A Survey of Efficient On-Chip Communications for SoC

Nikolay Kavaldjiev, Gerard J. M. Smit
Department of EEMCS
University of Twente
Enschede, the Netherlands

{nikolay, smit}@cs.utwente.nl

Abstract

This paper provides a survey of methods and techniques for flexible on-chip inter-processor communications for Systems-on-a-Chip (SoC) components. These devices are applied in battery-powered mobile multimedia devices. A classification is made of the most popular interconnection methods, techniques and mechanisms used for the inter-processor communications including interconnection network topologies, switching techniques, flow control, routing, deadlock avoidance and queuing techniques. Several projects dealing with on-chip interconnection networks are reviewed. As mobile multimedia devices are battery powered special attention is paid to energy-efficiency, the ways it can be achieved and how the different techniques contribute to it.

1 Introduction

A variety of portable equipment has found its way into our daily lives. Most things we carry with us these days are digital or have digital alternatives. There is a clear trend to make portable devices digital. Digital photo and video cameras, digital personal music players and digital gaming devices are all gaining popularity on their analogue ancestors. The next generation of these devices will have to be able to perform more and more complex tasks under almost the same energy constrains as today. This makes their energy-efficiency one of the most important design criteria. The hardware platform we foresee for these devices is a tiled heterogeneous reconfigurable System-on-a-Chip (HRSoC). In such a system two main components can be recognized – processing entities and communication entities. While the processing entities deal with data processing and comprise all computation resources of the system, the communication entities provide an infrastructure for data exchange between the processing resources. In this paper an overview of methods and techniques for on-chip interconnection systems is given.

In the CHAMELEON[1] project a tiled heterogeneous reconfigurable System-On-a-Chip (HRSoC) for future low-power mobile multi-media devices is proposed. This HRSoC is built up from of a set of heterogeneous processing tiles that may contain general-purpose processing elements (GPP), bit-level reconfigurable units (e.g. embedded FPGAs), domain specific word-level reconfigurable units (DSRC) or application specific (ASIC) units (see *Figure 1*). In this section we will first discuss why a tiled organisation is chosen and then we will introduce the associated communication issues.

1.1 Tiled organisation

Today's semiconductor manufacturing techniques provide a huge transistor budget and allow putting different types of functions on the same chip. As a result it is possible to build a complete system, including processors, memory and analogue parts on a single chip. The latter is called a system-on-chip (SoC). Flexible and adaptive SoCs can be realized by integrating reconfigurable hardware parts of different granularities into heterogeneous

[1] This research is supported by PROGRESS, the embedded systems research program of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW.

reconfigurable SoCs (HRSoCs). Reusable HRSoCs are ever more appealing to industry due to the exponentially increasing mask costs for an ASIC [48]. Such a HRSoC with reconfigurable hardware parts of different granularities is in line with the idea to match an algorithm with hardware that can execute it efficiently. In our view, such a HRSoC for mobile devices should contain domain specific processor tiles connected by an on-chip interconnection network. In this section the benefits of such a tiled architecture are discussed.

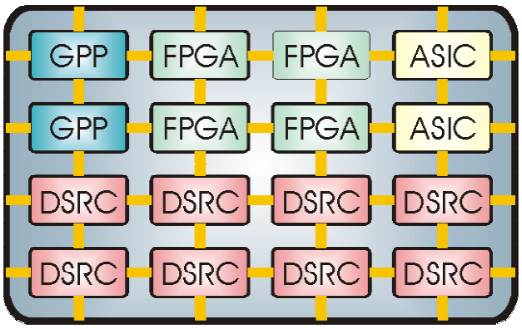

Figure 1: Heterogeneous tiled organisation

When a processing entity is replicated a number of times on a chip, a tiled architecture is obtained. A feasible template for a future-proof architecture is constructed from processing tiles that do not grow in complexity with technology.

Instead, as technology scales the number of tiles on the chip grows. An on-chip communication network then combines the tiles into a HRSoC [27]. An on-chip network that routes packets has a higher bandwidth than an on-chip bus, as it supports multiple concurrent communications. The well-controlled electrical parameters of an on-chip interconnection network enable the use of high-performance circuits that result in significantly lower power dissipation, higher propagation velocity and higher bandwidth than is possible with conventional circuits.

An architecture in which processor tiles are connected by an on-chip network has a very modular design. The modularity reduces design complexity and makes it possible to employ the enormous transistor budget that will be available in future integrated circuits. The design of a single tile is relatively simple and therefore a lot of effort can be put in power optimisations on the physical level of integrated circuit design. A tiled approach also eases verification of an integrated circuit design, since the design of identical tiles only has to be verified once. The computational power in a tiled architecture scales linearly with the number of tiles. The more tiles there are on a chip, the more computations can be done in parallel (providing that the network capacity scales with the number of tiles).

Accesses to memory display a high degree of locality. Temporal locality of reference refers to the observation that accessed data is often referenced again in the near future. Spatial locality of reference refers to the observation that once a particular memory location is accessed, a nearby location is often referenced in the near future. Accessing a small and local memory is much more energy-efficient than accessing a big and far distant memory. Transporting a signal over a 1mm wire in a 0.05µm technology will require more than 50 times the energy of a 32-bit operation in the same technology [49]. (The off-chip interconnect will consume more than a 1000 times the energy of a 32-bit operation!) A tiled architecture intrinsically encourages the usage of small and local in-tile memories. Exploiting the locality of reference principle extensively will improve the energy-efficiency substantially. The processor tiles in a tiled architecture can be viewed as a pool of resources.

Some algorithms might fit in a single processor tile, while others might require multiple tiles to achieve the required performance. Although tiles operate together in a complex system, an individual tile operates quite autonomously. In a tiled architecture every processor tile is configured independently. In fact, a tile is a natural unit for partial reconfiguration. Unused tiles can be configured for a new task, while at the same time other tiles are performing other tasks. That is to say, a tiled architecture can be reconfigured dynamically.

A tile processor might not need to run at full clock speed to achieve the required QoS at a particular moment in time. Also, when a task is pipelined on multiple processor tiles it is plausible that not all processor tiles in the pipeline require running at the same clock speed. An energy advantage can be gained when the processor tiles each have a configurable clock

frequency. The power consumption of a circuit varies linearly with its clock frequency, since power consumption is the time rate of energy consumption. A first order approximation of the dynamic power consumption P in a circuit is given by the formula: $P = \frac{1}{2} CV^2 f$ where f is the operating frequency of the circuit, C the total capacitance and V the power supply voltage. Power consumption varies linearly with clock speed and by the square of voltage. Reducing the supply voltage is far more lucrative than reducing the clock frequency. The supply voltage of a circuit cannot be reduced arbitrarily. The delay of a circuit increases significantly when the supply voltage is reduced. When the clock frequency is lowered, the supply voltage can be lowered as well. A configurable clock in combination with dynamic voltage scaling produces cubic reductions in dynamic energy consumption.

1.2 Interconnection networks

Inter-processor communication is a critical issue in a tiled HRSoC architecture. Communication properties (e.g. energy cost, latency, bandwidth, etc.) determine the granularity of the tasks that can be run on the processing tiles. The interconnection network is a key component for providing flexibility in reconfigurable systems. On the other hand the high energy consumption and the required chip area of the interconnections in related technologies (such as FPGAs) indicates that flexibility often comes at a high cost. Therefore, careful design of the interconnect architectures is needed to ensure low energy consumption and high flexibility. One of the problems is that the communication architecture has a number of conflicting requirements e.g. minimal energy consumption, the application's task graph should map to the physical communication topology of the on-chip network, flexible enough to handle yet unknown application, scalable from a small to a large number of processors and provide support for real-time multimedia traffic.

This research area can make use of the methods and techniques for the inter-processor communication networks for parallel systems. However, in parallel systems high performance is the one and only priority. In this research performance is not the only goal; there are other additional requirements: i.e. minimal energy consumption, and small size (realizable in a small silicon area). Because communication is expensive in terms of energy, *locality of reference* is an important concept in these systems: so the main strategy is to locate frequently referenced data close to the processor.

In the next section we collect the most important design criteria of interconnection networks for HRSoC. In the third section several related projects that concern interconnection networks for SoC are presented. Section four is a classification of the interconnection networks and presents characteristics and techniques used. Section five contains conclusions and some ideas for future work.

2 Requirements

This section summarises the most important requirements and criteria for on-chip interconnection networks for heterogeneous reconfigurable SoC designs (HRSoC) used in mobile devices.

2.1 Energy-efficiency

As energy-efficiency is a major design criterion for mobile systems we briefly review in this section the most important design principles for energy-efficient design. Although these principles are quite general, they are also applicable for designing energy-efficient on-chip communication networks. The following basic principles can be used to design efficient systems:

1. *Involve all layers*. Energy efficiency is an issue involving all layers of the system, its physical layer, its communication protocol stack, its system architecture, its operating system, and the entire application. Designing an energy-efficient system requires the system to be optimised for energy in an integral way. A system built out of components that have individually been optimised for low power does not necessarily result in the most energy-efficient system.

2. *Mapping application tasks onto the most suitable processing entity*. As indicated above three basic processor types can be distinguished:
    a. *General-purpose processing units* – that can be programmed to perform virtually any computational task, but have significant overhead for instruction stream processing and are quite energy hungry. However, they are very good in control type of applications (with frequent control constructions: if-then-else or while loops),
    b. Bit-level reconfigurable units – that offer fine-grain programmable resources, useful for application with bit-level operations (e.g. PN-code generation, turbo encoding),
    c. *Word-level reconfigurable units* – that offer resources with word-level reconfigurable data paths, designed for efficient processing of DSP-like algorithms (e.g. FFT, FIR filters). In the CHAMELEON project a word-level reconfigurable tile called MONTIUM is being developed [17]. The hardware organisation of such a tile is very regular and resembles very long instruction word (VLIW) architecture. The targeted algorithm domain of the tile comprises 16-bit digital signal processing algorithms that contain multiply accumulate operations such as FFT, FIR and linear interpolations. However, the MONTIUM is not limited to these algorithms [36].

    For an energy-efficient system the type of processing should be chosen in accordance with the computational characteristics of the task to be performed. The main philosophy is that operations on data should be done at the place where it is most energy efficient and where it minimizes the required communication.
3. *Applying locality of reference*. Locality of reference refers to locating frequently used data close to the processing entities. Accessing a small and local memory is much more energy-efficient than accessing a big and far distant memory.
4. *Quality of Service* is an essential mechanism for mobile multimedia systems not only to give users an adequate level of service, but also as a tool to achieve an energy efficient system. In a mobile multimedia environment, mobility and the need for efficient resource utilisation require the use of a 'soft' QoS model. Typically, the minimum QoS requirement for multimedia applications has a wide dynamic range depending on the user's quality expectations, application usage models, and application's tolerance to degradation. If the system would adapt to the required QoS and current environment, then the energy-efficiency can be improved considerably.
5. *Avoid useless activity*. This is the main driving force of for instance dynamic power management, link layer protocols and adaptive error control. Useless activity can be caused by various factors at all levels of the system (e.g. being unnecessary in a high power operational mode, applying error control to error-resilient data, trying to transmit a video frame that is already out-dated).
6. *Reduce the amount of communication*. This is quite obvious as communication consumes a large portion of the energy but this principle is applicable in all layers of the system. This also refers to the trade-off between communication and computation. Examples are adaptive error control that adapts its error coding according to the current channel conditions and the required quality, video transmission systems that adapt the quality to the expectations of the users, the available resources, and the channel conditions. Again, QoS can be used to determine whether it is really necessary to transport the data.
7. *Reduce energy dissipated in wiring*. Whereas the energy needed for computation and storage greatly benefits from technology improvement and device scaling, the energy for on-chip communication does not scale down [16]. Projections based on current optimisation techniques for global wires show that global on-chip communication will require more and more energy [15]. Hence, minimisation of the communication energy will be a growing concern in future systems. Minimisation of the communication energy dissipated in wires can be achieved by 1) physical optimisation of the on-chip interconnections, and 2) by efficient usage of these interconnections. For physical optimisation of the on-chip interconnections, i.e. reducing the energy for transmission of a data unit on the interconnection wires, there are two ways: *structured wiring* and *short wires*.

Structured wiring

We call a wiring structured when the interconnection wires are of equal length (or there is only a small set of different wire lengths) and are arranged in a simple, regular pattern. Structured wiring ensures interconnection wires with predictable electrical parameters that can be well controlled and optimized. These parameters, in particular low and predictable cross-talk, enables the use of aggressive signalling circuits that can reduce power dissipation and increase propagation velocity (reduce wire latency). Examples of such optimized circuits are pulsed low-swing drivers and receivers. Compared with a full-swing driver, they reduce power dissipation by factor of ten and increase propagation velocity by three times [16].

Short wires

A major part of the power consumption in the CMOS design is due to dynamic power consumption. This power is proportional to the load capacitance [33]. For an interconnect driver its load capacitance is determined mainly by the driven wire capacitance and is proportional to the wire length. Therefore, the energy dissipation due to the interconnect wires is proportional to the wires length, and keeping wires shorter will save energy. Keeping wires shorter is possible by choosing an appropriate interconnection topology and by exploiting locality of communication. Many algorithms display locality and regularity in the required interconnect patterns [2]. In a system performing such an algorithm we expect not all components communicate actively to each other, but active communication will be restricted to several groups of components. Placing components of such a group close to each other will restrict intensive communication on short connections between neighbours. Of course, to achieve this, the interconnection topology has to support such node grouping.

2.2 Cost

The different design decisions that we can take regarding the interconnection network may lead to different area and energy overhead of the network. The space and energy overhead in an on-chip network come from activities on the interconnection wires, the memory needed for the queues, the switching fabric and the routing and arbitration logic.

According to estimations made in [16], area overhead due to the network interface in their on-chip interconnection network is 6.6% in each tile, which is modest. The authors of [16] make a comparison between the available wiring resources in the inter-chip networks and in the on-chip networks. While the inter-chip networks are pin limited (less then 1000 pins for a routing chip), in the on-chip networks they assert it is easy to achieve over 24000 ‘pins’ (on the four edges of 3x3mm tile; 0.1um CMOS with 0.5um min. wire pitch). This is a 24:1 difference showing that on-chip networks have an enormous wiring resource potential at their disposal. Also in [16] an estimation is made about the energy overhead due to wires and interfaces. It has been shown that the wire transition power is significantly greater than the power used in the network interface logic.

As a summary we could say that it is acceptable to trade wiring resources for network performance, and use more complex and clever routing and flow-control mechanisms for saving wire transmission power.

2.3 Scalability

Scalability refers to the possibility to extend the network, while keeping the network’s characteristics unchanged. Adding network’s nodes (e.g. when the next semiconductor technology is introduced) should not affect the remaining nodes in the network. It should not change the throughput of the processing tiles, nor require modifications of the routing algorithm or the flow control mechanism used. Of course, it is not realistic to expect it is possible to keep all network parameters constant, while changing the network size. Changing the size of the network will probably modify the path and the distance that some messages have to travel and this may influence the message latency. However, we would like the variations in the network performance caused by the network scaling to be minimal.

2.4 Real-time behaviour

As HRSoC are used for multimedia applications predictable real-time behaviour is important. A reliable real-time network should display bounded and known message delivery delay in the presence of disturbing factors such as overhead or faults. An additional functional attribute of such a network could be the support for urgent massages, which is allowing some messages to get through ahead of others. Example of urgency classes could be [18]: critical or hard real-time; best effort or soft real-time; and background or non real-time.

2.5 Efficient layout

When choosing the network topology we should take into account that it will have to be implemented in a two-dimensional plane. It is preferable, in this implementation, to place neighbour nodes close to each other in order to keep the interconnection channels as short as possible. Attention should also be paid to the resulting wire density, which has to be uniformly distributed over the 2-D plane, without places of wire congestions.

2.6 Mapping of algorithms

In this context we represent a distributed algorithm as a set of processes that are connected by communication channels. A process is defined as a computation, described by a sequential program, and communication actions (e.g. send and receive) on channels. This representation of a distributed algorithm can be modelled by a graph; we call this graph a *computation graph*. In this graph processes are represented by the nodes of the graph and communication channels are represented by the edges of the graph (Kahn computation model [37], [14]).

The functionality of a computation graph is referred as a task. The computation graph may include hierarchical nodes, which means that on a lower level a node can be represented by a computational graph too. In this way the computation graph represents a system as an assembly of tasks and the root of such a hierarchy of tasks is called application.

To enrich the semantics of this model we can label the nodes and edges of the computational graph. The edge labels can represent requirements of the communication channels such as bandwidth, latency, etc., while the node labels can represent processes' computational requirements.

The on-chip interconnection network should facilitate an efficient mapping of the communication part of the computation graphs. It should provide communication channels that meet the computation graph edges' communication requirements. To be efficient a mapping has to come at a reasonable price i.e. ideally each edge of the computation graph should map to one channel of the SoC interconnection network. Unfortunately, in practise some nodes might have to forward messages on behalf of other nodes. The cost of the mapping can be divided into communication cost between processes, the cost of forwarding messages and initialisation costs. The costs can be expressed in energy consumption, resource usage, and aspects of time (latency, jitter, etc.) [29].

As a summary of this section we can say that for building an on-chip interconnection network we have much more wiring resources at our disposal compared with the inter-chip networks. To be energy efficient we have to choose an interconnection topology that keeps the interconnection wires structured and short, and allows exploiting the communication locality. This topology has to be suitable for mapping computational graphs of the targeted application domain. Since the energy spent for driving the network wires seems to be much more than the energy used for protocol processing, we can afford to use more complex network techniques (flow control, routing, switching, etc.) if they lead to better and efficient utilisation of the wiring resources. For diminution of the design complexity our network has to possess properties like scalability and reusability. For supporting multimedia applications it also has to display real-time behaviour.

3 Related projects

In this section we give a summary for several projects that address the design of on-chip communication networks.

3.1 Pleiades

The Pleiades project at *UC Berkeley* seeks to achieve ultra-low power high-performance multimedia computing through the reconfiguration of heterogeneous system modules. Achieving high-energy efficiency requires the elimination unnecessary actions that typically dominate the energy consumption in general-purpose programmable engines. Providing programmability at just the right granularity (instruction, functional module, data path or gate) makes it possible to eliminate virtually all overhead, while allowing to exploit other energy reducing techniques, such as parallelism, pipelining and dynamic voltage scaling [1].

One instance of Pleiades architectures is the Maia chip. It is a domain specific architecture that integrates processors of different granularity (microprocessor, reconfigurable Dataflow and FPGA), and is targeted towards the Voice Coding domain [2].

The processors are partitioned into clusters of tightly interconnected components. In each cluster a generalized mesh network provides the intra-cluster connections. A second, large granularity mesh network is used for inter-cluster communication. Network connections are set up at (re)configuration time. Processors (nodes) operate on a local clock and the interface between them is self timed with two-phase asynchronous hand shaking [20].

Several interconnection architectures were evaluated for comparison (multi-bus, mesh, hierarchical mesh). The results indicate that the hierarchical generalized mesh delivers best energy-efficiency while maintaining flexibility for heterogeneous reconfigurable systems. A hierarchical mesh reduces the energy consumption by 25% to 50% over a flat mesh interconnection and 5 to 10 times compared to multi bus architecture [2].

3.2 MorphoSys

MorphoSys research project at *the University of California, Irvine* presents a coarse-grain, integrated, reconfigurable system-on-chip targeted at high-throughput and data-parallel applications. It comprises a reconfigurable array of processing cells, a modified RISC processor core, and an efficient memory interface unit [3].

The reconfigurable array is composed of 64 word-configurable processing cells, arranged in 2-D, 8x8 matrix. The array comprises a three-level hierarchical interconnection network. The lowest level of this network is a 2-D mesh, which provides connection to the nearest neighbours, for every cell. Then the 8x8 matrix is divided in 4 quadrants of 4x4 cells. Within each quadrant, each cell can access the output of any other cell in its row/column in the same quadrant, ensured by the second level of the hierarchical network. In the third level of the network (inter-quadrant), there are buses between adjacent quadrants. These buses run across rows and columns and provide data from any one cell in the row/column of a quadrant to other cells in the adjacent quadrant but in the same row/column.

Data processed in the reconfigurable array are supplied by a Frame Buffer, which serves as a data cache. Because there are two sets of Frame Buffers, it is possible to overlap computation in the array and data load in the idle Frame Buffer.

Configuration of the array is stored in the Context Memory. All cells in the row/column share one context (SIMD). The Context Memory contains 32 words. Configuration data can be loaded into a non-active part of this memory without interrupting the array operation [4], [5].

While this architecture exploits data-parallelisms and locality of reference, it is not scalable, because the interconnection network is fitted for this array size.

3.3 NuMesh

NuMesh (*MIT, Laboratory for Computer Science*) is a packaging and interconnect technology supporting high-bandwidth systolic communications on a 3D nearest-neighbour lattice. The

goal is to combine Lego-like modularity with supercomputer performance. To date, the primary focus of the project has been the class of applications whose static communication patterns can be precompiled into independent finite state machines running on each node [6].

In the on-chip network every node contains a processing element and a communication interface. The communication interface is controlled by a RAM-programmable Communication Finite State Machine (CFSM). It allows two physical pipelines to operate concurrently, as each pipeline supports 32 independent virtual threads.

The scheduling is static, at compile time. Although communication paths are scheduled, data need not be sent during every scheduled cycle. A flow control protocol allows empty communication cycles as well as data being buffered in the network (1 word for every thread in the node). The last one is for wormhole routing support [7].

3.4 aSOC

In this work (*University of Massachusetts, Amherst*), a single-chip interconnect architecture, adaptive System-On-a-Chip (aSOC), is described. It provides scalable data transfer and can be reconfigured as application-level communication patterns change. An important aspect of the architecture is its support for compile-time scheduled communication [8].

In this approach each IP Core is supplemented with a small, highly-optimized communication interface. The Core and associated interface form a computational node. Nodes are connected in a mesh. Communications between nodes take place via pipelined, point-to-point connections.

The communication configuration consists of loading an Interconnection Memory in every node with settings for the connection multiplexers. In the operation mode, for every cycle a Configuration Pointer selects from the Interconnection Memory the current interface configuration.

Three DSP benchmarks have been mapped to candidate SoC devices. The described interconnect architecture is shown to be up to 5 times more efficient than bus-based SoC interconnect architectures [9].

The proposed interconnection architecture is easily extendable. Because it limits inter-core communication to short wires and allows exploitation of locality of reference in interconnection patterns, we may expect it to be energy-efficient.

3.5 Raw Architecture Workstation (Raw)

The Raw Architecture Workstation (*MIT, Laboratory for Computer Science*) is a simple, wire-efficient architecture that scales with increasing VLSI gate densities. The Raw architecture's goal is to provide performance that is at least comparable to existing architectures, but that can achieve orders of magnitude more performance for applications in which the compiler can discover and statically schedule fine-grain parallelism (Instruction Level Parallelism) [10].

The approach to achieving these goals is to implement a simple, highly parallel VLSI architecture, and to fully expose the low-level details of the hardware architecture to the compiler so that the compiler or the software can determine, and implement, the best allocation of resources for each application. Eliminating a fixed instruction-set interface between the compiler and the hardware, Raw is composed of a set of interconnected tiles, each tile comprising an instruction memory, a switch-instruction memory, data memory, an ALU, FPU, registers, and a programmable switch [10].

The tiles interconnect use four 32-bit full duplex on-chip networks. Two networks are static (routes specified at compile time) and two are dynamic (routes specified at run time). Each tile only connects to its four neighbours. Every wire is registered at the input to its destination tile. This means that the length of the longest wire in the system is no greater than the length or width of a tile. This property ensures high clock speed, and the continue scalability of the system [13].

The static networks provide single-cycle-per-hop latencies and can route two values in each direction per cycle. The dynamic networks use dimensional-ordered, wormhole routing. The messages consist of a single word header (destination, user field and message length) and

up to 31 data words. Travelling through the network takes one cycle per hop for a word, plus an extra cycle for every hop in which the message makes a turn. Because deadlock avoidance algorithms are based on restriction of network usage here a deadlock recovery approach has been preferred.

4 Efficient on-chip interconnection networks

This section provides an overview of issues related to on-chip communications for HRSoC systems. The following network aspects are reviewed: *interconnection network topology*, *switching*, *routing, flow control, queuing and scheduling*.

4.1 Interconnection network topologies

The topology of an interconnection network defines how the nodes (processing tiles) are interconnected via channels to other nodes. Usually the topology is represented as a graph. We start with some definitions used for characterising and comparing of the network topologies:

- *Node degree* – the number of channels that connect the node with other nodes; if all nodes have the same degree the network graph is called *degree regular*;
- *Path* in the graph – a set of consecutively connected nodes in the graph;
- *Route* – a path between two nodes;
- *Distance* between nodes – the length of the shortest path between the nodes (in number of channels that have to be passed);
- *Diameter* of the network – the maximum distance over all pair of nodes in the network;
- *Bisection width* of the network – the minimum number of channels that must be removed, or cut, to partition the network into two sub networks, each containing half the nodes in the network;
- *Channel width* – the number of bits that can be transmitted in parallel on a physical channel between two adjacent nodes;
- *Bisection density* – the product of bisection width and the channel width, usually used as a measure of the network cost.

Below we give a classification tree of the most popular interconnection topologies for multiprocessor architectures.

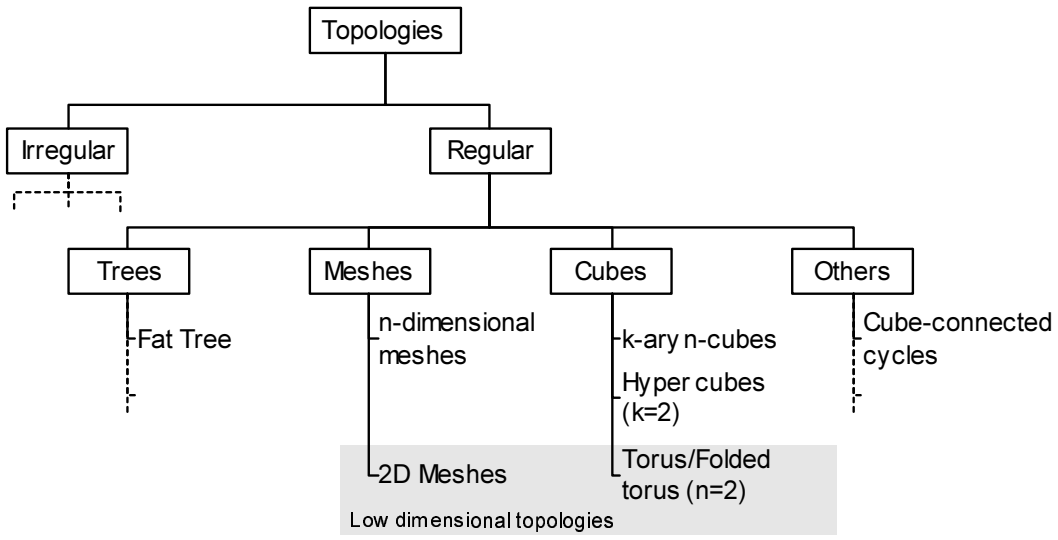

Figure 2: Topologies classification

Next, several basic, often used and well-understood network topologies are reviewed.

4.1.1 Point-to-Point

In this interconnection scheme a physical connection between two nodes is made if there is a need of communication between them in the computation graph. The scheme is free of problems like arbitration, congestions and so on. While this is a good approach for simple designs, for non-trivial designs routing complexity becomes unmanageable and leads to wire congestions. Because every connection is dedicated to one communication channel only and usually most of the communication channels are not used permanently, connections are weakly utilised. Additional disadvantages are the irregular topology, unpredictable wires (unknown at design time, dependent on the layout) with different length and non-constant node degree.

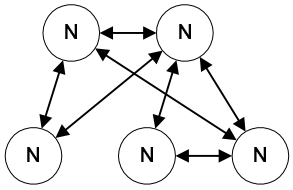

Figure 3: Point-to Point interconnection

4.1.2 Bus

The bus concept facilitates modularity by defining a standard interconnection interface. It has a simple topology and is routable, since only a small bundle of wires need to traverse the entire chip. Because the communication medium is shared between all nodes, it is well utilised. But it also has severe disadvantages.

First, it is *non-scalable*, because its bandwidth is shared between all nodes and adding new nodes will decrease communication throughput of the other nodes [21]. Adding new nodes will also change the electrical parameters of the medium, because of the changing drivers/receivers capacitances.

Second, it *does not support multiple concurrent communications* and it would be difficult for bus-based systems to exploit algorithms' parallelisms [16].

Third, it is *not energy efficient*. A bus consists of long wires spanning the entire chip and to drive them we need much energy, and it decreases the speed. In addition all messages are transmitted to all connected nodes, although in most of the cases they are destined to only one of them. This broadcast mechanism does not exploit communication locality to save energy.

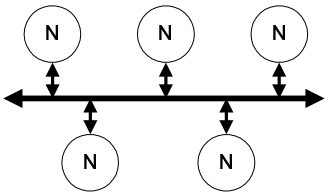

Figure 4: Bus topology

4.1.3 Multi-bus

The Multi-Bus partly solves some of the problems of the bus. The Multi-bus consists of several independent local buses interconnected through bridges. Such a bus partitioning keeps wires relatively short and so reduces the consumed power. In [22] it is asserted that a multi-bus could save from 16% to 50% power compared to a global bus, depending on the characteristics of the data transfer among the nodes and configuration of the multi-bus. Separated local buses also exploit communication locality and the use of multiple concurrent communications.

Although the multi-bus has some advantages with respect to the global bus, it is still non-scalable, energy inefficient and suffers from arbitration problems when the number of nodes increases.

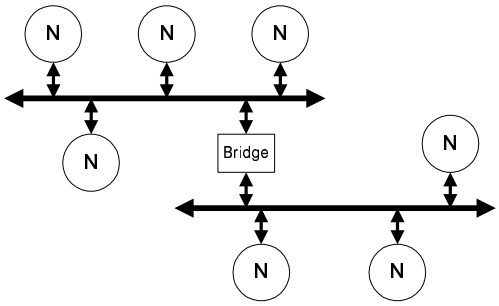

Figure 5: Multi-bus topology

Typically most of the network nodes under-utilise the bandwidth of their network interfaces and their local interconnections. Hence, it can be reasonable to use a cheaper nodes' network interface and shared medium for the local interconnections. An option is to use local buses that offer a simple network interface (in term of area) and shared communication media. Small local buses can be used as a first level of a hierarchical interconnection network, and on this level they can intra-connect groups of nodes. In the second level of the network hierarchy a switched network interconnects the local busses. In this way area can be saved by using simple network interfaces, well utilise the local buses' bandwidth and gain a significant reduction of the number of network switches [27].

4.1.4 Trees

Tree networks have a nice property that the diameter is O(log n). Unfortunately the bisection width is small (=1). This means that a tree is not fault-tolerant and performance problems could be expected in the root of the tree. To avoid these problems some improvements of the tree networks are suggested: fat-tree, full-ring binary trees, X-tree [18].

In the fat-tree for example, going towards root the link capacity is doubled in every level. This solves the performance problem, but introduces switches with different size and makes the placement irregular.

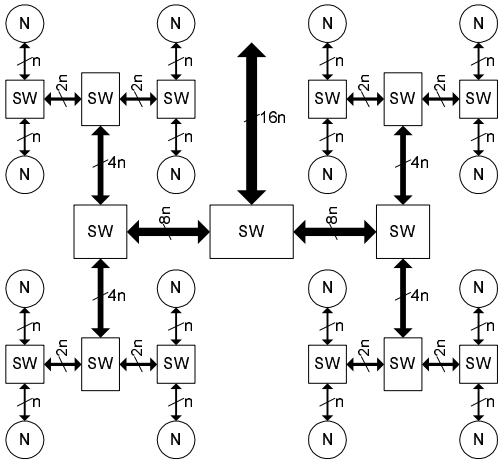

Figure 6: Fat-tree topology

4.1.5 n-dimensional Meshes

Most of the popular direct network topologies fall in the category of either n-dimensional meshes or k-ary n-cubes because their regular topologies simplify routing.

In n-dimensional mesh the nodes are arranged in n-dimensional array. Along each dimension there are k_i nodes ($k_i \geq 2$), where $0 \leq i \leq n-1$ is the dimension number. So the total number of nodes in the mesh will be $k_0 \times k_1 \times \ldots \times k_{n-2} \times k_{n-1}$. Each node x is defined by n coordinates, $\sigma_{n-1}(x)$, $\sigma_{n-2}(x)$,… , $\sigma_1(x)$, $\sigma_0(x)$, where , $0 \leq \sigma_i(x) \leq k_i-1$ for $0 \leq i \leq n-1$. Two nodes x and y are neighbours iff $\sigma_i(x) = \sigma_i(y)$ for every i, $0 \leq i \leq n-1$, except one, j, where $\sigma_j(x) = \sigma_j(y) \pm 1$. Thus, nodes have from n to$2n$ neighbours, depending on their location in the mesh. In the mesh there is a direct connection only between neighbours [30].

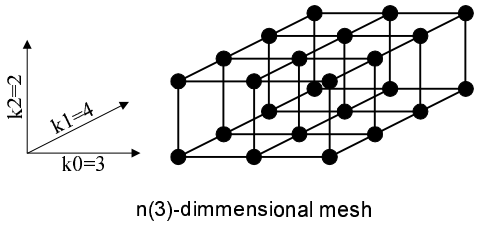

Figure 7: An example of 3-dimensional mesh

A problem in the mesh networks is that if assuming uniform traffic between nodes, the channels near the centre of the mesh are likely to experience higher traffic density than channels in the periphery.

4.1.6 k-ary n-cube

In k-ary n-cubes the nodes are arranged in n-dimensional array too, but here, as the name ”cube” hints, in each dimension there is an equal number of nodes k, so $k_0 = k_1 = \ldots = k_{n-2} = k_{n-1} = k$. The total number of nodes is k^n. Another difference with the n-dimensional meshes is the definition for neighbour nodes (here again the direct connections are only between neighbours). In k-ary n-cubes two nodes are neighbours iff $\sigma_i(x) = \sigma_i(y)$ for every i, $0 \le i \le n-1$, except one, j, where $\sigma_j(x) = (\sigma_j(y) \pm 1) \bmod k$. The use of modular arithmetic in the definition results in wraparound channels, which are not presented in the n-dimensional mesh. They double the bisection width and reduce the network diameter in respect to the mesh. If $k > 2$, then every node has $2n$ neighbours, if $k = 2$, then every node has n neighbours [30].

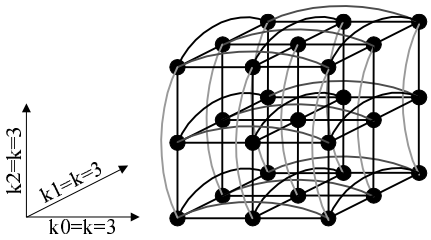

Figure 8: An example of 3-ary 3-cube

This topology gives a good compromise between different goals and is well understood. Many graphs e.g. ring, mesh, butterfly network can be embedded in a k-ary n-cube and as a consequence there is a large class of algorithms that have one-to-one mapping onto this topology. It has a recursive structure as well [18].

Unfortunately, 2D implementation of such a topology leads to inefficient layout with many high wire density spots [23]. In addition, the dependency of the node degree on the number of cube dimensions, n, makes this type of networks not scalable.

4.1.7 Hypercube

The hypercube (or binary/boolean cube) is a symmetric network and a special case of both n-dimensional mesh and k-ary n-cube. A hypercube is an n-dimensional mesh in which $k_i=2$ for all i, $0 \le i \le n-1$, that is, a 2-ary n-cube.

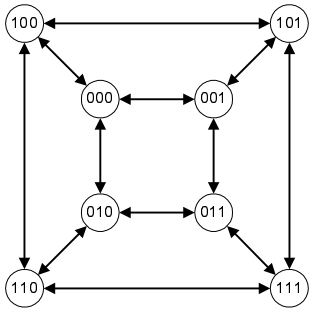

Figure 9: 3-dimensional hypercube

Many graphs e.g. ring, mesh, butterfly network can be embedded in a hyper-cube and as a consequence there is a large class of algorithms that have one-to-one mapping onto hyper-cubes. It has a recursive structure as well [18].

A disadvantage of the n-dimensional mesh, k-ary n-cube and hypercube, as their special case, is the variation of the node degree with variation of the number of dimensions *n*. To resolve this problem for a hypercube, several constant degree networks which can emulate it are proposed: Cube-Connected Cycles, Butterfly Network and Shuffle-exchange network [18].

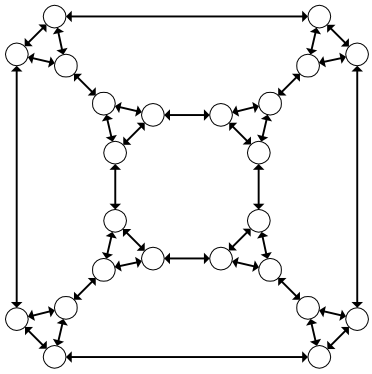

Figure 10: Cube connected cycles

A disadvantage is that high dimensional meshes and cubes are inconvenient for 2D implementation.

4.1.8 Low dimensional networks

When *n=2* n-dimensional mesh reduces to 2D mesh and k-ary n-cube reduces to 2D torus. This dimension number is convenient for planar implementations. To remove wraparound connection in the torus a folded torus is presented [23]. When *n=1*, the k-ary n-cube collapses to a ring with *k* nodes.

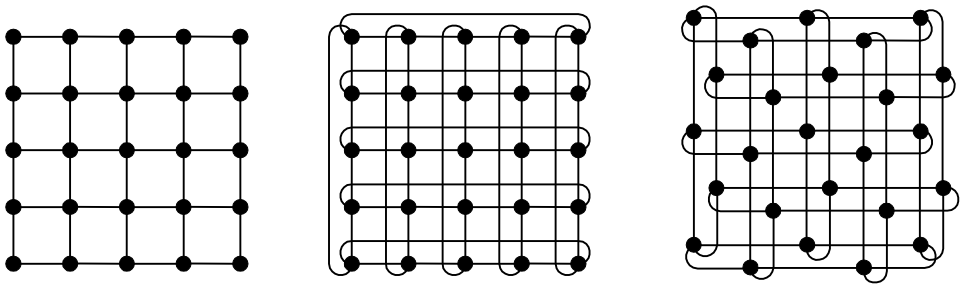

Figure 11: Examples for 2D topologies

In [23] a performance analysis is made of high-dimensional cubes with narrow interconnection channels against low-dimensional cubes with wide interconnection channels. The results are in favour of the low dimensional cubes, which show less wire density, low latency (especially when communication locality is exploited) and higher hot-spot throughput.

In [16] the same author presents an on-chip interconnection network that implements such a low dimensional topology with wide interconnection channels. In this design 16 tiles are interconnected in a folded torus of size 4x4, and interconnection channels are 256-bit wide.

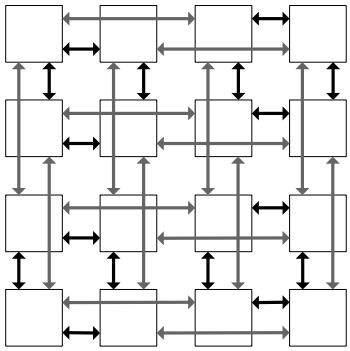

Figure 12: Folded torus 4x4

When the folded torus is of size 4x4 we have for every node two connections with near neighbours and two connections with far neighbours. This allows for communication locality. Unfortunately, when the network scales up, the number of nodes that have connections with two near neighbours stays a constant (16), and increases the number of nodes that have one or do not have connection with near neighbours.

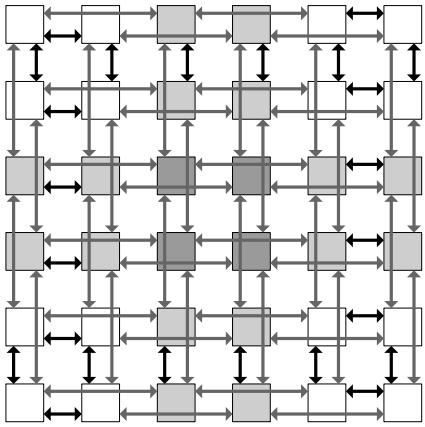

Figure 13: Folded torus 6x6

A comparison between power efficiency of the folded torus topology and the mesh topology is made in [16]. In this paper the total power required to send a flit through the network is decomposed into two parts – *nodes power* and *wires power*. The nodes power expresses the power consumed for passing a flit through the nodes on its path (for a flit passing from a node's input to a node's output), and the wires power expresses the power dissipated for passing a flit through the wires on its path. The mesh is shown to be more power efficient than the folded torus if wire transmission power dominates over power dissipated in the nodes.

In most of the projects that target HRSoC a mesh topology is chosen as an on-chip interconnection topology. A mesh topology allows exploiting locality of communication, it is scalable, regular and is shown to be energy efficient [2].

As a summary of this section the next table shows how each of the topologies presented fulfils our criteria. These criteria are: scalability of the topology; efficiency of the topology for a 2D-plane layout; availability of algorithms whose process graphs are naturally mapped on the topology and energy-efficiency of the topology.

		Interconnection							
		Point-to-Point	Bus	Multi-bus	Trees	n-dimensional Meshes	k-ary n-cube	Hypercube	Low dimensional topologies
Criteria	Scalability	no	no	no	no	no	no	no	yes
	Efficient layout	no	yes	yes	?	no	no	no	yes
	Mapping of algorithms	yes	?	?	?	yes	yes	yes	no
	Energy-efficiency	yes	no	yes	?	no	no	no	yes

Among the interconnection topologies reviewed above the *Low dimensional topologies* seem to be most suitable for on-chip implementation. They are the ones that fulfil the requirements defined in the previous section best – they result in an area-efficient and energy-efficient network layout and are easily scalable. A disadvantage is the lack of symmetry and the small set of computation graphs that can be mapped naturally on them, but we believe that this can be overcome by adding more flexibility in the network routers. For example using virtual channels will allow a physical channel to be shared between several application channels, which simplify the application mapping.

4.2 Switching techniques

Switching is the basic mechanism to move data from a source node to a destination node. There are a number of switching techniques that can vary between two extreme cases – circuit switching and packet switching.

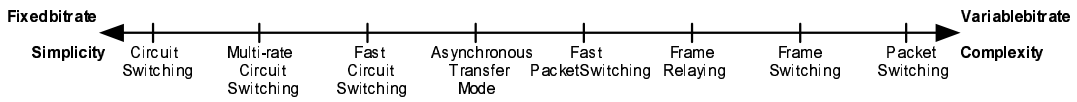

Figure 14: Switching techniques distribution

4.2.1 Circuit switching

In circuit switching (or Synchronous Transfer Mode - STM), all switches multiplex connections between them synchronously. To provide a channel between the end nodes the network switches, which are in between source and destination, connect their respective input to their respective output. When connection between nodes is made it stays active for a fixed time period called time slot. For a circuit switching network time is a sequence of timeslots and for every time slot the network provides connections between different end nodes. So every time slot the network applies a different interconnect pattern. After a constant number of time slots the pattern sequence starts from the beginning and repeats. The sequence of repeated time slots is called frame. Thus, connection between a pair of end nodes is made cyclically every frame - each connection has its own time slot that arrives at certain frequency. In the frame several connections can use different slots concurrently. A connection uses the same time slot in each frame cycle [18].

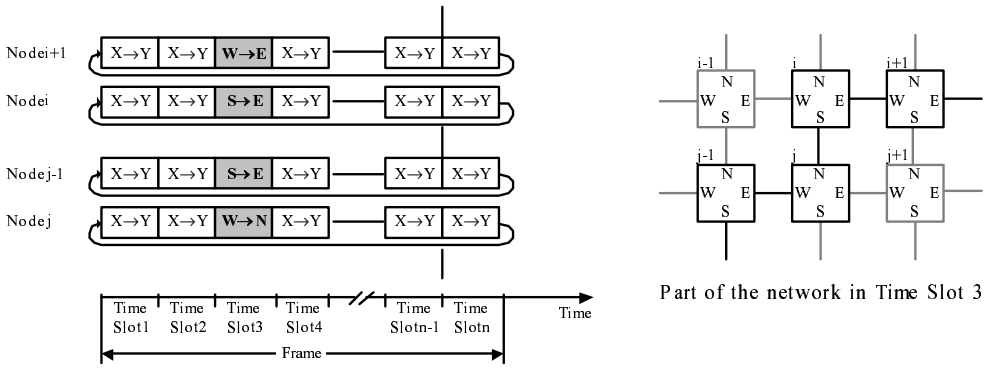

Figure 15: Example for circuit switching

This system has small and bounded connection delay and low jitter. Connection bandwidth is fixed and guaranteed. Disadvantages are the long set-up time, the need of a global time or frame reference, waste of unused channel bandwidth and inflexibility.

4.2.2 Multi-slot circuit switching

Multi-slot circuit switching operates in almost same way as circuit switching. The difference is that here applications are allowed to claim for more than one slot for a single connection. In this way the basic rate can be kept lower, which is more efficient. Of course this adds complexity to the switches [18].

4.2.3 Multi-rate circuit switching

Here, in the frame there are several sets of time slots with different size. Instead of claiming for several slots, application claims for a slot with a size that satisfies its bandwidth requirements. Although this is a solution to the inefficiency problem, the problem of fixed size of time slots still exists [18].

4.2.4 Fast circuit switching

This is a circuit switching scheme again, but here network resources are allocated only when information needs to be sent, and released again after the information burst is over. When connection is being established, the user claims a part of the available bandwidth. The system will not allocate the resources needed at that time, but will store information about them. When the source starts sending information, switches along the path are requested to allocate the agreed resources immediately. Resources do not stay permanently allocated to the connection, but only until they are needed [18].

Fast circuit switching is more adequate for dealing with fluctuating and burst data transfers. But the dynamic allocation of resources does not give guarantees that requested resources will be available at the moment of start of sending data. So there could be a situation in which the source will have to wait. Another drawback is design complexity and the signalling needed for allocation of resources.

4.2.5 Packet switching

In the Packet switching, data is transported from source to destination in the form of packets. The user's data that will be transferred is encapsulated in packets, generally of variable size. Apart from the user's data these packets contain additional information, which is used in the network for routing, flow control, error correction, etc. The packets are transferred asynchronously in the network, which means that every node makes its own decision when to send a packet. When a packet is ready to be sent the source tries to access the channel. When the channel is obtained all bandwidth is used to transmit the packet [18].

This switching scheme is more flexible with respect to bandwidth allocation. It can support dynamically varying bandwidth requirements up to maximum channel capacity. Bandwidth is allocated only when the packet is ready for transmission and remains allocated only for the time needed for packet transfer.

All these advantages come at the price of higher complexity of the protocol and protocol processing. The higher complexity increases the area and lowers the speed of the protocol processing circuits, but the technology advances make this disadvantage less critical and the advantages of this technique out weighted it. A more critical weakness of the packet switching is the absence of bandwidth guarantees. Required bandwidth is not guaranteed and some times a request has to wait, which leads to varying delay.

4.2.6 Dynamic TDM

The concept of Dynamic Time Division Multiplexing (DTDM) is a combination of conventional TDM and packet transmission techniques. It can be viewed as a transmission format for slotted packets to traverse synchronous digital pipes. In contrast to TDM the slots do not have a specific position in the DTDM frame. The fixed length slots permit a simplified buffering in the switches and ease the bandwidth allocation among different users. DTDM overcomes the inefficiency of TDM by allocating time slots only for active cannels [18].

4.2.7 Asynchronous Transfer Mode

The ATM switching is an advanced version of packet switching. Here, data is transported in small, fixed size packets, called cells. The cells are transferred from node to node on logical links called virtual cannels (VC). On one physical link many logical links reside. Each cell has a header that contains a field, called Virtual Cannel Identifier (VCI), which determines to which VC the cell belongs.

ATM switching is based on a connection oriented service. In a communication session there are three phases: *connection set-up*, *communication* and *connection release*.

In the *connection set-up phase* a path between source and destination is determined and bandwidth along the path is reserved. On each link, a virtual cannel number identifies the connection. Thus, the path between source and destination is described by a sequence of virtual channel numbers between the hops. If a bidirectional path is needed, two separate VC have to be reserved. If the requested bandwidth is not available, the connection is refused.

In the *communication phase* a node is allowed to send cells on the virtual cannel. The number of cells than can be sent is limited to reserved the bandwidth. In each hop the cell is buffered and its header is processed, which cause slight time delay. Header processing consists of header examination and header translation (the incoming VCI is replaced by outgoing VCI for next the hop).

In the *connection release phase* bandwidth and VC numbers are released to be re-used by other connections.

ATM is a high speed packet switching method, which has the ability to support combination of constant and variable bit rate traffic. It has low per-hop delay.

Cells contending for an output link have to be buffered, which brings problems as buffer overflow and queuing delay. However these problems could be reduced with suitable bandwidth allocation mechanism and a cell-priority scheme [18].

The next table gives a summary of the switching techniques presented. The criteria toward which these techniques are compared are:

- Flexibility of the technique to handle dynamic variations in the data flow;
- Type of bandwidth allocation – static (in compile time) of dynamic (in run time);
- Is the bandwidth allocated guaranteed;
- Relative complexity of the implementation of the techniques;

		Switching technique						
		Circuit switching	Multi-slot circuit switching	Multi-rate circuit switching	Fast circuit switching	Packet switching	Dynamic TDM	Asynchronous Transfer Mode
Criteria	Flexibility	low	low	low	medium	high	high	high
	Bandwidth allocation	static	static	static	?	dynamic	dynamic	dynamic
	Bandwidth guaranties	yes	yes	yes	yes	no	?	yes
	Switch complexity	low	low	low	high	high	high	high

None of the overviewed switching techniques seems to be an optimal solution for our network. The circuit switching based techniques are simple, fast, guarantee the bandwidth allocated and need minimum protocol overhead, but in general are quite inflexible and under-utilise the network resources. This technique is quite well suitable for streaming communication patterns. The bandwidth guarantees highly simplify the realization of real-time network behaviour, but the inflexibility may restrict the set of suitable applications and complicate the compiler if all inter-process communications have to be reduced to static communication patterns, scheduled on before hand at compile time.

Not less important is the fact that all routers in a circuit switching network have to work synchronously. With the advance of the technology ensuring signal integrity becomes possible only for short distances. As the on-chip network spans over the entire SoC to ensure clock distribution to all routers under these conditions will be very difficult task.

The packet switching oriented techniques, on the other hand, offer high flexibility, dynamism and much better utilisation of the network resources, but are more complex, add more protocol overhead and most important, do not allow bandwidth reservation. The last problem can be partly solved using priorities. Defining a class of high-priority messages will give some preferences to the members of this class and their priority will serve as a guarantee. Because the targeted SoC is supposed to work in a highly dynamic and unpredictable environment and often applications naturally communicate via messages it seams more reasonable to choose a packet switching approach.

4.3 Flow control mechanisms

Flow control deals with the allocation of channel and buffer resources to a packet as it travels from source to destination. Collisions might occur when a packet cannot proceed because some resource it needs is held by another packet. In this case the packet can be blocked in place, buffered or routed through another channel. This decision depends on flow control policy [30]. Below we present some popular flow-control mechanisms.

4.3.1 Store-and-Forward

In the case of *Store-and-Forward* flow-control a message is first received in each intermediate network node in its full length before transmission to the next node is started. If L is the message length, W is the channel width and T_C is the channel cycle time, then the transmission time for single message through n consecutive channels is:

$$T = \left(n \cdot \frac{L}{W} \right) \cdot T_C$$

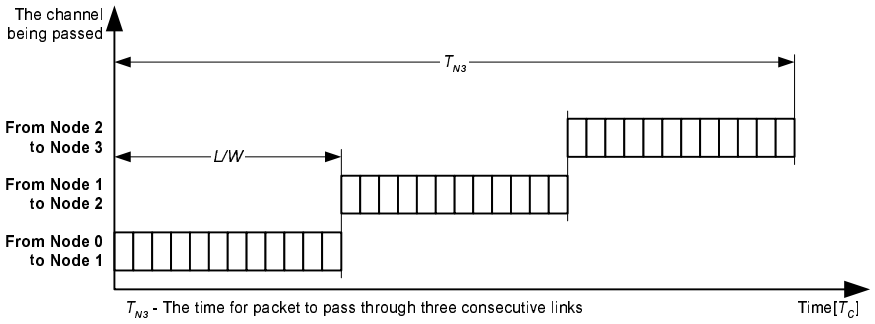

Figure 16: Store-and-Forward routing latency

This method has a latency that is proportional to the packet size. It also imposes significant buffer space requirements.

4.3.2 Wormhole routing

Wormhole routing divides a message into fixed size portions, called flits (flow control digits). Instead of storing the full packet in each intermediate node and then forwarding it, here only one (or a few) flit is stored and forwarded. As flits are forwarded, the packet is pipelined over the channels between the source and the destination.

Flits are smaller then a packet, which leads to improvements in storage and bandwidth allocation. Buffers are smaller even for large packet sizes.

Here the transmission time for single message through *n* links will be:

$$\left((n-1) + \frac{L}{W} \right) \cdot T_C$$

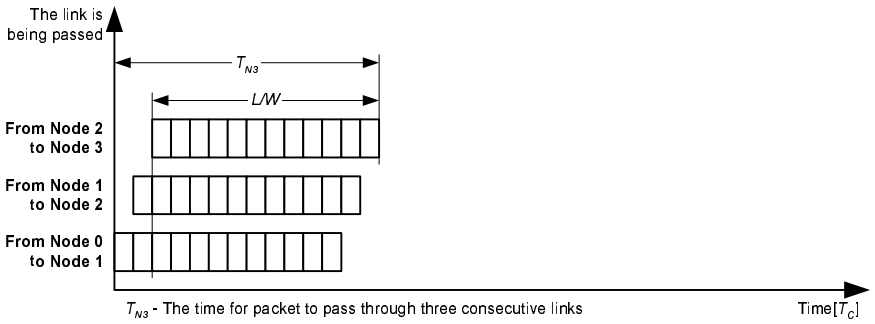

Figure 17: Wormhole routing latency

The disadvantage of wormhole routing is the inefficient use of the channel bandwidth. The physical channels through the path of the packet are held for the duration of the entire packet transmission. If the packet is blocked, all these physical channels are idle and other packets cannot use them. Therefore throughput decreases.

4.3.3 Virtual Cut-Through

Virtual Cut-Through is a method similar to wormhole routing. It behaves in the same way as wormhole routing in the absence of contention. It differs from wormhole routing in that it buffers messages when they are blocked, removing them from the network. This improves the hot-spot performance of the network and increases the throughput. However, message flow control is performed at the packet level and storage and bandwidth are allocated in packet-sized units [34].

4.3.4 Nosy worms

In this method each message attempts to establish a connection with the destination by forming an unbroken path across intermediate nodes. If a block or a failure is encountered along the route, the message will give up by recalling back to the sender, thus avoiding deadlock. After a non-deterministic delay the message tries again. Once a connection is made, the entire message is transferred from source to destination.

The method seams to be expensive in terms of communication cost and it will go worse with increasing the network diameter. Repetitive connection attempts are inefficient in terms of energy too. It is also difficult to predict how in the network with large diameter the delivery of a packet may be guaranteed. A packet may indefinitely try to find its way to the destination node and always fail half through [35].

4.3.5 Virtual channel flow control

In the methods presented above, to each physical channel a buffer is associated with a FIFO organization to store input flits. Flits do not contain routing and sequencing information. This information is kept in the source node for the packet using the channel. Therefore the input buffer can contain only flits from a single packet. If this packet becomes blocked, the physical channel is idle because no other packet is able to acquire the buffer resources needed to access the channel.

Virtual channel flow control [25] overcomes this problem. It decouples allocation of buffers from allocation of channels by providing multiple buffers for each channel in the network. Each of these buffers can hold one or more flits of a packet and has associated packet state information. If a packet currently using the physical channel becomes blocked other packets can allocate another input buffer and start using the physical channel, thus bypassing the blocked message. In this way several packets can use a physical channel simultaneously and share its bandwidth. Each of these packets is associated with a logical link called Virtual Cannel (VC). So several VC can reside on one physical link and the number of this VC is smaller or equal to the number of buffers associated to the physical connection.

Adding virtual cannel flow control to the network makes more efficient use of the most costly resources in an interconnection network, the channel bandwidth. The cost paid is a small amount of additional logic and buffer space.

4.3.6 Wave switching

Wave switching [26] is a hybrid between circuit switching and wormhole flow control. Every node in the network contains several separate switches, all of them working independently, implementing the same topology and having their own set physical channels. Actually there are several, parallel existing networks and every switch in the node belongs to a different network. One of these networks (switches in the node) implements wormhole routing with virtual channels flow control. The rest of the networks (switches in the node) implement pipelined circuit switching. For every circuit switch in the node there is a dedicated virtual channel in the wormhole switch called control channel. Setting up or teardown a connection in a circuit switch is made by sending a special control flit on the respective control channel of the wormhole switch.

The circuit switching networks are used for pre-established connections, scheduled in compile time, while the wormhole network is used for sporadic and unpredictable message exchange. Connections in the circuit switching networks are dedicated only to one communication channel, they offer high bandwidth, low latency communication end are deadlock and congestion free.

This method in fact combines two network types – circuit switching and wormhole network. Because the circuit switched channels are dedicated only to a single data link, we can expect these channels to be under-utilized. Also the presence of several switches in a single node will raise the node complexity. The flexibility provided by this approach comes at a price of inefficient usage of lot of resources.

4.3.7 Flit-Reservation flow control

The idea of this method [24] is to schedule the buffers and channels of the network dynamically, but in advance. To achieve this, the packet control information is separated from

the packet data and travels on a separate network (control network, which is 4 times faster than the data network) in form of control flits. The control flit travels ahead in advance of the data flits. A control flit received in a router carries information about the destination and the arrival time of up to N data flits, which are expected to arrive. This information allows the router to schedule in advance buffers and channels for data flits. Data flits travel on a data network and they contain only payload.

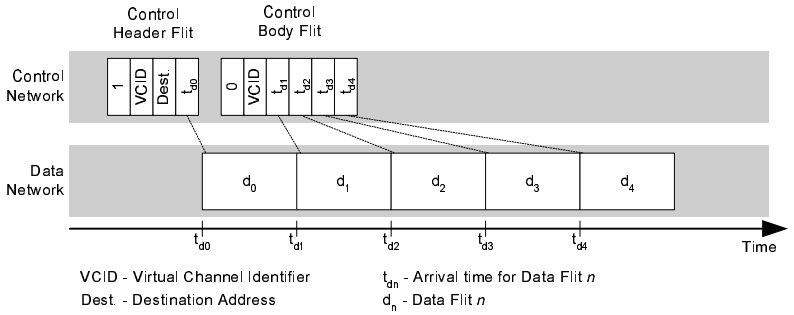

Figure 18: Flit-Reservation flow control

When a control flit arrives at a router, it is processed in three phases: routing, output scheduling and input scheduling.

In the *routing* phase the output port to which the expected data flits should be forwarded is determined.

In the *output scheduling* phase, for that output port, it is decided at what time the expected data flits will be forwarded to the next node. A credit based scheduling is used. For each output the output scheduler knows:

- The arrival time of the data flit that will be forwarded (this information is taken from the control flit);
- The clock cycles (from the present, ahead to the scheduling horizon) for which output channel is already reserved and for which it is free;
- The available buffer space in the next node from now to the scheduling horizon (credits back are sent in advance).

A data flit is scheduled to leave a node as soon as possible, which means at the first clock cycle after its arrival, for which the output channel is free and there is available buffer space in the next node.

When all expected data flits, for which a control flit has notified, are scheduled and their departure time is known, the control flit is updated with the new data flits' arrival times (in the next node) and forwarded to the next node. The new arrival time in the next node, t_a, for a data flit is:

$$t_a = t_d + t_p$$

Where:

t_d is data flit departure time;

t_p is data flit propagation time to the next node;

Input scheduling is the next phase of the control flit processing. Now, for every data flit expected the arrival time and departure time are known. These two moments determine the period of time for which a data flit is going to be buffered in the node. Buffer space is reserved for this period and a credit is send back to the previous node to update the buffer availability in its output scheduler.

This method uses buffer space efficiently because buffers are held only during actual buffer usage. It also eliminates routing and arbitration latency due to the advance scheduling. These advantages can be achieved by using of statically scheduled flow control, but the difference is that here flexibility of the dynamic routing is saved.

The flow control mechanisms presented are summarised in the table below. The comparison criteria are:

- Relative message latency achieved by the mechanism;
- Utilisation of the queues' buffer space;
- Utilisation of the network's interconnection channels;
- Relative complexity of the mechanism implementation;

		Flow Control mechanism				
		Store-and-Forward	Wormhole routing	Virtual Cut-Through	Virtual channel flow control	Flit-Reservation flow control
Criteria	Latency	high	low	low	low	low
	Buffer utilisation	low	high	low	high	high
	Channel utilisation	low	low	low	high	high
	Circuit complexity	low	low	low	medium	high

Form the flow control mechanisms reviewed above we believe that a combination of *wormhole routing* and *virtual channels flow control* is most suitable for an on-chip network. It offers high flexibility and good resources utilisation while the switch complexity stays moderate. The flexibility of wormhole routing is expressed in the ability to handle data on a flit level, which facilitates the design of fine grain communications mechanisms. These fine grain communication mechanisms in combination with low data latency (due to the virtual channels) and a possible allocation of priorities to virtual channels will also help in approving the real time behaviour of the network.

4.4 Routing

In a network every node must be able to send messages to every other node. The routing algorithm specifies what path a packet takes as it travels from the source node to the destination node. Efficient routing is critical for the energy performance of networks.

Looking at routing algorithms we can distinct several characteristics that can be used for routing classification. We present these characteristics below:

1. Depending on where the routing decision is taken we can have two types of routing [30]:
    - *Source routing* – the source node selects the entire path before sending the packet. Each packet has to carry this routing information and this increases the packet size. The path cannot be changed after the packet has entered the network.
    - *Distributed routing* – each router takes the decision where a received packet has to be sent to (to the local processor or to one of the node neighbours). The time for taking the routing decision strongly influences network latency. In this approach each node needs a routing table, or routing algorithm.
2. Depending on what the routing decision is based on (what information is used to take a routing decision) we can have *deterministic (or oblivious) routing* and *adaptive routing* [30]:
    - *Deterministic (oblivious) routing* – the path is completely determined only by the source and destination addresses.
    - *Adaptive routing* – the path for a particular packet is determined by the source address, destination address and the dynamic network conditions, for example presence of faulty or congested channels. So, which path will be chosen for a packet depends of the current condition in the network. To take

such a decision routers need global network information. Providing this information to each router creates additional traffic and requires additional storage space in each router.

3. Depending on the path chosen we can have *minimal routing* and *non-minimal routing* [30]:
    - *Minimal routing* – the selected path is one of the shortest paths between the source and destination nodes (there could exist several shortest paths). In this path every next channel brings the packet closer to the destination.
    - *Non-minimal routing* – algorithm allows packets to follow a longer path, usually in response to current network conditions. If non-minimal routing is used, care must be taken to avoid a situation in which the packet will continue to be routed through the network but never reach the destination (*livelock*).

These three characteristics of the routing algorithm can be seen as three dimensions in the space of routing algorithms, although some combinations are highly improbable (e.g. deterministic, non-minimal). A schematic representation of the classification is presented in the next figure.

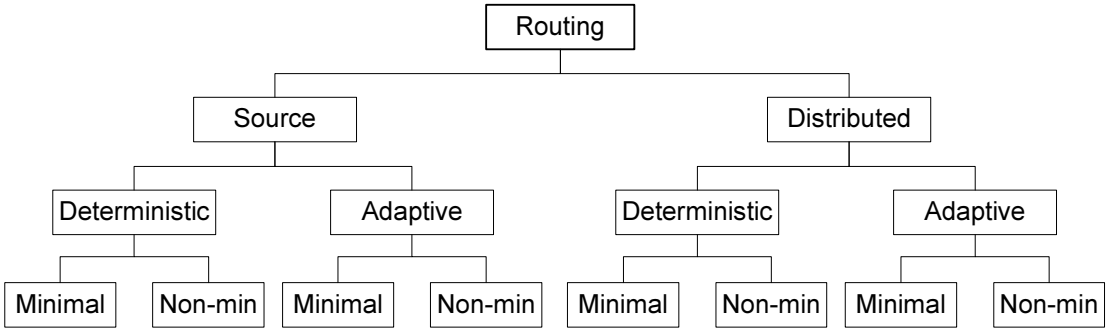

Figure 19: Routing classification

4.4.1 Routing problems

During the routing process several problematic situations might occur:

- *Deadlock* – occurs when a packet waits for an event that cannot happen (usually for freeing network resources);
- *Livelock* – occurs when the routing of a packet never leads it to its destination (this is possible only when routing is *adaptive* and *non-minimal*).
- *Indefinite postponement (fairness)* – occurs when a packet waits for an event that can happen but never does. For example, a packet may wait forever to acquire a network resource for which other packets are always competing successfully.

A routing algorithm must be constructed in such a way that it avoids above unfortunate situations – hence it must be *deadlock free*, *livelock free*, *indefinite postponement free*. As other common requirements toward the routing algorithms we can mention the *low communication latency*, *high network throughput* and *easy to implement*.

Deadlock

In this section we will give an overview of mechanisms to tackle the deadlock problem. Deadlock is a situation that can occur due to distribution of shared resources. In an interconnection network examples of shared resources are buffers or physical links. In the nodes messages are stored in buffers. For a message to be transferred from node A to node B it needs: to allocate buffer space in B; to cross the channel between node A and node B; and to free buffer space in node A. We can express this in a graphical form, called resource dependency graph.

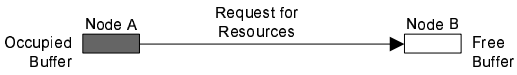

Figure 20: Example resource dependency graph

In a network a circular situation can arise like shown below:

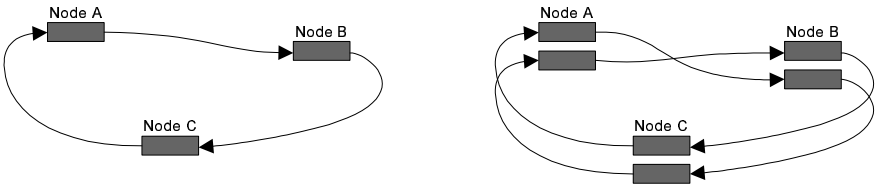

Figure 21: Example deadlock situations

These figures show circular dependencies between resource requests. This is a deadlock situation. It is expressed in a permanent impossibility for allocation of required resources. In a graph representation a deadlock is seen as a cycle in the graph [28]. There are two ways to solve the deadlock problem – *deadlock pre-emption* and *deadlock avoidance* [30]:

- *Deadlock pre-emption* – in the scenario it is allowed to pre-empt packets involved in a potential deadlock situation. Pre-empted packets can be either rerouted or discarded. The former policy gives rise to adaptive non-minimal routing techniques. The later policy requires that the packets be recovered at the source and retransmitted (energy inefficient). Because of requirements for low latency and reliability, packet pre-emption is not used in most direct network architectures;
- *Deadlock avoidance* – in this approach deadlock is avoided by the routing algorithm (more common). The methods for deadlock avoidance, presented below are based on preventing cycles in the resource dependency graph.

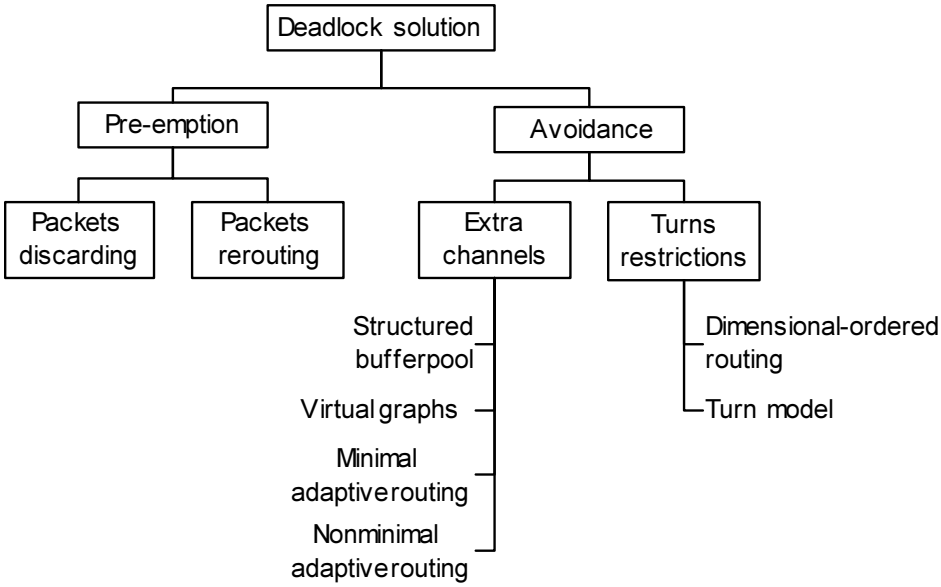

Figure 22: Classification of the solutions for the deadlock problem

In the following section the following deadlock avoidance methods are reviewed: structured buffer pull, virtual graphs, dimension-ordered routing, and the turn model.

4.4.2 Structured Buffer Pool

Let a network graph have a diameter M and every node is assigned M+1 buffers. If in the resource dependency graph each edge connects buffer number *n* in the source node with buffer number *n+1* in the destination node then this network is deadlock-free [18].

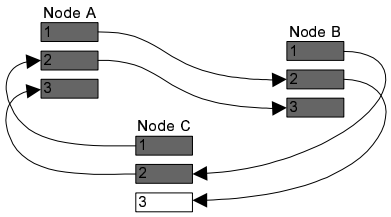

Figure 23: Structured Buffer Pool – deadlock free flow control solution

Note: buffer number 1 can be used to inject messages in the network, and buffer number 3 can be used to extract messages from the network.

4.4.3 Virtual Graphs

This method is based on construction of sets of independent acyclic graphs, called virtual graphs. Each path in the network graph can be mapped onto a path in one of the virtual graphs. The source selects a particular virtual graph that contains path from source to destination. Routing is done via this path. In each node additional information is necessary for selecting an appropriate virtual graph [18]. The virtual graphs method is particularly well suited to the mesh topology. In this case only four virtual graphs are required.

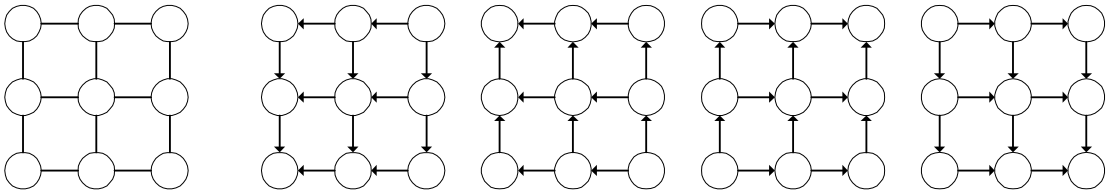

Figure 24: Mesh and its four virtual graphs

4.4.4 Dimension-ordered routing

One approach to avoid cycles in the resource dependency graph is to assign each channel a unique number and allocate channels to packets in strictly ascending (or descending) order [28][30]. In this class of routing algorithms the channel numbering scheme is based on the dimension of channels. Each packet is routed in one dimension at a time until it arrives at the proper position in this dimension. Then it is routed to the next dimension. By enforcing a strictly monotonic order on the dimensions traversed, deadlock-free routing is guaranteed.

Example 1: E-cube routing (for binary n-cube)

The E-cube routing algorithm guarantees deadlock-free routing in binary n-cubes. In a binary cube of dimension *d* a node is denoted with *d*-digit binary number. Each node has *d* output channels, one for each dimension. When a node in the cube receives a packet, the routing algorithm compares the packet's destination address with the address of the current node. If two addresses differ, the packet is forwarded on the channel in the dimension *k*, where *k* is the position of the rightmost (alternatively, leftmost) bit in which the addresses differ [30][28]. This algorithm is minimal and deterministic.

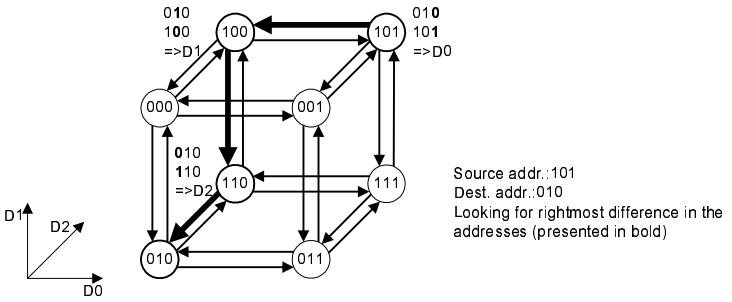

Figure 25: Example for E-cube routing in binary 3-cube

Example 2: XY routing (for meshes)

This is a dimensional routing algorithm for a 2D mesh. The packets are first send along the *X* dimension and then along the *Y* dimension. At most one turn is allowed and this turn must be from the *X* dimension to the *Y* dimension. In a 2D mesh each node is represented by its position (*x, y*). Let (s_x, s_y) and (d_x, d_y) denote the addresses of a source and destination node, respectively. Let (g_x, g_y) = (d_x – s_x, d_y - s_y). XY routing can be implemented by placing g_x and g_y in the header of the packet. During the travelling of the packet along the *X* direction, in every node g_x is decremented or incremented, depending on whether it is greater or less than 0. When g_x becomes 0 this means that the packet is at the right position in *X* dimension and it is forwarded to the *Y* dimension. Along *Y* dimension the packet travels until g_y becomes 0 [30]. This method is again deterministic and minimal. (gx, gy) is the relative address in the mesh of the destination node in respect to source node. The use of relative addresses makes the communication independent of the right position where an algorithm is mapped in the mesh.

Example 3: Routing in k-ary n-cubes

Using the technique of virtual channels, the E-cube routing algorithm can be extended to handle all *k*-ary *n*-cubes (cubes with dimension *n* and *k* nodes in each dimension). Each node in *k*-ary *n*-cube is identified by an n-digit radix k number. The i^{th} digit of the number represents the node's position in the i^{th} dimension. A 3-ary 2-cube is given as an example below. Here the node n_{02} is at position 2 in dimension 0 (horizontal) and at position 0 in dimension 1 (vertical). The channels are identified by the number of their source node and their dimension. For example, the channel from n_{11} to $n_{10,}$ which is in the dimension 0 (horizontal), is $c_{011.}$

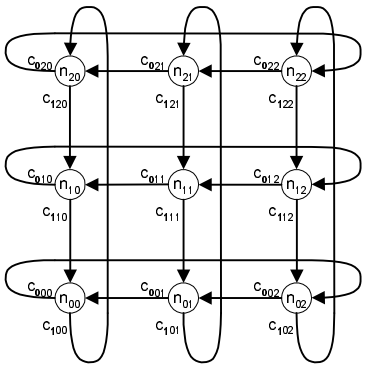

Figure 26: 3-ary 2-cube

To brake cycles, each channel is divided into an upper and lower virtual channel. The upper virtual channel of c_{011} is c_{0111} and lower virtual channel is $c_{0011.}$ To give internal channels (channels to/from the local processor) the lowest priority they are labelled with a dimension higher than dimension of the cube. Routing is restricted to route through channels in order of descending subscript. Priority is always given to the message from the channel with a lower subscript. Routing is made in order of dimension. In each dimension *i*, a message is routed in that dimension until it reaches a node whose subscript matches the destination address in the i^{th} position (the position where is the destination node in this dimension). The message is routed on the higher channel if the i^{th} digit of the destination address is greater than the i^{th} digit of the present node's address. Otherwise the message is routed on the low channel [28]. This routing algorithm is non-minimal and deterministic.

4.4.5 Minimal adaptive routing

This is a general adaptive routing technique, which requires the use of additional channels. It works by partitioning the channels into disjoined subsets. Each subset constitutes a corresponding sub-network. Packets are routed through different sub-networks, depending on the location of destination nodes. The additional channels can be realized, either as physical or virtual channels.

Example: Double Y-channel routing algorithm

This algorithm is an application of the Minimal Adaptive Routing for a 2D mesh. The mesh contains an additional pair of channels added to the *Y* dimension. The network can be partitioned into two sub-networks called the +*X* sub-network and –*X* sub-network, each having a pair of channels in the *Y* dimension and a unidirectional channel in the *X* dimension. An example is shown in the figure below.

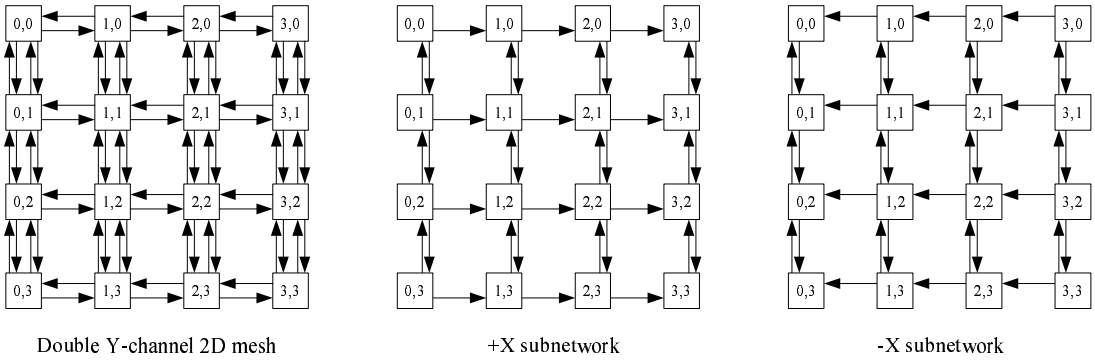

Figure 27: Double Y-channel routing

If the destination node is to the right of the source, the packet will be routed through the +*X* sub-network. If the destination node is to the left of the source, the -*X* sub-network is used. If source and destination node are at the same *X* position, either network can be used. This algorithm can be proven to be deadlock-free by ordering the channels appropriately. It is minimal and fully adaptive – a packet can be delivered through any of the shortest paths [30]. Providing deadlock-free minimal, fully adaptive routing algorithm for hypercube, torus, or more general k-ary n-cube topologies require additional channels. It is shown that k-ary n-cube can be partitioned into 2^{n-1} sub-networks, *n+1* levels per subnetwork, and k^n channels per level.

4.4.6 Non-minimal adaptive routing

Deadlock-free non-minimal adaptive routing can be provided using fewer addition channels. The following non-minimal adaptive routing algorithms, proposed by Dally and Aoki, can be applied to *k*-ary *n*-cube and mesh topology in which *r (1<r)* pairs of channels connect every pair of adjacent nodes [30].

Static dimension reversal routing algorithm

To apply this algorithm we need *r* pairs of channels between any two adjacent nodes. The network is partitioned into *r* sub-networks. The *class-i* sub-network consists of all the i^{th} pairs of channels ($0 \le i \le r-1$). The packet header carries an additional field *c*, called class field, which is initially set to 0. It shows the class sub-network in which the packet is now routed. For $c < r - 1$ the packet can be routed in any direction in *class-c* sub-network. However, each time a packet is routed from a high-dimension channel to a low-dimension channel, that is, reverse to the dimension ordering, the *c* field is increased by 1 (the packet enters in the next class sub-network). Once the value of *c* reaches *r-1* and a packet enters *class-(r-1)* sub-network, it must use deterministic dimensional-ordered routing. The parameter *r* dictates the degree of adaptivity of the routing algorithm. In fact, in every *class* sub-network a packet is routed following the dimension order, if the dimension order is violated, the packet enters the next class sub-network. So the parameter *r* (that is the number of sub-networks) determines how many times the dimension ordering can be violated [30].

Dynamic dimension reversal routing algorithm

In this routing algorithm the channels are divided into two nonempty classes: adaptive and deterministic. Packets originate in the adaptive channels. There they can be routed in any direction with no limit on the number of times the packet can be routed in reverse dimensional order. Here, again, the packet header carries the class field, c. The routing algorithm defines that a packet with $c = p$ is not allowed to wait on a channel currently occupied by a packet with $c = q$ if $p \ge q$. If a packet reaches a node where all permissible output channels are

occupied by packets whose *c* value is less or equal to its own, it must switch to the deterministic class of channels. When a packet enters the deterministic channels, it must follow the dimensional-ordered routing and cannot re-enter the adaptive channels. Here a circular waiting on channels it is impossible to occur and therefore the algorithm is deadlock-free. An important design issue concerns how many channels are classified as adaptive and how many are deterministic between each pair of adjacent nodes [30].

4.4.7 Turn model

This is a model for designing wormhole routing algorithms that are deadlock free, livelock free, minimal or non-minimal, and maximally adaptive for the networks. A feature of this model is that it is not based on adding physical or virtual channels to network topologies. Instead, the model is based on analysing the directions in which the packets can turn in a network and the cycles that the turns can form. It looks for a minimal number of turns that must be prohibited to break all the cycles.

Here we will present this model using an example with 2D mesh. For more details see [31]. In a mesh topology we have four directions, from which eight 90-degree turns can be formed. These eight turns form two abstract cycles.

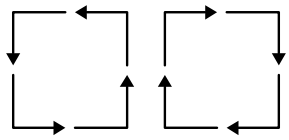

Figure 28: The possible turns and abstract cycles in 2D mesh

To avoid a deadlock situation these two cycles must be broken. In the *XY* routing algorithm the cycles are broken by prohibiting four turns (two turns in each cycle), which is half of the turns, and does not allow any adaptiveness.

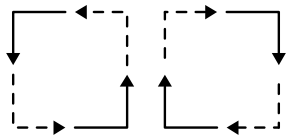

Figure 29: Allowed turns(solid lines) in the XY routing algorithm

On first instance we might conclude that only two turns need to be prohibited, one from each abstract cycle. This will allow some adaptiveness. But prohibiting any two turns will not prevent deadlock. Out of the 16 possible combinations of pairs of turns, four do not prevent deadlock, because they allow situations like those shown on the figure below.

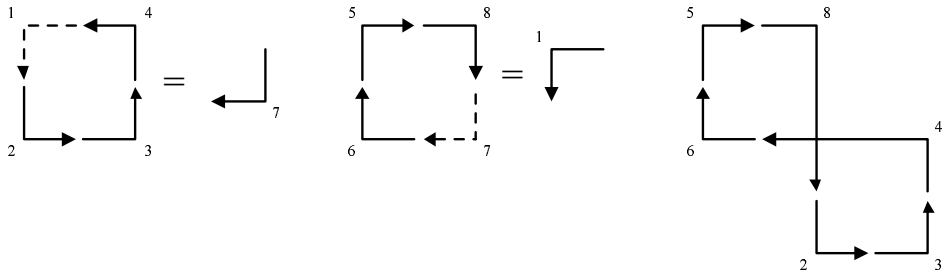

Figure 30: Six turns that complete the abstract cycle and allow deadlock

The remaining 12 combinations prevent deadlock and three are unique if symmetry is taken into account. These three unique combinations produce three routing algorithms based on the turn model – *West-First* routing algorithm, *North-Last* routing algorithm and *Negative-First* routing algorithm.

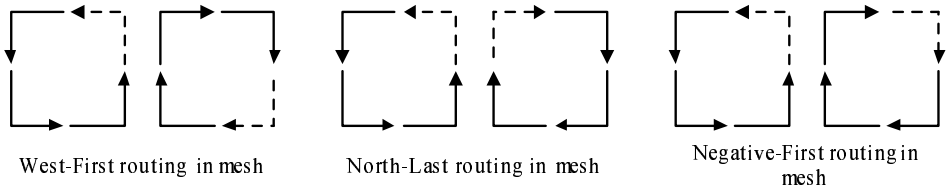

Figure 31: The three unique combinations, which produce three routing algorithms

West-First routing algorithm

In this routing algorithm the prohibited turns are the two to the west. Therefore, to travel west, the packet must start out in that direction: 'route a packet first west, if necessary, and then adaptively south, east and north'.

North-Last routing algorithm

In this case the prohibited turns are the two when travelling north. Therefore, a packet should only travel north when that is the last direction it needs to travel: 'route a packet first adaptively west, south and east, and then north'.

Negative-First routing algorithm

Here the prohibited turns are the two from a positive direction to a negative direction. Therefore, to travel in a negative direction, a packet must start out in a negative direction: 'route packets first adaptively west and south, and then adaptively north and east'. In [31] these three algorithms are proven to be deadlock free.

The turn model in n-dimensional mesh, k-ary n-cube and Hypercube

Similar algorithms can be applied to a n-dimensional mesh. Their names are respectively – *all-but-one-positive routing algorithm*, *all-but-one-positive-last routing algorithm* and *negative-first routing algorithm*. All these algorithms are deadlock free. The partly adaptive routing algorithms for the mesh can be extended for *k*-ary *n*-cube. The new algorithms are deadlock free but are strictly non-minimal. For *k*-ary *n*-cube with $k > 4$, it is impossible to construct deadlock free routing algorithms that are minimal without adding extra channels [31].

The partly adaptive routing algorithms for the hypercube are special cases of the algorithms for *n*-dimensional mesh and *k*-ary *n*-cube.

The table below summarise the presented classes of routing algorithms. The criteria are:

- Which topologies are suitable for implementation of this class of routing algorithms;
- What in the deadlock avoidance method used;
- Basic classification characteristics of the class;

		Class routing algorithms			
		Dimension-ordered routing	Minimal adaptive routing	Nonminimal adaptive routing	Turn model
Criteria	Suitable topologies	2D Mesh, Hypercubes, k-ary n-cubes	2D Mesh, Hypercube, k-ary n-cubes	k-ary n-cubes	2D-Mesh, Hypercube, k-ary n-cube
	Deadlock avoidance method	turns restrictions (plus additional channels for the k-ary n-cubes)	additional channels	additional channels	turns restrictions
	Characteristics	minimal, deterministic (except for the k-ary n-kubes - nonminimal)	Minimal, adaptive	nonminimal, adaprive	minimal or nonminimal, adaptive

To be energy-efficient the routing algorithm should always take the shortest path or in other words it has to be minimal. From the minimal deadlock free routing algorithms dimension-

ordered routing is the simplest one, but it has a disadvantage of unfair load distribution over the channels of different dimensions. The Turn model reduces this effect but does not eliminate this problem. The Minimal adaptive routing algorithms do not suffer from that problem, but need more resources. Here the extra channels complicate the resources arbitration while these resources stay under utilised. To select one of the algorithms a comparison should be made between the network quality improvements due to the use of Minimal adaptive routing and the arbitration complexity of the algorithm.

4.5 Router architectures

The interconnection networks' performance critically depends on the performance of the routers from which they are constructed. To study the router's performance we use router performance models. Such a router model has been proposed by Chien [51]. It is a parameterized performance model of a wormhole router with virtual channels and is based on the canonical architecture of a wormhole router shown on the picture below

Chien used the model to study the implementation complexity of some advanced routers' features such as adaptivity and virtual channels. His general conclusion is that the adaptive routing and virtual channels raise the implementation complexity, increase the router delay and this eliminates the performance advantages that they are suppose to have.

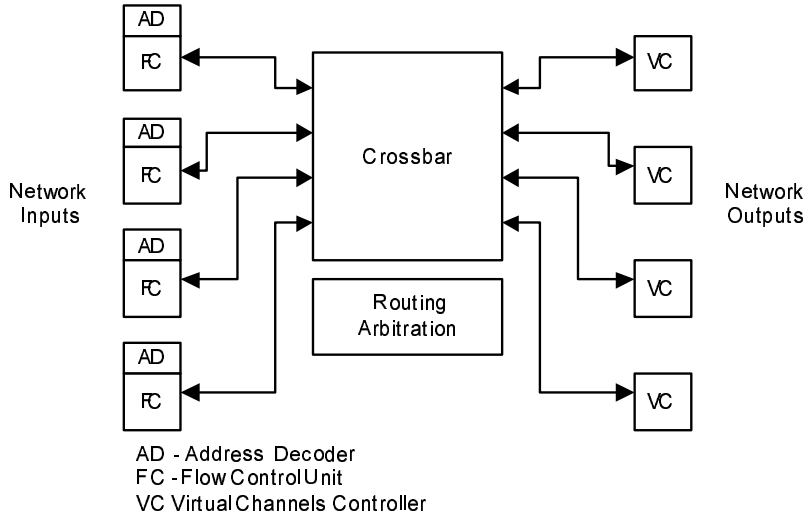

Figure 32: A canonical architecture for a wormhole router

In the Chien's model the implementation complexity is studied without considering the possibilities of pipelining. As most contemporary routers are heavily pipelined this model is inconsistent with the reality.

A more realistic router delay model was introduced by Peh [52]. It takes into account the pipelined nature of the routers and proposes pipelines matched to the specific flow control method employed. The results show that the latency of the virtual-channel router does not increase as the number of virtual channels is scaled up to 8 per physical channel, while throughput of up to 40% over a wormhole router can be gained.

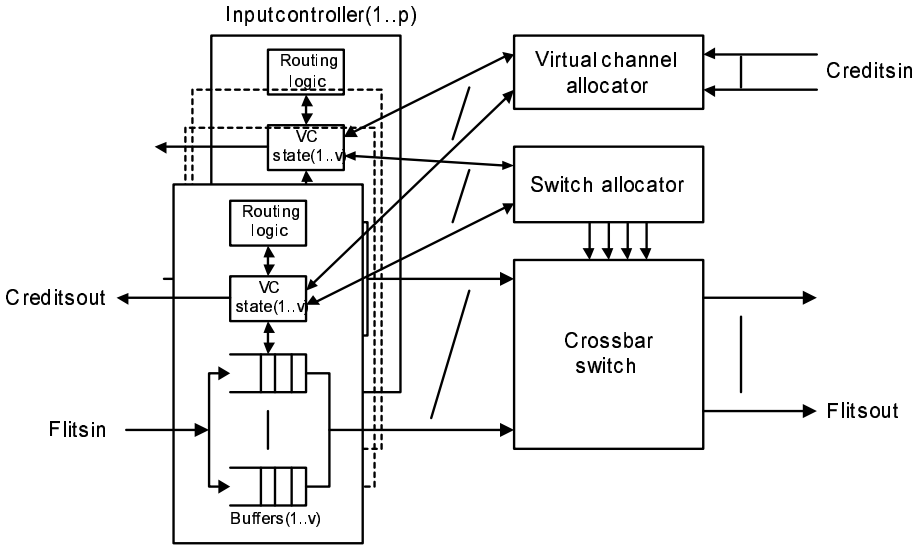

Figure 33: The canonical virtual-channel router architecture used by Peh, with p *physical channels and* v *virtual channels*

In the pipelined model several typical pipeline stages, called atomic modules, are proposed: *decoding and routing the packet header*, *allocation of virtual channel*, *allocation of a path through the crossbar*, *passing the crossbar*. The *VC allocation* stage does not exist in the wormhole router because there are no virtual channels. The *VC allocation* and *switch allocation* stage concern scheduling between several channels and it is not guaranteed that their requests for scheduling will be granted in the same cycle. Thus, after a packet has been routed it has to stay in the *VC allocation stage* until a grant for the VC arrives. Then it enters the *switch arbitration* where it requests a channel through the crossbar.

In [53] a speculative virtual-channel router is proposed in which the packet enters both stages *VC allocation* and *switch allocation* at the same time speculatively assuming that it will succeed in the first one. If it does not succeed the reserved crossbar allocation will be wasted.

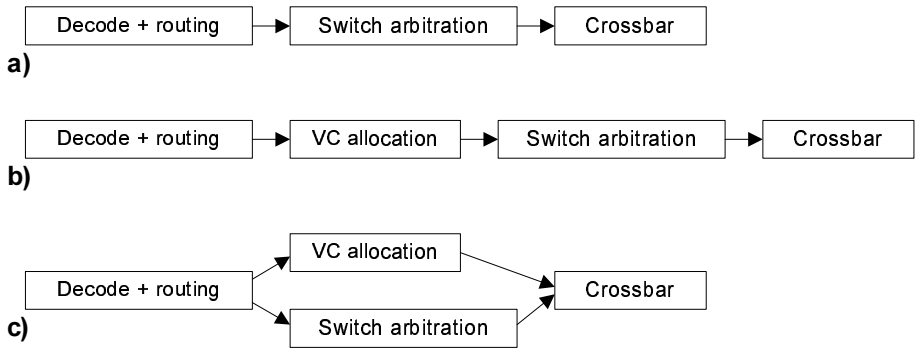

Figure 34: Atomic modules (pipeline stages) and dependencies of a: wormhole router (a), virtual-channel router (b), speculative virtual-channel router (c)

It is reported that this speculative architecture largely eliminates the latency penalty of using virtual-channel flow control, reducing the latency of a virtual-channel router to that of a wormhole router. The shorter pipeline also results in increased throughput for routers with small number of buffers as it reduce credit regeneration time.

4.6 Queuing

In a router it is possible for several input channels to receive packets going to the same output channel. In this situation an output contention occurs. As the physical output channel (usually) can accept only one packet at a time only one of the received packets will advance while the others will have to wait in the switch. These packets are stored in buffers and this

procedure is known as queuing. Different queuing strategies exist of which the most common one's are: *output queuing*, *input queuing*, and *central buffering*.

4.6.1 Output Queuing

In an output queued switch packets are queued at the switch's output queues. When a packet arrives at an input, it traverses the switch fabric to the required output and enters its queue. When packets arrive at the head of a queue they are sent on to the output channel. As it is possible for all the N inputs to receive packets simultaneously to the same output, the switching fabric must be able to switch up to N packets to one output in a single channel cycle (it should run at rate N times higher than the channel rate). The number of buffers required equals the number of output ports N. The flow control works on a packet level and does not allow fine grain communication.

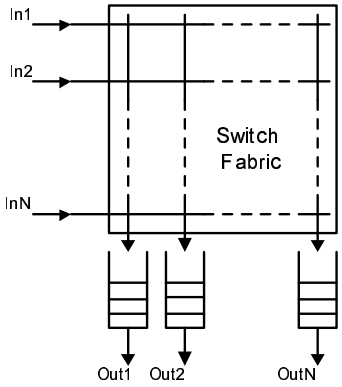

Figure 35: Output queued switch

Output queuing has better performance than input queuing. Assuming uniform load distribution the output queues saturate only when the input traffic rate approaches 1, so throughput of 100% can be achieved [38]. A disadvantage is that the switching fabric has a high complexity. An example of an output queued switch is the Knockout Switch [39].

4.6.2 Input Queuing

In an input queued switch packets are buffered at the switch's inputs. Each input has a queue. When a packet arrives at an input, it enters its queue. When a packet reaches the head of a queue it is scheduled for attaining the required output channel. The number of buffers required equals the number of switch's inputs N.

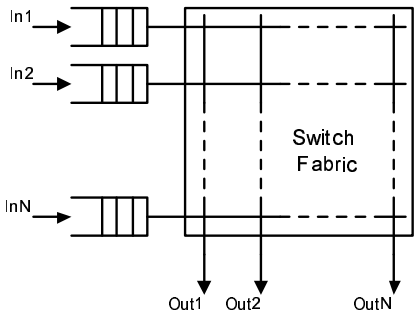

Figure 36: Input queued switch

The maximum achievable channel utilisation using input queuing with random selection policy and assuming uniform traffic and large number of inputs is approximately 58.6% [38]. The cause for this limitation is the Head-of-Line (HOL) blocking problem. When the first packet of a queue (head of the queue) is blocked all the other packets queuing behind it have to wait even if some of them are destined for currently idle outputs. This is a direct result of the FIFO queuing policy. The HOL problem can be reduced by using non-FIFO queues [40][41][42] or avoided by using multiple queues at each input.

Virtual Output Queuing. Instead of only one queue, each input maintains a separate queue for each output channel. In this way the HOL problem is avoided, but the number of buffers required is NxN (the number of inputs and the number of outputs is equal N). Using virtual output queuing and appropriate scheduling algorithms (Longest Queue First /LQF/ or Oldest Cell First /OCF/) 100% throughput can be achieved [43].

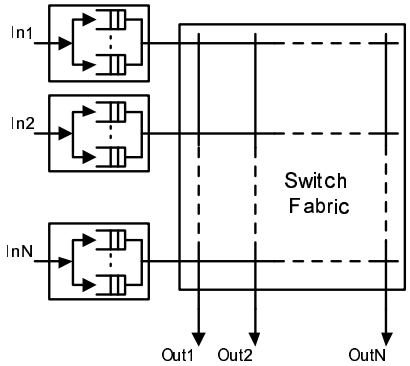

Figure 37: Virtual Output queuing

The virtual output queuing uses a static buffer allocation or there is dedicated buffer for each queue. The static allocation results in under-utilisation of the buffer space. For better a buffer space utilisation several dynamic buffer allocation policies have been proposed. When dynamic buffer allocation is used for each input there is a memory pool. A buffer space for an input queue is allocated in this pool only when and for the time it is needed. Some of the proposed management policies are: *linked list buffer management* [46], *self-compacting buffer management* [45], and *circular buffer management* [44].

4.6.3 Central Buffer

In switches using this type of buffers organisation the data is buffered in a large central shared memory. In this memory a buffer space is allocated for each data link between input and output port. Each switch's input can be connected to the input of the memory and the output of the memory can be connected to every switch's output. The buffers space allocation, which determines the switch efficiency, can be performed according different buffer management schemes: *complete sharing (CS)*, *complete partitioning (CP)*, *sharing with minimal allocation (SMA)*, *sharing with maximum queue (SMXQ)*, *hot spot push out (HSPO)*, etc [56].

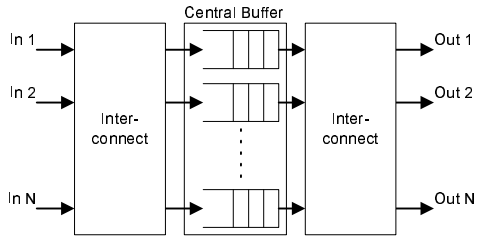

Figure 38: Central Buffered Switch

The throughput of this type of switches is maximized and like the output queued switches they can achieve 100% throughput [54]. The memory required is significantly less than for the others queuing strategies and the difference grows as the number of input and output channel increases. In typical cases, the amount of memory needed in a shared buffer system is just two to tree times larger than the amount needed for a single output port in the case of dedicated buffering [55].

From implementation point of view a problem of the central buffer switch is the high bandwidth required for the shared buffer memory. This memory has to accept all data from all the inputs and to write all data to al the outputs. In other words, the shared memory for switch with *N* inputs, *N* outputs and channel rate *R* must have bandwidth of *2NR*.

An example of real central buffered switch is the IBM's SP2 [57]

4.7 Scheduling

Scheduling is the process of taking the decision about when data is sent from particular inputs to their desired outputs. This decision is taken by a system block called scheduler. Some desirable properties of a scheduler are: *to be fast*, *to be fair* and *to be easy to implement*.

Generally there are two types of scheduling: *static scheduling* and *dynamic scheduling*.

Static scheduling. Static scheduling is strongly related to time division multiplexing and is performed at compile time. Bandwidth allocation is made on the basis of timeslots within in frames. The scheduler is based on a table. In each node there is a scheduling table that describes for each input of the router for each timeslot in the frame which is the destination output. Advantages of the method are its bounded latency and guaranteed bandwidth, which simplify the implementation of real-time properties. Disadvantage is its inflexibility and its inability to react dynamically to changes of the load.

Dynamic scheduling. This type of scheduling is performed at run-time. In contrast of the previous one, it does not rely on information preliminary stored in a table, but the scheduler has to take decision every cycle. Normally, when data arrives at a given input this input sends a request to the scheduler. From all requests received in each cycle the scheduler selects a conflict-free subset and tries to satisfy them. The method ensures high flexibility but does not give bandwidth and latency guaranties.

There are number of scheduling algorithms: *Maximum Size Matching*, *Maximum Weight Matching*, *Parallel Iterative Matching (PIM)*, *Iterative Round Robin Matching with Slip (SLIP)*, and *Least Recently Used (LRU)*. Here we only enumerate them. More information and some performance analysis can be found in [50], [43].

5 Conclusion

The next generations of semiconductor technology is going to lead to a chip design complexity that is unmanageable with the current design methods and tools. A possible solution of this problem is a design methodology based on a tiled heterogeneous reconfigurable System on a Chip (HRSoC). In such an approach two basic system components can be recognized – reconfigurable computation components and communication components. While the computation components perform the required data processing, the communication components have to provide a unified, reliable and efficient communication environment for data exchange. Studies have shown that the communication between the computation components is as important for the system performance and the system efficiency as the computation components. Defining a communication network is not a trivial task because there are many contradicting requirements – performance, flexibility, scalability, cost, energy, etc.

The problem of inter-node communication is not new, it is known from the multi-processor systems domain and a lot of work has been done in that area. While the main goal for the multiprocessors' inter-chip interconnection networks is high performance, for the on-chip interconnection networks other issues are important: energy-efficiency, flexibility and reusability. The chip socket's pin limitation is not a factor for nodes in on-chip network - the technology allows using a larger amount of wires compared to inter-chip networks.

In the Chameleon project we are addressing the design of a heterogeneous, reconfigurable System-on-a-Chip. Because of the targeted application domain (future low-power hand-held systems) the emphasis is on pairing high-performance with low power consumption. This reflects on the interconnection network requirements. The low power consumption requirement adds a new aspect to the design of an interconnection network – energy-efficiency.

The topology

In nearly all projects concerning SoC a 2D mesh is used as a topology for the interconnection network. The 2D mesh meets most design criteria of an energy-efficient on-chip interconnection network topology and seems to be a good choice for a SoC network. Being a low dimensional topology, it is not difficult to implement the 2D mesh in a plane. Its layout is efficient, without wire congestions, the wires are short and structured, which is an important

factor for reaching energy-efficiency. The mesh's structure is regular and it is easy extendable. Because of its connections between nearest neighbours it allows exploiting the communication locality effectively, which is another precondition for energy-efficiency. The mesh topology has been used in conventional interconnection networks, it is well understood and many deadlock-free routing algorithms can be found for it. Another popular well studied low dimensional topology is folded torus. It outperforms the mesh with a lower diameter, higher connectivity and symmetry. But when the number of nodes in the network increase (above 16) lack of "nearest neighbour" connections is seen in the centre of the node array.

Routing

For a routing algorithm to be energy-efficient the first property it must obey is that it must be minimal. A minimal routing algorithm routes a packet through the shortest path between the source and destination node, which means that the packet will pass through a minimal number of channels. The energy spent for communication is mainly determined by the energy dissipated in the wires. Therefore minimizing the number of hops reduces the communication energy cost. Considering the communication latency, the routing decision must not be complex and must be taken for as quick as possible. The time to take a routing decision is one of the factors that determine the packet per-hop delay and therefore the total packet latency.

There are two types of routing algorithms that ensure deadlock freedom – the first type is based on using additional (virtual) channels and the second type is based on turns restrictions. The algorithms based on turns restriction have several advantages: easy implementation, the routing decisions are simple, and they do not require additional channels. Examples of such kind of algorithms are the Dimensional-Ordered routing algorithms and the algorithms derived from the Turn Model. Unfortunately, these routing algorithms destroy the nodes' channels load balance, because the turn restrictions lead to restrictions in the channel usage.

The deadlock-free routing algorithms based on usage of additional channels give more freedom when taking routing decisions, but they require additional resources. Considering these algorithms, the Double Y-channel routing is the one that requires the least additional channels in a 2D mesh topology – it only needs a doubling of the channels in the Y dimension. These channels can be implemented as virtual channels, which mean that the added resources are virtual channels' control logic and buffering space. However, deciding to save control logic and use a routing algorithm based on turns restrictions, might introduce channels congestions, inefficient channel usage and as a consequence bad network performance.

Flow control

The contributions that a flow control can make to the energy-efficiency are to reduce the control signals activities and to avoid data retransmissions. The later one is ensured by the credit-based flow control (similar to the TCP/IP windowing mechanism). In general the most popular flow control method is wormhole, which in combination with virtual channels ensures flexibility, high network throughput, good channel utilization and efficient usage of the node's buffer space [25].

Queuing

Input queued switches are the most popular at the moment. In combination with some technique for avoiding the Head-Off-Line problem it becomes a solution with quite reasonable cost-performance ratio. We think that input queuing combined with virtual channels will be a good solution that offers high performance and high flexibility, while keeps the implementation complexity manageable.

Acknowledgements

This research is supported by PROGram for Research on Embedded Systems & Software (PROGRESS) of the Netherlands Organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the technology foundation STW.

References

[1] http://bwrc.eecs.berkeley.edu/Research/Configurable_Architectures/

[2] Zhang H., Wan M., George V., Rabaey J, “Interconnect Architecture Exploration for Low-Energy Reconfigurable Single-Chip DSPs.”, *IEEE Computer Society Workshop on VLSI*, pp. 2-8 1999.

[3] http://gram.eng.uci.edu/morphosys/index.html

[4] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J. Kurdahi, Nader Bagherzadeh, Eliseu M. C. Filho, “MorphoSys: An Integrated Reconfigurable System for Data-Parallel Computation-Intensive Applications.”, *IEEE Transactions on Computers*, volume 49, Issue 5, pp. 465-481, May 2000.

[5] Hartej Singh, Guangming Lu, Ming-Hau Lee, Fadi Kurdahi, Nader Bagherzadeh, Eliseu Filho, Rafael Maestre, “MorphoSys: Case Study of a Reconfigurable Computing System Targeting Multimedia Applications.”, *37th Design Automation Conference*, Session 34 [p.573], Los Angeles, CA, June 2000.

[6] http://www.cag.lcs.mit.edu/numesh/

[7] David Shoemaker, Frank Honoré, Chris Metcalf, Steve Ward, “NuMesh: An Architecture Optimized for Scheduled Communication.”, *Journal of Supercomputing*, 10(3):285-302 (1996).

[8] http://www.ecs.umass.edu/ece/tessier/rcg/jian.html

[9] J. Liang, S. Swaminathan, and R. Tessier, “aSOC: A Scalable, Single-Chip Communications Architecture.”, *In the Proceedings of the IEEE International Conference on Parallel Architectures and Compilation Techniques*, Philadelphia, PA. October 2000.

[10] http://www.cag.lcs.mit.edu/raw/

[11] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal, “Baring it all to Software: Raw Machines.”, *IEEE Computer*, September 1997, pp. 86-93.

[12] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal, “Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures”, *MIT/LCS Technical Report LCS-TR-859*, July 2002

[13] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Jae-Wook Lee, Paul Johnson, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe and Anant Agarwal, ‘The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs”, *IEEE Micro*, Mar/Apr 2002.

[14] Smit G.J.M., Havinga P.J.M., Smit L.T., Heysters P.M., Rosien M, “Dynamic Reconfiguration in Mobile Systems”, *12th International Conference on Field Programmable Logic and Application*, Montpellier, September 2002

[15] Luca Benini, Giovanni De Micheli, “Networks on Chips: A New SoC Paradigm.”, *Computer*, January 2002 (Vol. 35, No. 1), pp. 70-78

[16] W. J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks”, *DAC*, June 2001, pages 684-689.

[17] Paul M. Heysters, Jaap Smit, Gerard J.M. Smit, Paul J.M. Havinga, “Exploring Energy-Efficient Reconfigurable Architectures for DSP Algorithms.”, *PROGRESS2000*, October 2000

[18] G. J. M. Smit, “The Design of Central Switch Communication Systems for Multimedia Applications.”, Ph.D. Thesis, University of Twente, 1995.

[19] “Distributed Systems”, ACM Press New York, 1993

[20] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, A. Sagiovanni-Vincentelli, “Addressing the system-on-a-chip interconnect woes through communication-based design.”, *In Proceedings of the 38th Design Automation Conference*, June 2001

[21] Pierre Guerrier, Alain Greiner, “A generic architecture for on-chip packet-switched interconnections”, *Design, Automation and Test in Europe*, Proceedings pages 250 - 256, Paris, France, 2000.

[22] Cheng-Ta Hsieh, Massoud Pedram, “Architectural power optimization by bus splitting.”, *Proceedings of the conference on Design, automation and test in Europe*, p.612-616, March 27-30, 2000, Paris, France

[23] W. J. Dally, “Performance analysis of k-ary n-cubes interconnection networks.”, *IEEE Transaction on Computers* 39(6) (1990) 775-785.

[24] Li-Shiuan Peh and William J. Dally, “Flit-Reservation Flow Control’, *In Proceedings of the 6th International Symposium on High-Performance Computer Architecture (HPCA)*, Toulouse, France, January10-12, 2000, pp. 73-84.

[25] W. J. Dally, “Virtual channel flow control”, *IEEE Transactions on Parallel and Distributed systems*, vol. 3, no. 2, pp. 194-205, March, 1992.

[26] José Duato, Pedro López, Federico Silla, Sudhakar Yalamanchili, “A High Performance Router Architecture for Interconnection Networks”, *In Proceedings of the International Conference on Parallel Processing*, Bloomington, Illinois, pp. 61-68, August 1996.

[27] Paul Wielage, Kees Goossens, “Networks on Silicon: Blessing or Nightmare?”, *Euromicro Symposium On Digital System Design (DSD 2002)*, Dortmund, Germany, September 2002.

[28] W. Dally and C. Seitz, L., “Deadlock-Free Message Routing in Multiprocessor Interconnection Networks”, *IEEE Trans. on Comput.* vol. 36, pp. 547-553, 1987

[29] Gerard J.M. Smit, Paul J.M. Havinga, Lodewijk T. Smit, Paul M. Heysters, Michel A.J. Rosien, “Dynamic Reconfiguration in Mobile Systems”, *12th International Conference on Field Programmable Logic and Applications (FPL2002)*, September 2-4, 2002, Montpellier, France.

[30] L. M. Ni, P. K. McKinley, “A Survey of Wormhole Routing Techniques in Direct Networks”, *Computer*, vol. 26, no. 2, pp. 62--76, Feb. 1993.

[31] Christopher J. Glass, Lionel M. Ni, “The turn model for adaptive routing”, *ACM SIGARCH Computer Architecture News*, v.20 n.2, p.278-287, May 1992

[32] E. A. de Kock, W. J. M. Smits, Pieter van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, Paul Lieverse, Kees A. Vissers, G. Essink, “YAPI: application modeling for signal processing systems.”, *DAC 2000*: pp. 402-405

[33] J. Rabaey, "Digital Integrated Circuits: A Design Perspective", Prentice Hall, 1996.

[34] P. Kermani, L. Kleinrock, “Virtual cut-through: A new computer communication switching technique”, *Computer Networks*, 3:267--286, 1979.

[35] Whobrey D.: "A communications chip for multiprocessors", *Proc. CONPAR* 88 pp 464-473, 1988.

[36] Paul M. Heysters, Gerard J.M. Smit, “Mapping of DSP Algorithms on the MONTIUM Architecture”, *International Parallel and Distributed Processing Symposium (IPDPS 2003)*, April 22-26, 2003, Nice, France.

[37] G. Kahn, "The semantics of a simple language for parallel programming.", In *Info. Proc.*, pages 471-475, Stockholm, Aug. 1974.

[38] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Transactions on Communications*, vol. 35, no. 12, pp. 1347--1356, December 1987.

[39] Y.-S. Yeh, M. G. Hluchyj, and A. S. Acampora, "The knockout switch: A simple, modular architecture for high-performance packet switching," *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, pp. 1274--1282, Oct. 1987.

[40] M. Karol , M. Hluchyj: "Queueing in High-Performance Packet Switching", *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, December 1988, pp. 1587-1597.

[41] M. Chen and N. D. Georganas, "A fast algorithm for multi-channel/port traffic scheduling," in *Proc. IEEE Supercom/ICC`94*, pp. 96--100.

[42] Huang, A. and Knauer, S., “Starlite: a wideband digital switch”, *Proc. GLOBECOM'84* (November 1984), pp 121-125.

[43] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260-1267, August 1999.

[44] N. Ni, M. Pirvu, and L. Bhuyan, “Circular buffered switch design with wormhole routing and virtual channels”, In Proceedings of the *1998 IEEE International Conference on Computer Design*, pages 466--473, October 1998.

[45] J. Park, B. O'Krafka, S. Vassiliadis, and J. Kelgado-Frias, “Design and evaluation of a DAMQ multiprocessor network with self-compacting buffers”, In *IEEE Supercomputing '94, The Conference on High Performance Computing and Communications*, pages 713--722, Nov.14-18 1994.

[46] Y. Tamir and G. L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches", *IEEE Transactions on Computers*, Vol. 41, No. 6, June 1992.

[47] Miltos D. Grammatikakis, Derbiau Frank Hsu, Miro Kraetzl, “Parallel System Interconnections and Communications”, CRC Press, 2001

[48] Becker J.: “Configurable Systems-on-Chip: Challenges and Perspectives for Industry and Universities”, Proceedings Engineering of Reconfigurable Systems and Algorithms, pages 109-115, Las Vegas, USA, June 2002.

[49] Bolsens I., “Challenges and Opportunities for FPGA platforms”, pages 391-392, Proceedings of Field-Programmable Logic and Applications, Montpellier, Fance, September 2002.

[50] McKeown N., Anderson T.E., "A quantitative comparison of scheduling algorithms for input-queued switches", *Computer Networks & ISDN Systems*, vol. 30, n. 24, Dec. 1998, pp. 2309-26.

[51] A. A. Chien, “A Cost and Speed Model for k-ary n-cube Wormhole Routers”, In *Proceedings of the Hot Interconnects Workshop*, August 1993

[52] Li-Shiuan Peh and William J. Dally, "A Delay Model for Router Micro-architectures", In *IEEE Micro*, Jan/Feb 2001.

[53] Li-Shiuan Peh and William J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers", In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, Jan. 22-24, 2001, Monterrey, Mexico, pp. 255-266.

[54] Sundar Iyer, Rui Zhang, and Nick McKeown, "Routers with a Single Stage of Buffering", *ACM SIGCOMM Aug. 2002, Pittsburgh, USA. Also in Computer Communication Review,* vol. 32, no. 4, Oct 2002.

[55] J. Turner and N. Yamanaka, "Architectural choices in large scale ATM switches," *IEICE Trans. Communications*., vol.E81-B, no.2, pp.120-137, Feb. 1998.

[56] S. Fong, S. Singh, M. Atiquzzaman, "An Improved Buffer Sharing Scheme for ATM Switches under Bursty Traffic", *Proceedings of the 19th Australian Computer Science Conference*, pp.26-34, Feb. 1996.

[57] C. Stunkel, D. Shea, B. Abali, M. Atkins, C. Bender, D. Grice, P. Hochshild, D. Joseph, B. Nathanson, R. Swetz, R. Stucke, M. Tsao, and P. Varker, “The SP2 High-Performance Switch”, *IBM Systems Journal*, 34(2):185--204, 1995.