; Source; Destination; Size(in bytes); Message Type
0;    0;   15; 8;   REQ_C
571; 15; 0;   72; RES_D
578; 0;   15; 8;   UNB_C
580; 8;   12; 8;   REQ_C
```

Figure 4.9.2: Flow diagram of the formatting algorithm and a trace file extract.

The algorithm loops through every packet injection, checks the source core number and stores time and destination into a corresponding core array to be written to a text file. In some cases the trace data will have source core number errors, these are omitted.

5 Implementation

This section presents an overview of the implementation of design choices using a hierarchy of the system and module descriptions for the SystemVerilog design. All modules were modified however, only the modules exhibiting major alterations to realise the design choices in Section 4 will be described.

5.1 Design Hierarchy

Fig 5.1 presents the complete design hierarchy.

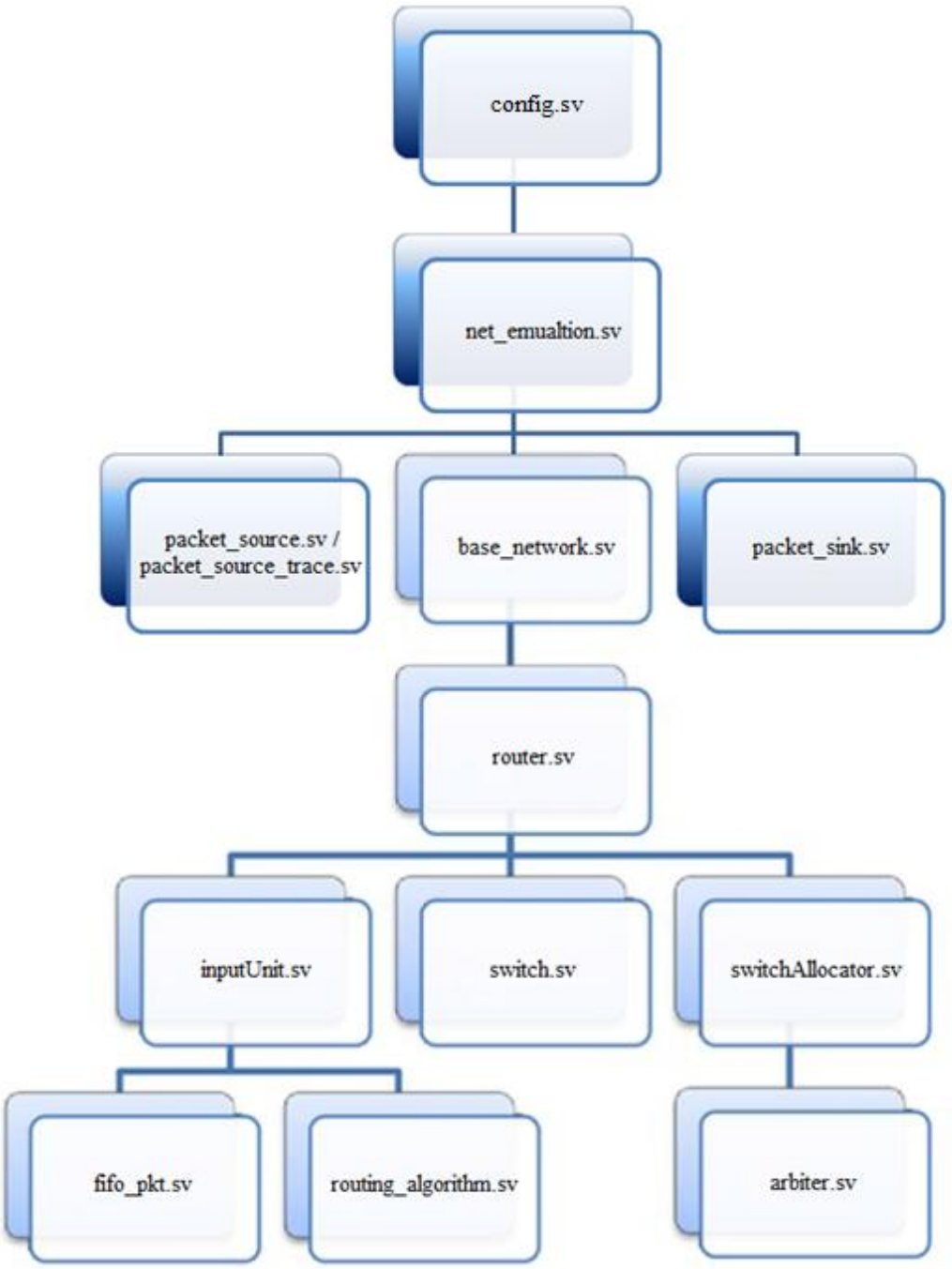


Figure 5.1: Hierarchy of the design under test and testbench in darker shading.

5.2 Router

The router instantiates five ‘inputUnits.sv’ modules, a ‘switchAllocator.sv’ module and a ‘switch.sv’ module. Combinational logic appears in series; before instantiation of ‘inputUnit.sv’ to enable packet hop count measurements for verification purposes and also, after the ‘switch.sv’ module to update the packet misroute constant used to prevent livelock as described in Section 4.3.3.

5.2.1 Module

The following diagram shows the parameters, inputs and outputs of ‘router.sv’.

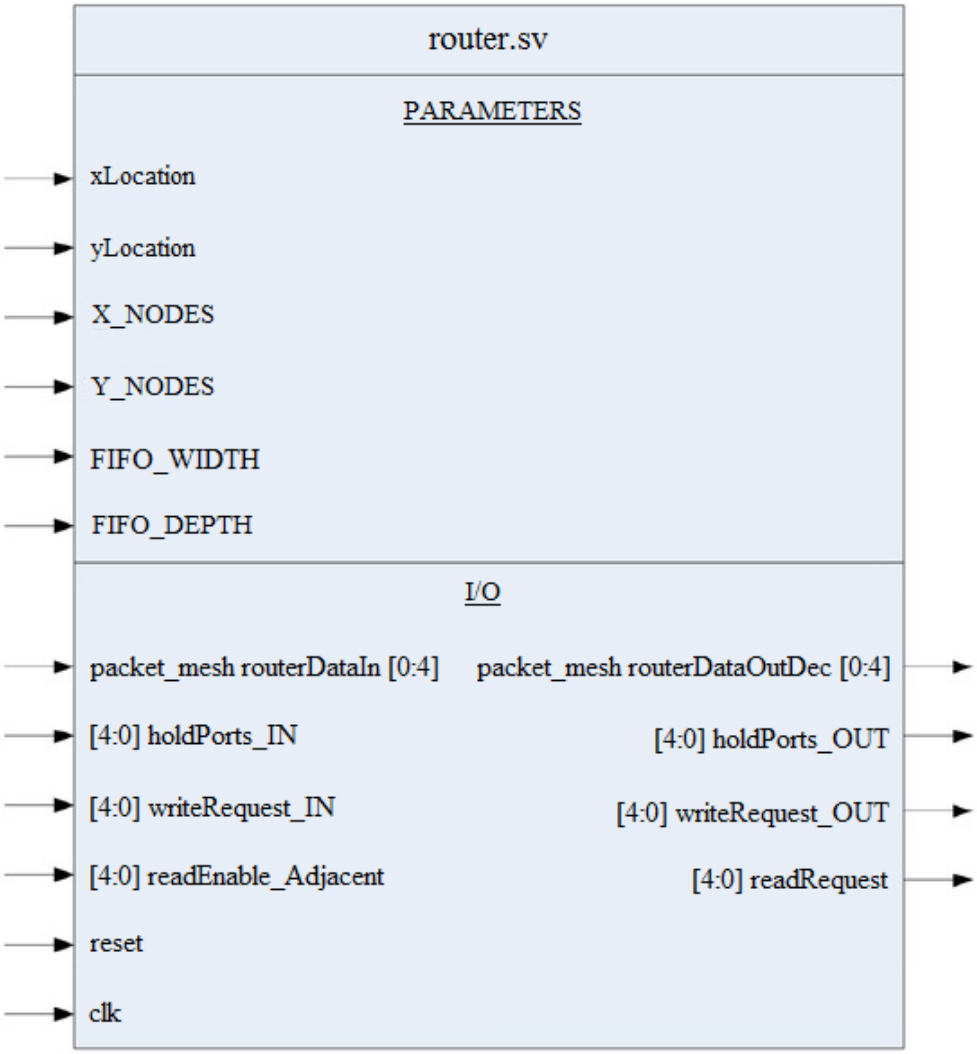


Figure 5.2: Model of the ‘router.sv’ module.

5.2.2 Parameters

The following table describes the parameters of ‘router.sv’.

Name	Description
xLocation	Current router x-location.
yLocation	Current router y-location.
X_NODES	Number of network nodes in the x-direction.
Y_NODES	Number of network nodes in the y-direction.
FIFO_WIDTH	Packet size.
FIFO_DEPTH	Input buffer size.

Table 5.2.1: Parameters of the ‘router.sv’ module.

5.2.3 Inputs

The following table describes the inputs of ‘router.sv’ (see Appendix 10.2 for details on packet_mesh size.

Name	Size	Description
routerDataIn	packet_mesh * [0:4]	Five packet-sized input port connections to neighbouring routers (NESW) and local core.
holdPorts_IN	[4:0]	Each bit corresponds to a neighbouring router or local core. Logic high indicates next-hop buffer is full.
writeRequest_IN	[4:0]	Logic high indicates ‘routerDataIn’ is valid and writes to the back of the FIFO queue next clock cycle.
readEnable_Adjacent	[4:0]	5-bit word from next-hop routers switch allocator corresponding to the output ports NESWL (LSB is north). Individual bits at logic high indicate a packet is read from the corresponding FIFO of the next-hop router.
reset	1	Logic high nullifies resources.
clk	1	Clock.

Table 5.2.2: Inputs of the ‘router.sv’ module.

5.2.4 Outputs

The following table describes the outputs of ‘router.sv’.

Name	Size	Description
routerDataOutDec	packet_mesh * [0:4]	Five packet-sized output port connections to neighbouring routers (NESW) and local core. Packet reroute constants are decremented before exiting if routed unproductively.
holdPorts_OUT	[4:0]	Each bit corresponds to a neighbouring router or local core. Logic high indicates to the next-hop router that this current buffer is full.
writeRequest_OUT	[4:0]	Logic high indicates ‘routerDataOutDec’ is valid to the next-hop router.
readRequest	[4:0]	5-bit word from the local switch allocator corresponding to the NESWL buffers. Individual bits at logic high indicate a packet is read from the corresponding FIFO buffer.

Table 5.2.3: Outputs of the ‘router.sv’ module.

5.3 Input Unit

This module initially contained buffer logic and the deterministic DOR-XY routing algorithm. It is now decoupled into a hierarchy which instantiates these functions as modules to produce a structure that is easier to comprehend and develop.

5.3.1 Module

The following diagram shows the parameters, inputs and outputs of ‘inputUnit.sv’.

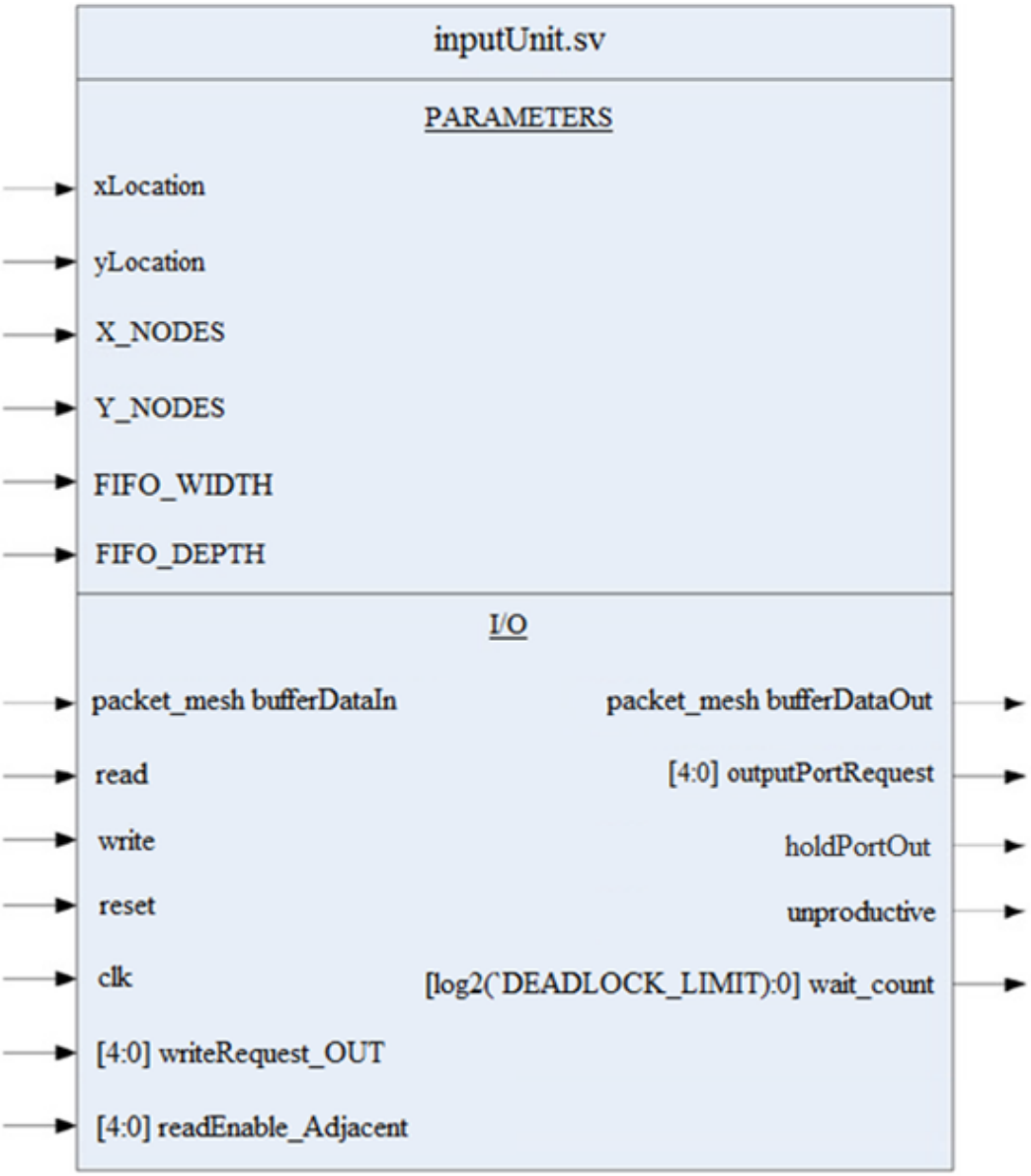


Figure 5.3: Model of the ‘inputUnit.sv’ module.

5.3.2 Parameters

The following table describes the parameters of ‘inputUnit.sv’.

Name	Description
xLocation	Current router x-location.
yLocation	Current router y-location.
X_NODES	Number of network nodes in the x-direction.
Y_NODES	Number of network nodes in the y-direction.
FIFO_WIDTH	Packet size.
FIFO_DEPTH	Input buffer size.

Table 5.3.1: Parameters of the ‘inputUnit.sv’ module.

5.3.3 Inputs

The following table describes the inputs of ‘inputUnit.sv’.

Name	Size	Description
bufferDataIn	packet_mesh	Packet from a next-hop router.
read	1	Logic high indicates head-of-line FIFO packet is granted use of the switch and is sampled next clock cycle.
write	1	Logic high indicates ‘bufferDataIn’ is valid and writes to the back of the FIFO queue next clock cycle.
reset	1	Logic high nullifies resources.
clk	1	Clock.
writeRequest_OUT	[4:0]	5-bit word granted write from the switch allocator corresponding to the output ports NESWL (LSB is north). Logic high indicates a packet is sent to the corresponding output port.
readEnable_Adjacent	[4:0]	5-bit word from next-hop routers switch allocator corresponding to the output ports NESWL (LSB is north). Logic high indicates a packet is read from the corresponding FIFO of the next-hop router.

Table 5.3.2: Inputs of the ‘inputUnit.sv’ module.

5.3.4 Outputs

The following table describes the outputs of ‘inputUnit.sv’.

Name	Size	Description
bufferDataOut	packet_mesh	Head-of-line FIFO packet.
outputPortRequest	[4:0]	One-hot, 5-bit word router port requests from the routing algorithm.
holdPortOut	1	Flag bit, logic high indicates that the input buffer is full.
unproductive	1	Flag bit, logic high indicates that the output port request is unproductive.
wait_count	[log2(`DEADLOCK_LIMIT):0]	Signal indicating how many clock cycles have passed without a buffer read.

Table 5.3.3: Outputs of the ‘inputUnit.sv’ module.

5.4 Routing Algorithm

The algorithm described in Section 4.1.3 was implemented using SystemVerilog hardware description language. The algorithm is instantiated by each input unit of a router and produces an output port request together with an unproductive flag signal for livelock control for the head-of-line FIFO buffer packets. Conditional logic linking to the testbench allows the user to select which of the two algorithms the routers run during simulation. To enable buffer count functionality, the buffer read signals of adjacent routers link to this module. These connections have been implemented by adding extra logic into the 'base_network.sv' module which considers topology boundary conditions.

5.4.1 Module

The following diagram shows the parameters, inputs and outputs of 'routing_algorithm.sv'.

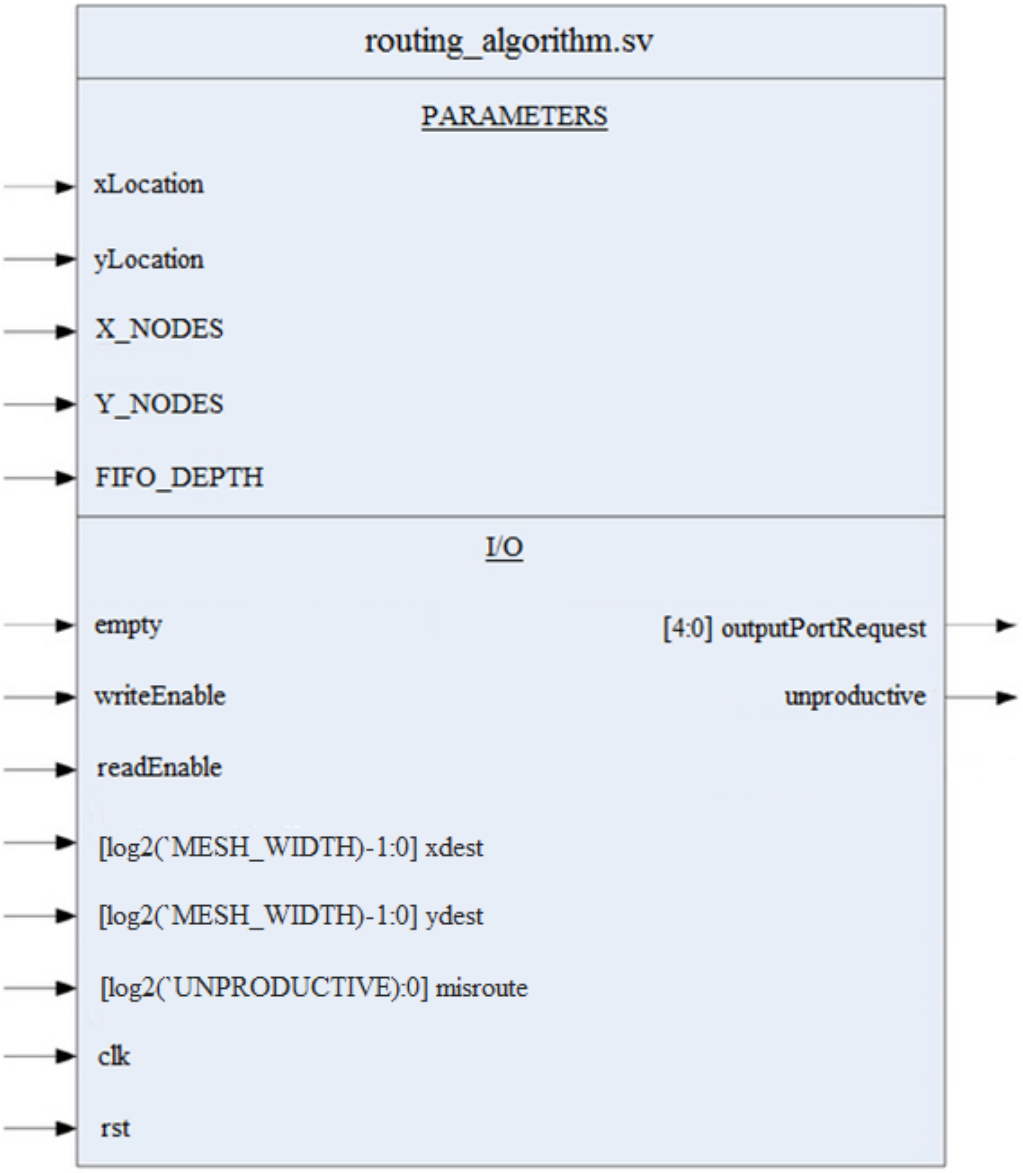


Figure 5.4: Model of the 'routing_algorithm.sv' module.

5.4.2 Parameters

The following table describes the parameters of ‘routing_algorithm.sv’.

Name	Description
xLocation	Current router x-location.
yLocation	Current router y-location.
X_NODES	Number of network nodes in the x-direction.
Y_NODES	Number of network nodes in the y-direction.
FIFO_DEPTH	Input buffer size.

Table 5.4.1: Parameters of the ‘routing_algorithm.sv’ module.

5.4.3 Inputs

The following table describes the inputs of ‘routing_algorithm.sv’.

Name	Size	Description
empty	1	Logic high indicates empty FIFO.
xdest	[log2(`MESH_WIDTH)-1:0]	Head-of-line FIFO packets x-location.
ydest	[log2(`MESH_WIDTH)-1:0]	Head-of-line FIFO packets y-location.
misroute	[log2(`UNPRODUCTIVE):0]	Head-of-line FIFO packets reroute constant.
writeEnable	[4:0]	5-bit word granted write from the switch allocator corresponding to the output ports NESWL (LSB is north). Logic high indicates a packet is sent to the corresponding output port.
readEnable	[4:0]	5-bit word from next-hop routers switch allocator corresponding to the output ports NEWSL (LSB is north). Logic high indicates a packet is read from the corresponding FIFO of the next-hop router. MSB always zero as routing algorithm does not consider local node traffic.
clk	1	Clock.
rst	1	Logic high nullifies resources.

Table 5.4.2: Inputs of the ‘routing_algorithm.sv’ module.

5.4.4 Outputs

The following table describes the outputs of ‘routing_algorithm.sv’.

Name	Size	Description
outputPort Request	[4:0]	One-hot, 5-bit word corresponding to the output ports NESWL (LSB is north). Logic high indicates head-of-line FIFO packet desires an output port.
unproductive	1	Flag bit, logic high indicates that the output port request is unproductive.

Table 5.4.3: Outputs of the ‘routing_algorithm.sv’ module.

5.5 Buffer

A FIFO buffer of parameterised size is implemented with the deadlock recovery design in Section 4.2.3. The deadlock monitoring signal connects to the 'switch_allocator.sv' module which contains logic to set corresponding buffer read enable signals high when the deadlock limit is exceeded.

5.5.1 Module

The following diagram shows the parameters, inputs and outputs of 'fifo_pkt.sv'.

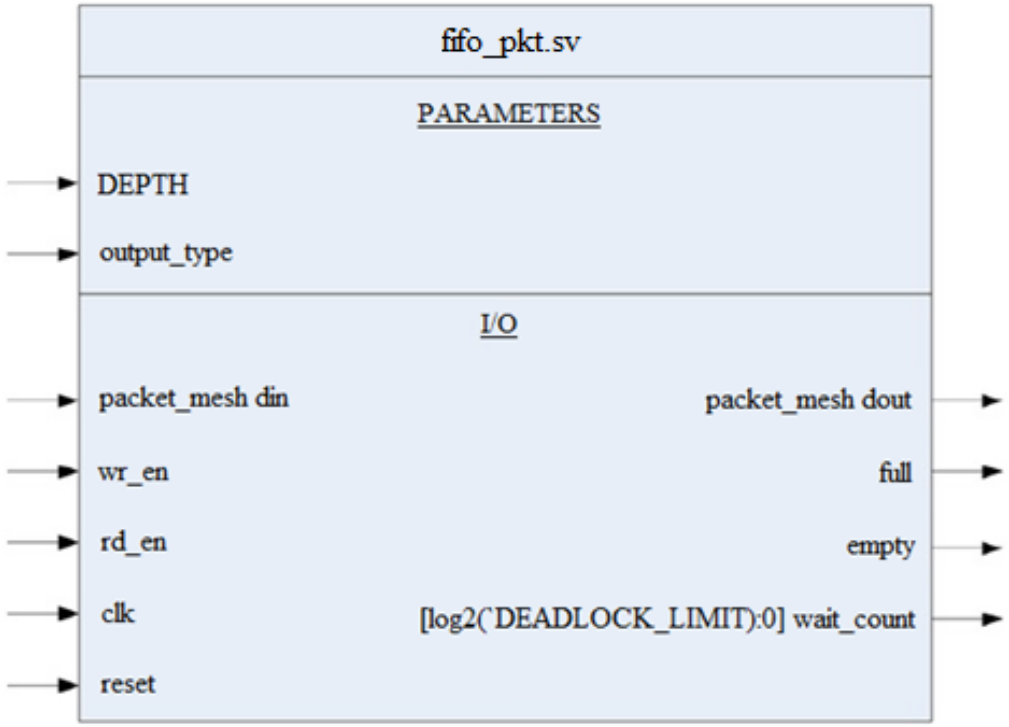


Figure 5.5: Model of the 'fifo_pkt.sv' module.

5.5.2 Parameters

The following table describes the parameters of 'fifo_pkt.sv'.

Name	Description
DEPTH	Input buffer size.
output_type	Allows registered and unregistered outputs.

Table 5.5.1: Parameters of the 'fifo_pkt.sv' module.

5.5.3 Inputs

The following table describes the inputs of ‘fifo_pkt.sv’.

Name	Size	Description
din	packet_mesh	Packet from a next-hop router.
wr_en	1	Logic high indicates ‘din’ is valid and writes to the back of the FIFO queue next clock cycle.
rd_en	1	Logic high indicates head-of-line packet is granted use of the switch and is sampled next clock cycle.
reset	1	Logic high nullifies resources.
clk	1	Clock.

Table 5.5.2: Inputs of the ‘fifo_pkt.sv’ module.

5.5.4 Outputs

The following table describes the outputs of ‘fifo_pkt.sv’.

Name	Size	Description
dout	packet_mesh	Head-of-line FIFO packet.
full	1	Logic high indicates full buffer.
empty	1	Logic high indicates empty buffer.
wait_count	[log2(`DEADLOCK_LIMIT):0]	Signal indicating how many clock cycles have passed without a logic high read.

Table 5.5.3: Outputs of the ‘fifo_pkt.sv’ module.

6 Method of Analysis

This section describes the necessary preparation procedure required to acquire meaningful results and the method to perform the actual measurements for both routing algorithms.

6.1 Network Equilibrium

For ease of design, the network is initialised with vacant buffers and idle resources. Initial packets injected into the network will therefore experience minimal contention and exit with low latency. After a certain time period, buffers begin to populate and increase overall packet latency. Eventually, the initialisation effect diminishes and the network reaches steady state. A similar phenomenon occurs at the end of the simulation where the last remaining packets exit the network and endure less traffic because packet injection stop this is known as the drain phase.

Latency measurements within these periods introduce systematic errors therefore, to minimise their influence, measurements will be taken during steady state operation. To do this, generated packets will be flagged with a ‘measure’ bit once the warm-up phase has passed, the testbench sink will assess this and only record measurements when it is logic high. Testing will be monitored to ensure packets injections continue long enough for all the measurement packets to reach their destinations. This continual injection is required up until the very last measurement packet so that it experiences similar background traffic interactions as the previous packets did throughout the test.

6.2 Latency

Packets contain the time of their creation within their payloads. Therefore, to measure packet latency, when the network sink receives a packet with the measure bit flagged, it subtracts the timestamp within the packets contents from the current simulation time. This is performed by 16 sink modules which produce a running total of latency measurements. To obtain the total latency of all the measurement packets, logic will be added to the testbench to cycle through all 16 sink instantiations and sum the latency values.

6.3 Synthetic Data

The network will be exposed to randomly generated packets with a uniform injection rate across all cores. The resulting total packet latency will be measured and average latency calculated. The injection rate will be increased until each network running deterministic and adaptive routing saturates. The testbench will be configured to ensure that at least tens of thousands of packets are measured for sufficient results and that it conforms to the steady state requirements.

Uniform random traffic is benign and must be complemented with other patterns to produce credible algorithm characterisation. Therefore, a hotspot test will be performed which will direct 30% of all network traffic towards the central cores to imitate increase demands for certain cores (see Fig 6.a).

A pattern typically common in SoCs will also be recreated. This pattern forces a specific core to stream packets to a single destination (see Fig 6.b). The rate of packet injection from this core will be increased and the latency at the corresponding destination measured. The remaining network will inject random traffic at a constant rate for the entire test.

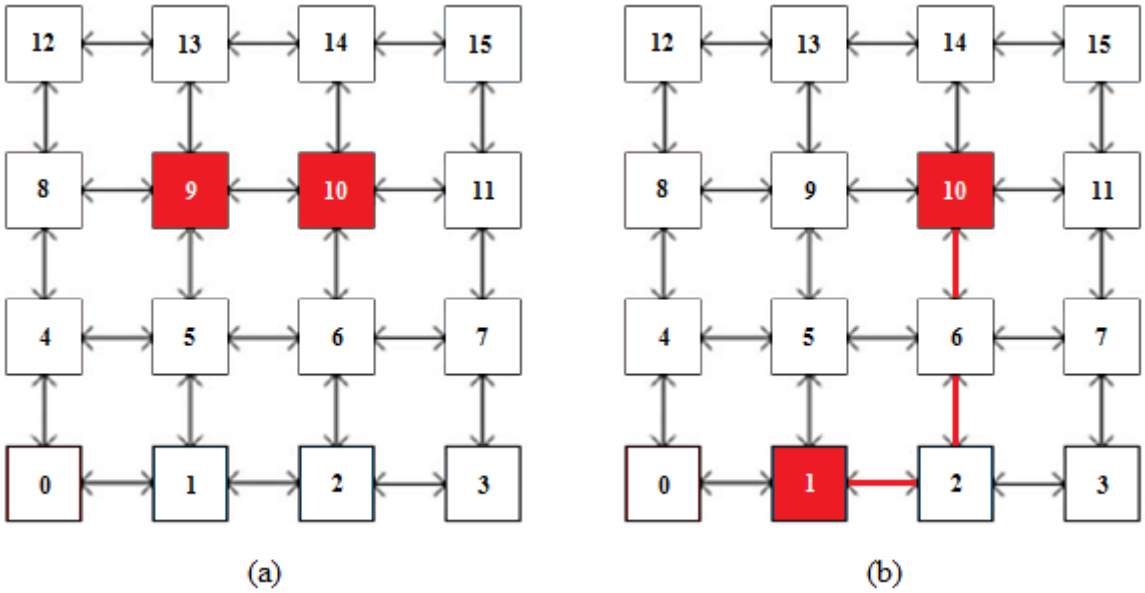


Figure 6: Identification of the main network cores considered in testing. (a) Core 9 and 10 are hotspot nodes receiving 30% of all network traffic. (b) Streaming of packets between core 1 (source) and 10 (destination) with different injection rates. The route used by the DOR-XY algorithm is highlighted.

6.4 Real Data

Trace data from various applications that are expected to run on future multi-core chips will be injected for both algorithms to produce average packet latency measurements (see Table 6). For each trace file, a limited number of measurements will be taken as there is no control over the injection rate. Measurements will be taken from both the start and the middle of the trace due to the possibility of different traffic behaviour depending on the time.

Programs	Description
Black Scholes	Option pricing with Black-Scholes Partial Differential Equation (PDE)
Fluid Animate	Fluid dynamics for animation purposes with Smoothed Particle Hydrodynamics (SPH) method
x264	H.264 video encoding

Table 6 [61]**:** A list of programs with trace files available for testing.

7 Testbench Implementation

The module ‘net_emulation.sv’ instantiates ‘packet_source.sv’/’packet_souce_trace.sv’ and ‘packet_sink.sv’ a parameterised number of times corresponding to the number of cores in the network and one instance of the NoC. Combinational logic was implemented to convert a generic packet type into a mesh topology type allowing xy destination values to be obtained from a destination node number. Logic to sum the total number of packets in and out of the network and the total packet latencies from all ‘packet_sink.sv’ modules was also added to verify the correctness and to allow characterisation of the routing algorithms. Correct operation was shown when the total number of packets in was identical to the number ejected and when no destination errors were present. Combinational logic was appended to take batch latency measurements and write them to a text file to be analysed using spreadsheet software. This was required to determine the warm-up period of the network.

7.1 Configuration

A test configuration file ‘config.sv’ is used to declare the packet structure and define constants to parameterise the design. Using parameters throughout the whole design enabled the network to be scaled to 16-cores efficiently. This file allows complete control over the setting of network parameters, routing algorithm parameters and the testing pattern desired for simulation. Detailed code comments have been added to enable future referencing (see Appendix 10.2).

7.2 Packet Source

The design described in Section 4.4.3 has been implemented into ‘packet_source_trace.sv’ which a copy of ‘packet_source.sv’ but, has been adapted to allow 16 files corresponding to each core to be read by using a SystemVerilog case statement. Time and destination data extracted are concatenated and used to generate input packets. A Matlab script for pre-processing was written to format the raw trace file following the algorithm in Section 4.4.3 (Fig 4.9.2) (see Appendix 10.3 for a snippet code for core zero). Both packet source modules were modified to set the ‘measure’ bit of packets high when packets meet the warm-up period and when the total number of measured packets is less than a chosen constant to allow correct steady-state operation. The ‘packet_sink.sv’ module has been adapted to only record packet latencies with the ‘measure’ bit flagged.

The difference between ‘packet_source.sv’ and ‘packet_source_trace.sv’ is that only ‘packet_source.sv’ was modified to incorporate the extra test patterns; hotspot and streaming. Hotpot traffic was implemented by using LFSRs and conditional statements to formulate an approximate 30% chance of sending to two central cores. For streaming, the packet generation of a specific source was hardcoded with a destination and the ‘measure’ bit was set only if the packet had the specific source number. Both modules utilise identical top-level sensitivity signals therefore only one will be presented.

7.2.1 Module

The following diagram shows the parameters, inputs and outputs of ‘packet_source_trace.sv’.

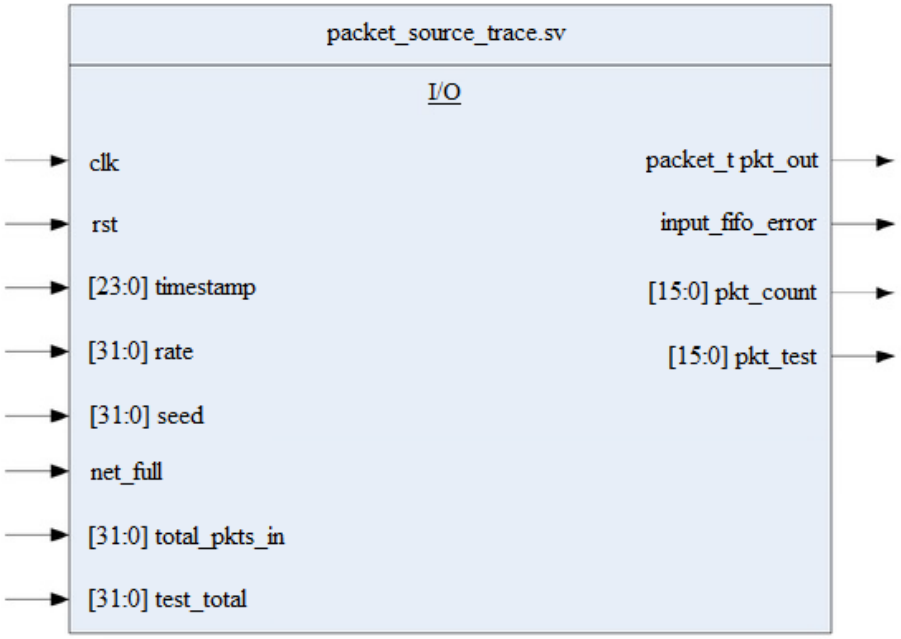


Figure 7.2: Model of the ‘packet_source_trace’ module.

7.2.2 Inputs

The following table describes the inputs of ‘packet_source_trace.sv’.

Name	Size	Description
clk	1	Clock.
rst	1	Logic high nullifies test resources.
timestamp	[23:0]	Number of clock cycles passed.
rate	[31:0]	Rate of packet injection.
seed	[31:0]	Clock.
net_full	1	Logic high indicates the network is full.
total_pkts_in	[31:0]	Count of total number of packets injected
test_total	[31:0]	Count of total number of packets injected passing a specific condition (time or number of packets already injected).

Table 7.1: Inputs of the ‘packet_source_trace.sv’ module.

7.2.3 Outputs

The following table describes the outputs of ‘packet_source_trace.sv’. (See Appendix 10.2 for details on packet_t size).

Name	Size	Description
pkt_out	packet_t	Head-of-line FIFO packet.
input_fifo_error	1	Logic high indicates full buffer.
pkt_count	[15:0]	Increments for valid packet injections.
pkt_test	[15:0]	Increments for valid packets injected passing a specific condition (time or number of packets already injected).

Table 7.2: Outputs of the ‘packet_source_trace.sv’ module.

8. Results

This section analyses the latency results recorded from simulations of 16-core 2D mesh NoC under different traffic patterns during steady state for the two routing algorithms.

8.1 Warm-Up Estimation & Drain

To estimate the warm-up period of the network, batch averages of packet latencies were obtained with batch sizes of 50 individual packet latencies. Using this method, time-evolution graphs of packet latencies from the network at 60% capacity were plotted for both algorithms (see Fig 8.1). The results reveal that for both algorithms, a single simulation does not show steady after 10,000 packet injections, this results because a random process is being sampled, introducing sampling error. To reduce this error, an ensemble average of 25 independent simulations was performed and the resulting curve clearly illustrates that steady state is achieved after approximately 2,000 successful packet injections. For an injection rate of 0.6, approximately 3,333 cycles corresponds to the required warm-up time to inject 2,000 packets. This period was utilised for the subsequent tests and equates to approximately 20,000 cycles for the minimum rate tested.

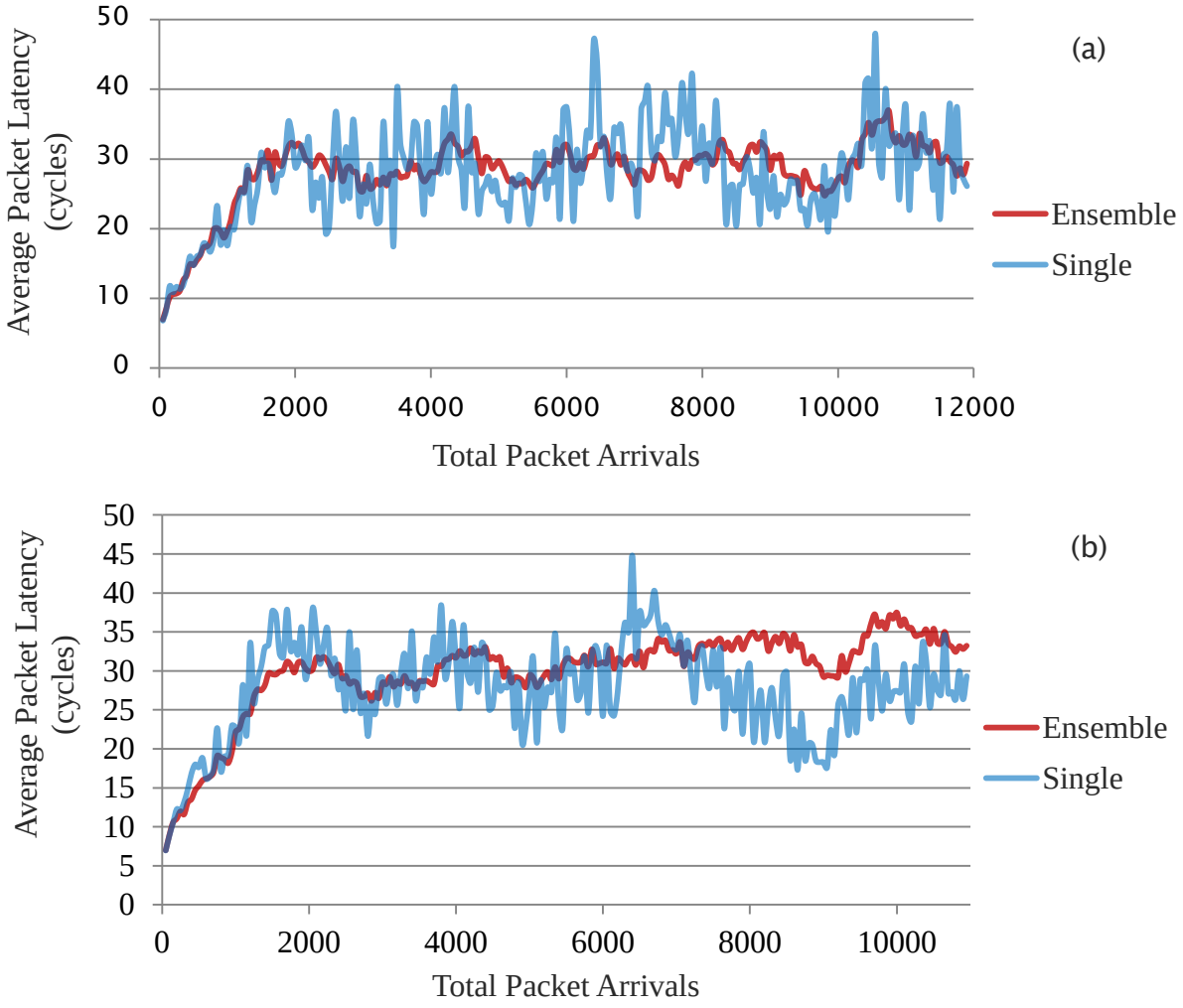


Figure 8.1: Batch latency averages of NoC packets using different routing algorithms under uniform load at 60% capacity. (a) Deterministic. (b) Adaptive.

8.2 Synthetic Traffic

Under uniform random traffic both algorithms display similar average latency results of approximately 6 - 11 cycles when faced with traffic under 45% capacity (see Fig 8.2). The saturation point of the network utilising adaptive routing occurs at 60%, a slightly lower injection rate than compared to the network using deterministic. This drop in performance may be linked to the adaptive algorithms lack of global traffic awareness; it has only local knowledge and therefore, may route to a less congested next-hop path but this path may be heavily congested downstream.

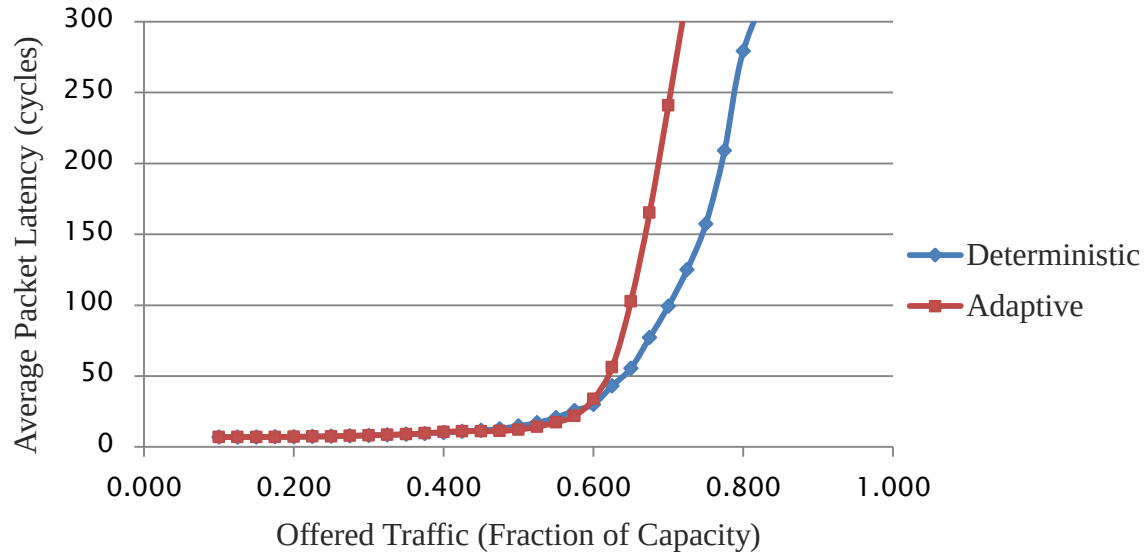


Figure 8.2: Latency performance of both routing algorithms under uniform traffic.

The centralised hotspot test shows that the adaptive algorithm has better latency performance saturating at a load approximately 15% higher than the deterministic algorithm (see Fig 8.3) due to the ability to avoid congestion by taking alternative routes however, induces higher total packet hops and possibly greater power overhead.

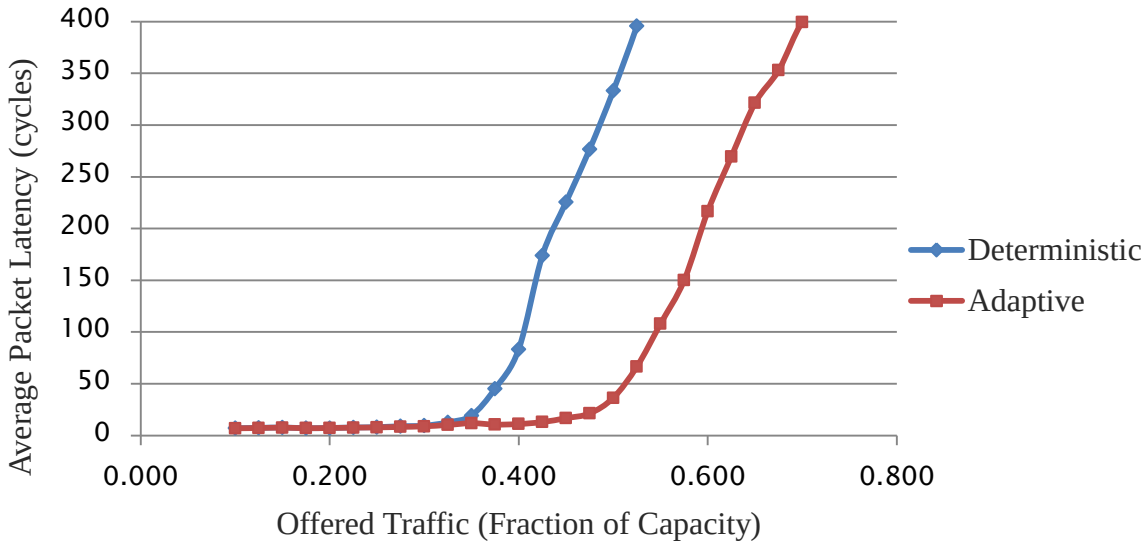


Figure 8.3: Latency performance of both algorithms under 30% centralised hotspot traffic.

The test results from streaming between two cores with increasing rate alongside random traffic on the remaining network shows benefits of the congestion-awareness of the adaptive algorithm. It is able to tolerate much higher traffic saturating at a load 6% higher than the deterministic routing as shown in Fig 8.4 and exposes its suitability in cases where network faults are present.

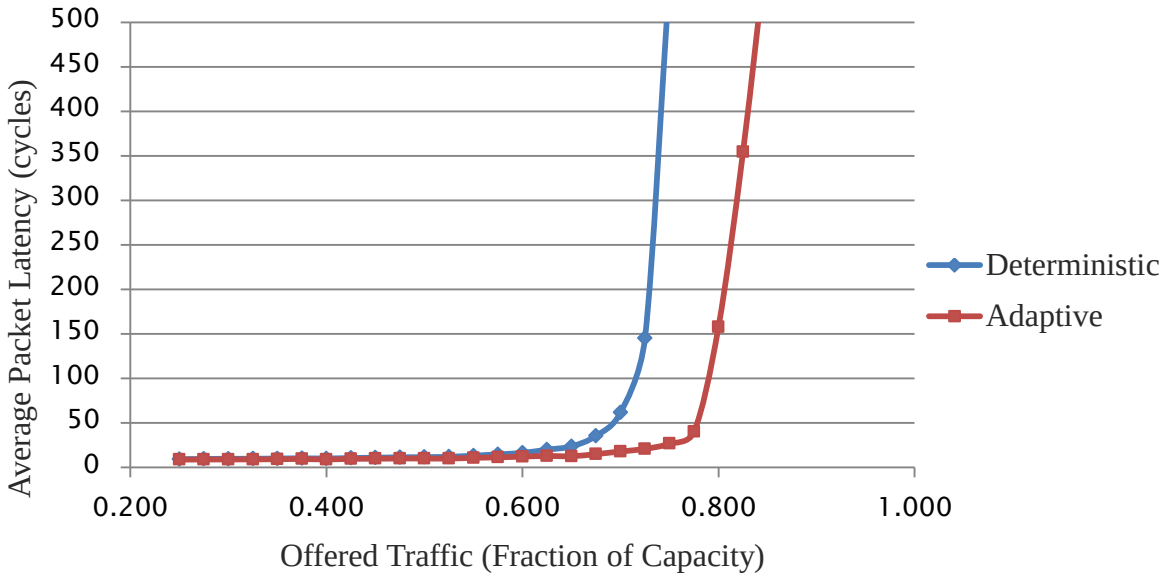


Figure 8.4: Latency performance of both algorithms under explicit core streaming.

8.3 Real Traffic

Both routing algorithms were simulated with traffic from real applications. The simulations produce identical latency results for both algorithms for each test program, this is because the traffic appeared in small bursts separated by long idleness allowing the network sufficient time to deliver packets and return to an empty state. The traffic was low rate because it was sourced by general purpose cores running only a single program. Some files contained high traffic densities such as the video codec however, were all manageable by both algorithms and produced average latency results under 10 cycles for traffic at different points in the program, as shown in the table below.

Programs	Average Latency from Early Program Simulation (cycles)	Average Latency from Mid-Program Simulation (cycles)
Black Scholes	4.9	6.2
Fluid Animate	4.9	5.5
x264 Video Encoding	5.0	4.8

Table 8: Average latency results from simulation of both routing algorithms using the programs described in Section 6.4 for early program and mid-program traces.

9. Conclusion

A unique fully-adaptive routing algorithm with livelock and deadlock recovery methods has been successfully implemented into a baseline Network-on-Chip along with a reusable testbench facility which enables both, synthetic and real-world network load injections for simulation by using pseudorandom number generators and trace file formatting. The complete design has been fully parameterised and network properties can be configured using a single SystemVerilog file to allow scaling for future development.

The design has been scaled to a 4x4 16-core 2D mesh Network-on-Chip and, simulations have enabled characterisation of the latency performance of a deterministic DOR-XY and a adaptive routing algorithm. Random uniform synthetic traffic tests have shown that deterministic routing allows larger capacity with higher traffic latency saturation than compared to the adaptive routing and, implies that the deterministic nature provides better load distribution. Testing with centralised hotspots of 30% shows that the adaptive algorithm outperforms since saturation occurred at load approximately 15% higher than the static algorithm. A test pattern of specific high link utilisation, which also impersonates a network fault, has also been conducted and streams traffic of increasing rate between a source-destination pair within a network of constant rate random traffic. The results show that the freedom induced by the adaptive algorithm allowed packets to be routed via paths with lower contention resulting in larger injection rate latency saturation than compared to the deterministic case. Tests involving application-driven workloads displayed identical latency results for both routing algorithms due to the low communication rates of the traces provided. Overall, these results provide a strong argument for adaptive routing to replace deterministic routing for resolving faults, however it heavily depends on the underlying application.

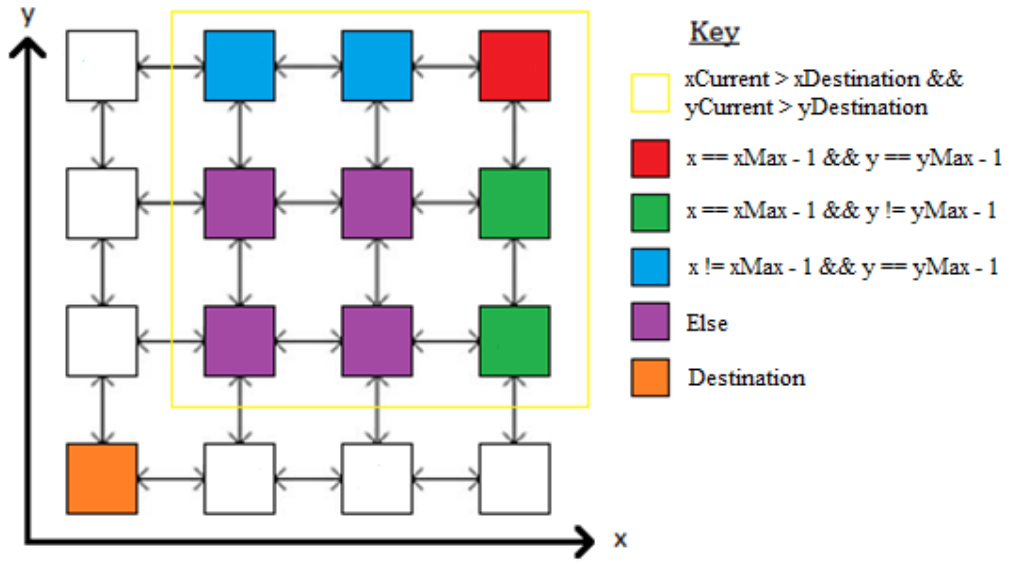
To further develop the adaptive routing algorithm, route calculation should consider global traffic in place of localised loads so that an optimal route is chosen since, the current algorithm can suffer from downstream congestion or faults due to poor local decisions. This should be complemented with an evaluation of the trade off between performance and design complexity. To allow testing of true fault avoidance, fault detection functionality should be implemented into the router design; the current design only imitates faults through use of high rate packet streams. To fully explore the capabilities of the algorithms, application-driven traffic from cores running multiple programs simultaneously should be used to source high communication loads. Some other possible improvements would be to, synthesise the router modules and examine the hardware difference required for both types of algorithms or to, synthesise the complete design onto a FPGA for timing and power analysis.

Simulation results show that the adaptive capabilities of adaptive routing present a potential to replace static algorithms in the case of fault-tolerance, however other figures of merit such as power and area must also be taken into consideration to produce a fully justified argument.

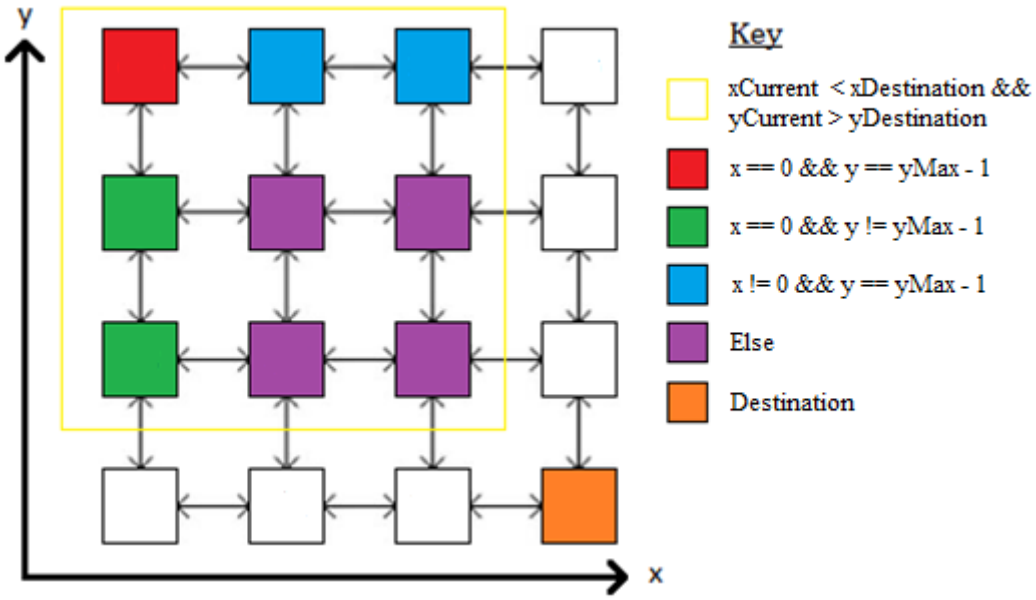
10 Appendix

10.1 Routing Algorithm Boundary Conditions

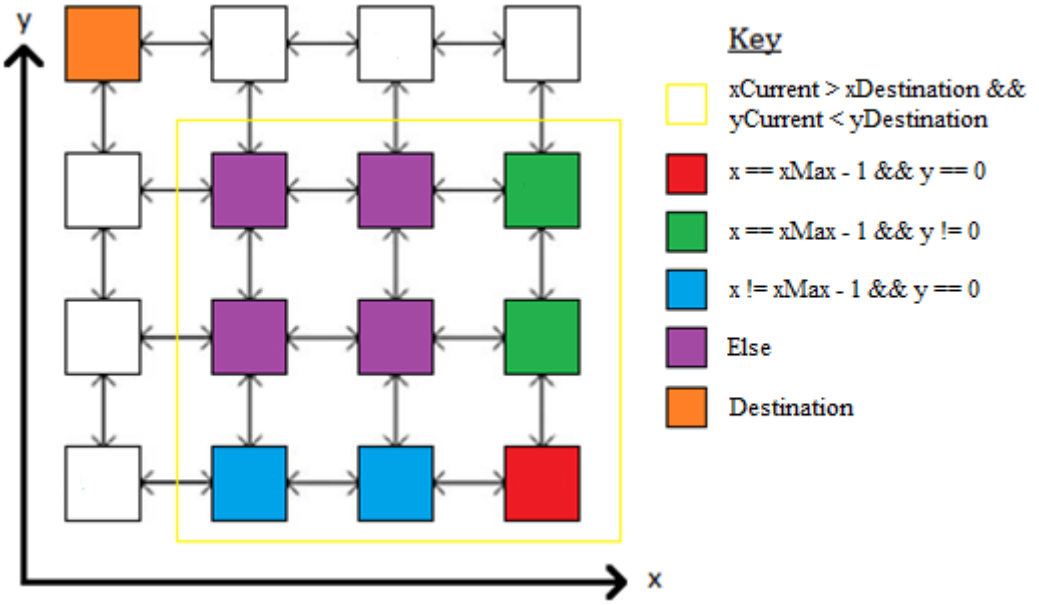
10.1.1 (X Current > X Destination) AND (Y Current > Y Destination)



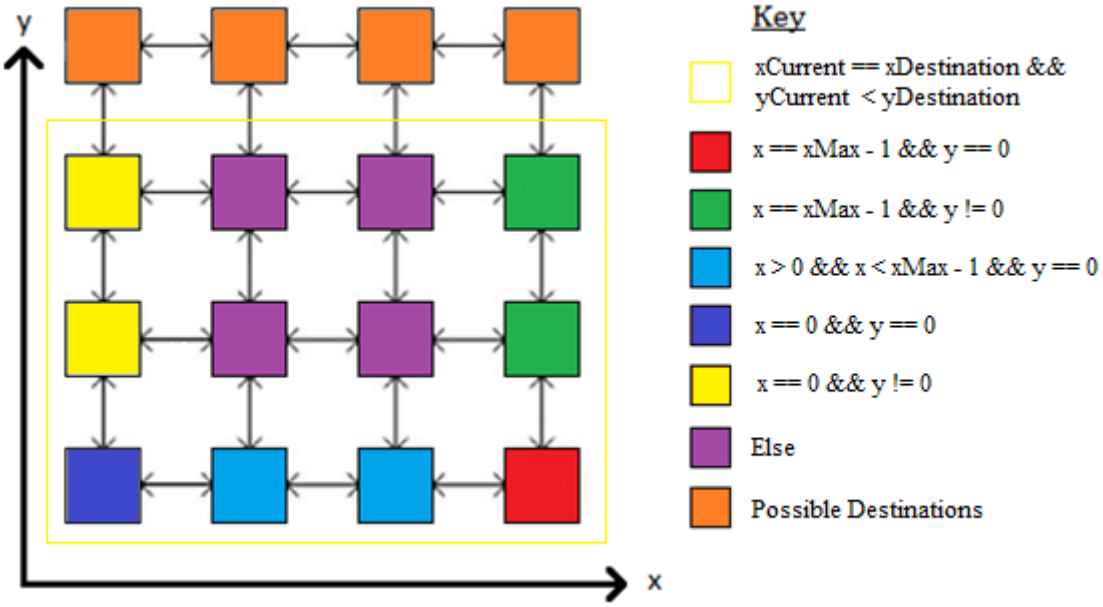
10.1.2 (X Current < X Destination) AND (Y Current > Y Destination)



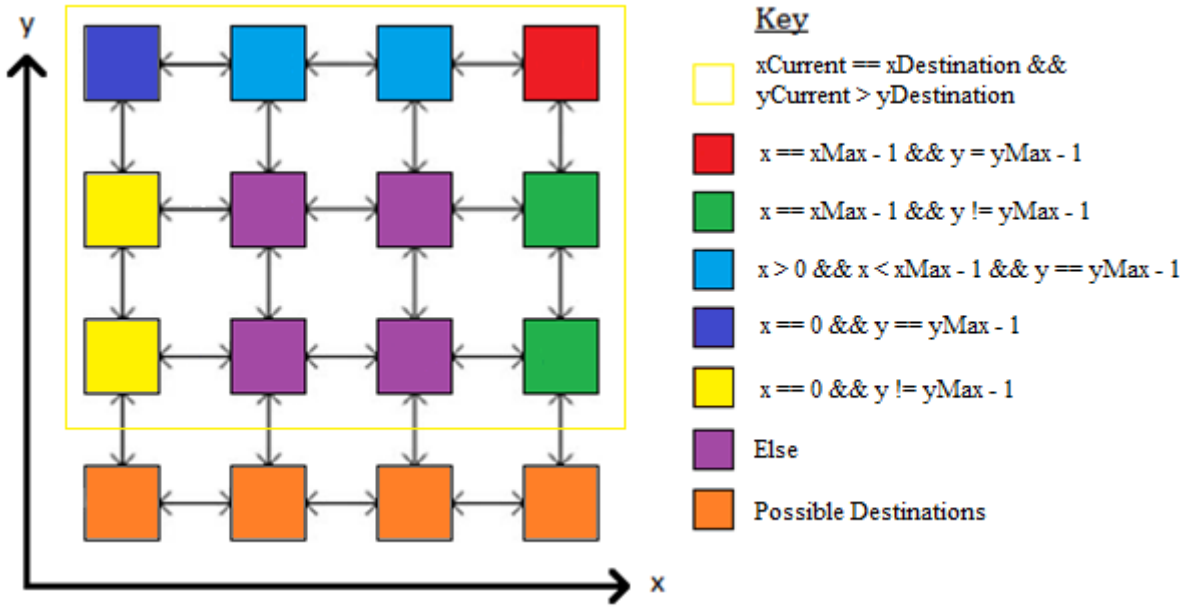
10.1.3 (X Current > X Destination) AND (Y Current < Y Destination)



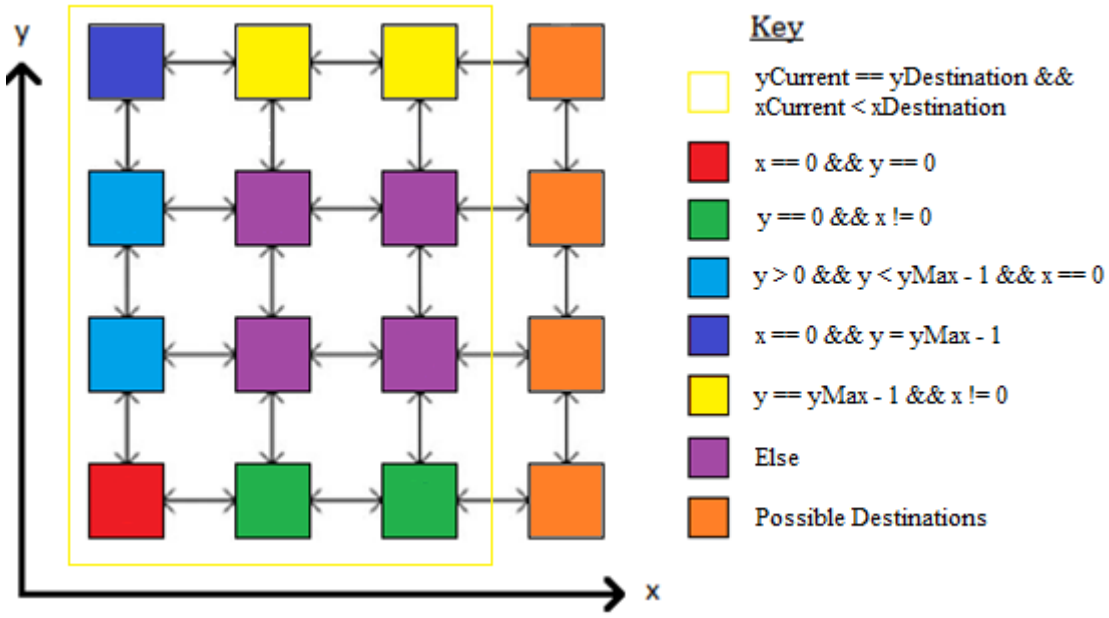
10.1.4 (X Current = X Destination) AND (Y Current < Y Destination)



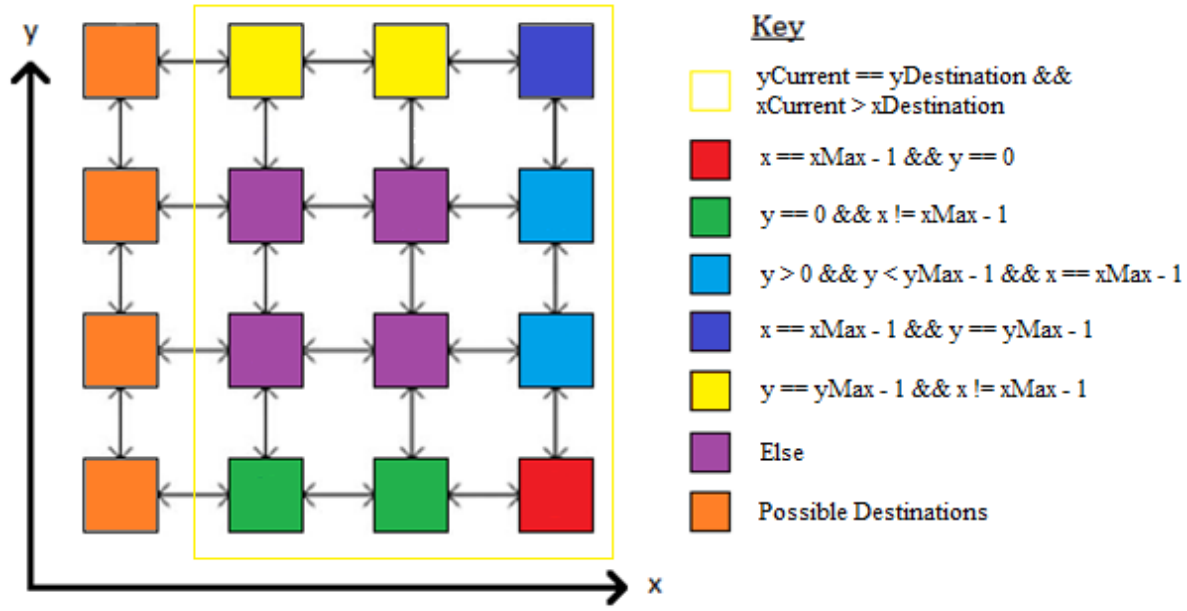
10.1.5 (X Current = X Destination) AND (Y Current > Y Destination)



10.1.6 (Y Current = Y Destination) AND (X Current < X Destination)



10.1.7 (Y Current = Y Destination) AND (X Current > X Destination)



10.1 SystemVerilog Configuration File

```
`include "functions.sv"

// Network parameters
`define PORTS 16           // Number of Cores
`define FIFO_DEPTH 4       // Size of FIFO
`define PAYLOAD 64         // Packet payload in bits
`define MESH_WIDTH 4       // Width of mesh network, used to calculate xy location
`define THRESHOLD 3        // Buffer threshold, used in the adaptive routing algorithm
`define UNPRODUCTIVE 4     // Limit the number of unproductive reroutes/misroutes (LIVELOCK)
`define DEADLOCK_LIMIT 16 // Number of cycles to assume deadlock
`define LOAD 400000000     // Determines injection rate, 100% load is 4e9

// Comment this out to run fully adaptive routing algorithm
//`define DETERMINISTIC

// Comment this out for synthetic packet simulation
//`define TRACE

// Uncomment the test desired - Has no effect if TRACE uncommented
`define UNIFORM
//`define HOTSPOT // Currently ~ 30% centralised
//`define STREAM  // Currently streaming from node 1 to 10, rate defined below
    `define STREAMRATE 1.4 // Rate of the STREAM test, fraction of LOAD

// Define the packet structure
typedef struct packed {
   logic [`PAYLOAD-1:0] data; // Packet payload
   logic [log2(`PORTS)-1:0] source; // Source node number
   logic [log2(`PORTS)-1:0] dest; // Destination node number
   logic valid;
   logic [log2(2*`MESH_WIDTH + 2*`UNPRODUCTIVE)-1:0] hopCount; // Number of hops a packet experiences
   logic [log2(`UNPRODUCTIVE):0] reroute; // Bits used for livelock control
   logic measure; // Logic high indicates packet passes Warm-up & Drain conditions and should be measured
       } packet_t;

// Define the packet structure for Mesh Network
typedef struct packed {
   logic [`PAYLOAD-1:0] data; // Packet payload
   logic [log2(`PORTS)-1:0] source; // Source node number
   logic [log2(`MESH_WIDTH)-1:0] xdest; // Destination x address
   logic [log2(`MESH_WIDTH)-1:0] ydest; // Destination y address
   logic valid;
   logic [log2(2*`MESH_WIDTH + 2*`UNPRODUCTIVE)-1:0] hopCount; // Number of hops a packet experiences
   logic [log2(`UNPRODUCTIVE):0] reroute; // Bits used for livelock control
   logic measure; // Logic high indicates packet passes Warm-up & Drain conditions and should be measured
       } packet_mesh;
```

10.2 Matlab Trace Data Formatting Script

```
% The script currently operates on traces of the format (writing only Time and Destination to a new text file):
% Time; Source; Destination; Size(in bytes); Message Type
% To use different formats change the conditional statement and offsets depending on the requirements.
% Bit sizes of Time and Destination and number of unique trace fields have been parameterised using constants.

clc;
clear all;
input = textread('raw_trace_data.txt','%s','delimiter',';');
input = str2double(input); % Convert each input to type double
[sizeY sizeX] = size(input); % Obtain the dimensions of the matrix

uniqueFields = 5; % Number of unique fields in each trace file packet (row)
timeSize = 23; % Number of bits to store the Time field
destinationSize = 4; % Number of bits to store the Destination field

% Write array index counter for source/node 0
zeroAppend = 1;

% Loops through the delimited data in steps of the number of unique fields contained
% and stores the relevant fields as binary numbers into the core zero array
for data = 1:uniqueFields:(sizeY)
    % Identify the current source, Convert time field and destination field to binary
    % and copy input data into output array
    if input(data+1) == 0
        if (input(data+2) < 16)
            nodeZero{zeroAppend} = dec2bin(input(data),timeSize);
            zeroAppend = zeroAppend + 1;
            nodeZero{zeroAppend} = dec2bin(input(data+2),destinationSize);
            zeroAppend = zeroAppend + 1;
        end
    end
end

% Opens core zero text files and writes the core zero array 'nodeZero'
fileID = fopen('nodeZero.txt','w');
for i = 1:size(nodeZero,2) % Loop from 1 to second dimension of array
    fprintf(fileID,'%s\n',nodeZero{i}); % Write cell array to text file
end
fclose(fileID);
```

11 References

[1] G. Blake, R.G. Dreslinski, and T. Mudge, "A Survey of Multicore Processors," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 26-37, Nov. 2009.

[2] L. Karam, I. AlKamal, A. Gatherer, G. Frantz, D. Anderson, and B. Evans, "Trends in Multicore DSP Platforms," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 38-49, Nov. 2009.

[3] R. Kamal and N. Yadav, "NoC and Bus Architecture: A Comparison," *International Journal of Engineering Science and* Technology, vol. 4, no. 4, pp. 1438-1442, Apr. 2012.

[4] N.E. Jerger and L. Peh, "On Chip Networks (Synthesis Lectures on Computer Architecture)," Morgan and Claypool Publishers, 2009.

[5] M. Hübner, "Multiprocessor System-On-Chip: Hardware Design and Tool Integration," Springer, 2011.

[6] S. Pasricha and N. Dutt. "On-Chip Communication Architectures: System on Chip Interconnect," Morgan Kaufmann, 2008.

[7] A. Jerraya and W. Wolf, "Multiprocessor Systems-on-Chips," Morgan Kaufmann, 2004.

[8] C.E. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, A. Wasicek, "The ACROSS MPSoC - A New Generation of Multi-core Processors Designed for Safety-Critical Embedded Systems," *Digital System Design (DSD), 15th Euromicro Conference*. pp. 105-113, Sept. 2012.

[9] H. Iwai. "Roadmap for 22 nm and beyond," *Microelectronic Engineering*, vol. 86, no. 7-9, pp. 1520-1528, July-Sept. 2009

[10] I. W. Group. International technology roadmap for semiconductors. 2011 Edition. [Online] Available at: http://www.itrs.net/Links/2011ITRS/Home2011.htm [Accessed: 18 December 2012].

[11] I. Cidon and I. Keidar, "Zooming in on network-on-chip architectures," *SIROCCO Proceedings of the 16th international conference on Structural Information and Communication Complexity*, pp. 1, 2009.

[12] K. S. Hassan, A. Reza, M. Reshadi, "Yield modeling and Yield-aware Mapping for Application Specific Networks-an-chip," *NorChip*, pp. 1-4, Nov. 2011.

[13] S. Rodrigo, C. Hernández, J. Flich, F. Silla, J. Duato, S. Medardoni, D. Bertozzi, A. Mejía and D. Dai, "Yield-oriented evaluation methodology of network-on-chip routing implementations," *System-on-Chip, International Symposium*, pp. 100-105, Oct. 2009.

[14] I. W. Group. International technology roadmap for semiconductors. 2007 Edition. [Online] Available at: http://www.itrs.net/links/2007ITRS/Home2007.htm [Accessed: 10 February 2012].

[15] H. Haznedar, M. Gall, V. Zolotov, P. S. Ku, C. Oh and R. Panda, "Impact of Stress-Induced Backflow on Full-Chip Electromigration Risk Assessment," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1038-1046, Jun. 2006.

[16] S. Shamshiri and K.Cheng, "Modeling Yield, Cost, and Quality of a Spare-Enhanced Multicore Chip," *IEEE Trans. Computers*, vol. 60, no. 9, pp. 1246-1259, Sept. 2011.

[17] M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. T. Kandemir and M. J. Irwin, "Fault tolerant algorithms for network-on-chip interconnect," *VLSI Proceedings. IEEE Computer society Annual Symposium*, pp. 46-51, Feb. 2004.

[18] M. Behrouzian Nejad, A. Mehranzadeh, M. Hoodgar, "Performance of Input and Output Selection Techniques on Routing Efficiency in Network-on-Chip," *International Journal of Computer Science and Information Security*, vol. 9, no. 9, pp. 125-130, 2011.

[19] J. Lin, X. Lin, L. Tang, "Making-a-stop: A new bufferless routing algorithm for on-chip network," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 515-524, Jan. 2012.

[20] N. Salehi, A. Khademzadeh and A. Dana, "Minimal fully adaptive fuzzy-based routing algorithm for Networks-on-Chip," IEICE Electronics Express, vol. 8, no. 13, pp. 1102-1108, Jul 2011.

[21] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Computing Surveys*, vol. 38, no. 1, p. article No. 1, 2006.

[22] J. Probell, "Power Benefits of Modular Interconnect Design Using Network-on-Chip Technology Arteris."

[23] L. Benini, G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Trans. Computers*, vol. 35, no. 1, pp. 70-78, Jan. 2002.

[24] L. Benini, G. De Micheli, "Networks on Chips: Technology and Tools," Morgan Kaufmann, 2006.

[25] M. Horowitz, R. Ho and K. Mai. "The Future of Wires." *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490-504 Apr. 2001.

[26] S.W. Keckler, D. Burger, C. R. Moore, R. Nagarajan, K. Sankaralingam, V. Agarwal, M. S. Hrishikesh, N. Ranganathan, and P. Shivakumar, "A Wire-Delay Scalable Microprocessor Architecture for High Performance Systems". *Proceedings of the 2003 International Solid-State Circuits Conference*, vol. 1, pp. 168-169, 2003.

[27] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor System-on-Chip (MPSoC) Technology," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1701-1713, Oct. 2008.

[28] I. Stojmenovic, R. K. Thulasiram, L. T. Yang, W. Jia, M. Guo, R. Fernandes de Mello, "Parallel and Distributed Processing and Applications," *Lecture Notes in Computer Science. 5th International Symposium*, vol. 4742, pp. 278-288, 2007.

[29] M. Ali, M. Welzl, M. Zwicknagl and S. Hellebrand, "Considerations for fault-tolerant Network on Chips," *Microelectronics ICM, The 17th International Conference*, Dec. 2005.

[30] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection Networks," *Design Automation Conference, Proceedings of the 38th Conference*, pp. 684–689, Jun. 2001.

[31] C. Silvano, M. Lajolo and G. Palermo, "Low Power Networks-on-Chip," Springer, 2011.

[32] W. J. Dally and B.P. Towles, "Principles and Practices of Interconnection Networks," Morgan Kaufmann, 2004.

[33] S. Chen, Y. Lan, W.Tsai, "Reconfigurable Networks-On-Chip," Springer, 2012.

[34] A. Jantsch and H. Tenhunen, "Networks on Chip," Springer, 2003.

[35] Arteris S.A, "From Bus and Crossbar to Network-On-Chip," 2009. [Online] Available at: http://chipdesignmag.com/sld/files/2009/11/NoC-Evolution_v11.pdf [Accessed: 20 May 2012].

[36] M. Mirza-Aghatabar, S. Koohi, S. Hessabi and M. Pedram, "An Empirical Investigation of Mesh and Torus NoC Topologies Under Different Routing Algorithms and Traffic Models," *Digital System Design Architectures, Methods and Tools, 10th Euromicro Conference*, pp. 19-26, Aug. 2007.

[37] D. L. Nicholls, "Network-on-Chip," Mar. 2012

[38] P. Lotfi-Kamran, A. M. Rahmani, M. Daneshtalab, A. Afzali-Kusha and Z. Navabi, "EDXY - A low cost congestion-aware routing algorithm for network-on-chips," *Journal of Systems Architecture: the EUROMICRO Journal*, vol. 56, no. 7, pp. 256-264, Jul. 2010.

[39] A. Patooghy and S.G. Miremadi, "XYX: A Power & Performance Efficient Fault-Tolerant Routing Algorithm for Network on Chip," *Parallel, Distributed and Network-based Processing, 17th Euromicro International Conference*, pp. 245-251, Feb. 2009.

[40] M. Li, Q. A. Zeng, and W. B. Jone, "DyXY - a proximity congestion-aware deadlock-free dynamic routing method for network on chip," *Design Automation Conference, 43rd ACM/IEEE*, pp. 849-852, 2006.

[41] M. Nickray, M. Dehyadgari and A. Kusha, "Adaptive Routing Using Context-Aware Agents for Networks on Chips," *Design and Test Workshop (IDT) 4th International*, pp. 1-6 Nov. 2009.

[42] T. K. Moon, "Error Correction Coding: Mathematical Methods and Algorithms," John Wiley & Sons, 2005.

[43] C. Maxfield, "Bebop to the Boolean Boogie: An Unconventional Guide to Electronics," Newnes, 2008.

[44] W.J. Dally and G.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, vol. 36, no. 5, pp. 547-553, May. 1987.

[45] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320-1331, Dec. 1993.

[46] M. E. Gomez, N. A. Nordbotten, J. Flich, P. Lopez, A. Robles, J. Duato, T. Skeie and O. Lysne, "A routing methodology for achieving fault tolerance in direct networks Computers," *IEEE Trans. Computers*, vol. 55, no. 4, pp. 400-415, Apr. 2006.

[47] J. Hu and R. Marculescu, "DyAD - smart routing for networks-on-chip," *Design Automation Conference Proceedings. 41st*, pp. 260-263, Jul. 2004.

[48] C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *Computer Architecture. Proceedings. The 19th Annual International Symposium*, pp. 278-287, 1992.

[49] G.M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel Distributed Systems*, vol. 11, no. 7, pp. 729-738, Jul. 2000.

[50] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 10, pp. 1055-1067, Oct. 1995.

[51] N. Agrawal and C. P. Ravikumar, "Adaptive Routing Based on Deadlock Recovery," *Proceeding Euro-Par '98 Parallel Processing, 4th International Euro-Par Conference*, pp. 981-988, Sep. 1998.

[52] T. Moscibroda and O.Mutlu, "A case for bufferless routing in on-chip networks," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 196-207, Jun. 2009.

[53] C. Fallin, C. Craik, and O. Mutlu, "CHIPPER: A low-complexity bufferless deflection router," *HPCA Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 144-155, 2011.

[54] S. D. Chawade, M. A. Gaikwad and R. M. Patrikar, “Design XY Routing Algorithm for Network-On-Chip Architecture,” *International Journal of Research And Development*.

[55] C. S. Grecu, A. Ivanov, R. A. Saleh, C. Rusu, L. Anghel, P. P. Pande and V. Nuca, “A flexible network-on-chip simulator for early design space exploration,” *Microsystems and Nanoelectronics Research Conference MNRC 1st*, pp. 33-36, Oct. 2008.

[56] A. B. Garcia-Hernando, J. F. Martínez-Ortega, J. M. López-Navarro, A. Prayati, and L. Redondo-López, “Problem Solving for Wireless Sensor Networks,” Springer, 2009.

[57] Y. Sun, S. Kumar and A. Jantsch, “Simulation and Evaluation for a Network on Chip Architecture Using Ns-2,” *Proceedings IEEE NorChip Conference*, 2002.

[58] J. Xu, W. Wolf, J. Henkel and S. Chadradhar, “A Design Methodology for Application-Specific Networks-on-Chip,” *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 263-280, May. 2006.

[59] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, “Cost Considerations in Network on Chip,” *Integration, the VLSI Journal - Special issue: Networks on chip and reconfigurable fabrics*, vol. 38, no. 1, pp. 19-42, Oct. 2004.

[60] L. Zhonghai and A. Jantsch, “Traffic configuration for evaluating networks on chips,” *System-on-Chip for Real-Time Applications, 5th International Workshop*, pp. 535-240, Jul. 2005.

[61] The PARSEC Benchmark Suite. [Online]
Available at: http://parsec.cs.princeton.edu/overview.htm [Accessed: 19 March 2013].