Thesis Report

Project Title: ***Effective Dynamic Re-mapping Algorithm for low power Network-on-chip NoC.***

Prepared by: ***Aminu Shehu Mahdi, African University of science and technology, Abuja, Nigeria.***

Project Supervisor: ***Ben Abdallah Abdelrezak, The University of Aizu, Graduate School of Computer Science and Engineering, Adaptive systems laboratory. Aizu, Japan.***

September 2010.

ABSTRACT

With the increase in the possibility of incorporating multiple cores on a single chip (MCSoC), the issue of an efficient interconnection that is scalable, takes up small area and has low power consumption must be taken into consideration carefully. Network on chip (NoC) has evolved as a promising solution for efficiently interconnecting multiple core on a single chip (MCSoC). NoC brings conventional networking theories and methods to chip communication and brings notable improvements over the conventional bus systems.

The aim of my research will be to study power consumption in NoC architecture and propose *an effective dynamic remapping algorithm* to reduce power consumption on NoC. This is done by monitoring the NoC at run time and dynamically re-mapping the cores to reduce power consumption. Low power consumption is desirable in MCSoC because high power increases capacitance, electro magnetic interference (EMI) and dissipates more heat, thereby reducing performance. In addition, must devices built using MCSoC are hand-held devices (battery powered) and therefore might not have access to continuous power supply.

I will be using the OASIS NoC which was developed at the *Adaptive systems laboratory, The University of Aizu, Graduate School of Computer Science and Engineering. Aizu, Japan* To test my Algorithm. OASIS NoC is a complexity effective on-chip interconnection network.

ACKNOWLEDGEMENT

I owe my deepest gratitude to my supervisor, Dr. Ben Abdallah Abderazek, whose encouragement, supervision and support throughout the course of my thesis work enabled me to develop an understanding of the subject. This thesis work wouldn't have been possible without him. I offer my regards and blessings to all those who supported me in any respect during the completion of my thesis, especially the teaching and administrative staff and my school mates in AUST.

I am also grateful to Nafisa Abdullahi and Mr. Babatunde Olajide for making my stay in AUST a memorable one and for all their advice and support towards my Academics. My gratitude also goes to my friends, whose support and encouragement helped me overcome difficulties and setbacks during my graduate studies.

Most importantly, none of this would have been possible without the love and patience of my family. My Mum to whom this dissertation is dedicated to, has been a constant source of love, concern, support and strength all these years.

Dedication

To my Mum,
whose love and care has been the shining light that brightens up my life.

Table of Contents

Chapter I

Chapter II

Chapter III

Chapter IV

Appendix.

References.

List of Figures.

List of Tables.

Chapter I

1.0 Introduction

For the next decade, Moore's Law is still going to bring higher transistor densities allowing billions of transistors to be integrated on a single chip. However, it became obvious that exploiting significant amounts of instruction-level parallelism with deeper pipelines and more aggressive wide-issue super-scalar techniques, and using most of the transistor budget for large on-chip caches has come to a dead end. Scaling performance with higher clock frequencies, especially, is getting more and more difficult because of heat dissipation problems and energy consumption that is too high. The latter is not only a technical problem for mobile systems, but is also becoming a severe problem for computing centres because high energy consumption leads to significant cost factors in the budget. Improving performance can only be achieved by exploiting parallelism on all system levels [1].

Multicore architectures offer a better performance/Watt ratio than single-core architectures with similar performance. Combining multi-core and co-processor technology promise extreme computing power for high CPU-time-consuming applications. [1]

1.1 System on chip (SoC):

System-on-a-chip or system on chip (SoC or SOC) refers to integrating all components of a computer or other electronic system into a single integrated circuit (chip). It may contain digital, analogue, mixed-signal, and often radio-frequency functions – all on a single chip substrate. A typical application is in the area of embedded systems.

A typical SoC consists of:

- One micro-controller, microprocessor or DSP core(s). Some SoCs – called multiprocessor System-on-Chip (MPSoC) – include more than one processor core.
- Memory blocks including a selection of ROM, RAM, EEPROM and flash.
- Timing sources including oscillators and phase-locked loops.
- Peripherals including counter-timers, real-time timers and power-on reset generators.
- External interfaces including industry standards such as USB, FireWire, Ethernet, USART, SPI.
- Analogue interfaces including ADCs and DACs.
- Voltage regulators and power management circuits.

These blocks are connected by either a proprietary or industry-standard bus such as the AMBA bus from ARM. DMA controllers route data directly between external interfaces and memory, by-passing the processor core and thereby increasing the data throughput of the SoC [2].

1.2 Multiprocessor System on chip (MPSoC):

In computing, a processor is the unit that reads and executes program instructions, which are fixed-length (typically 32 or 64 bit) or variable-length chunks of data. The data in the instruction tells the processor what to do. The instructions are very basic things like reading data from memory or sending data to the user display, but they are processed so rapidly that we experience the results as the smooth operation of a program.

Processors were originally developed with only one core. The core is the part of the processor that actually performs the reading and executing of the instruction. Single-core processors can only process one instruction at a time. (To improve efficiency, processors commonly utilize pipelines internally, which allow several instructions to be processed together, however instructions are still consumed into the pipeline one at a time.)

A ***multi-core processor*** is composed of two or more independent cores. One can describe it as an integrated circuit which has two or more individual processors (called cores in this sense). Manufacturers typically integrate the cores onto a single integrated circuit die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package

Multi-core processors are widely used across many application domains including general-purpose computing, embedded systems, networks , digital signal processing (DSP), and graphics [2].

1.3 Communication on MPSoC:

Traditionally, a bus based system is used for communication on MPSoC. The bus serves as an on-chip interconnect for all communicating cores on a single chip. The bus-based architectures have evolved over the last couple of years from single shared buses to multiple bridged buses (hierarchical buses), and crossbars.

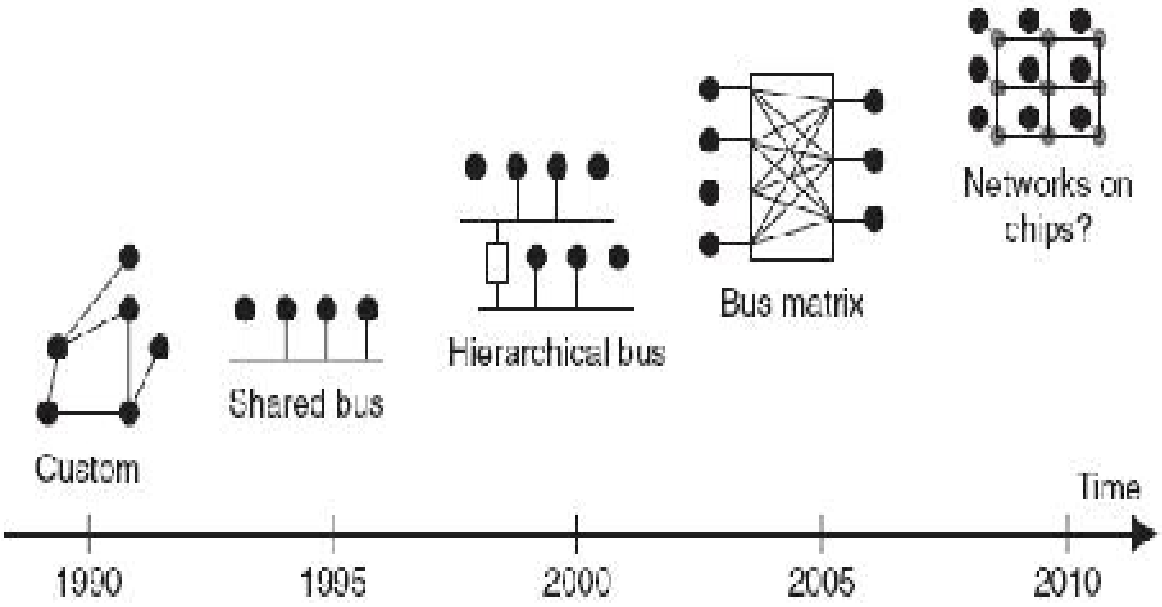

Figure 1: Evolution of on-chip communication.

Common network topologies to interconnect cores include bus, ring, 2-dimensional mesh, and crossbar topologies. Cores may or may not share caches, and they may implement message passing or shared memory inter-core communication methods.

The hierarchical buses, allow multiple buses to operate in parallel, however, all the communicating partners need to be connected to a single crossbar. These bus-based architectures suffer from scalability problems for a large number of connected processing elements[4].Figure below shows conventional bus-based communication approach.

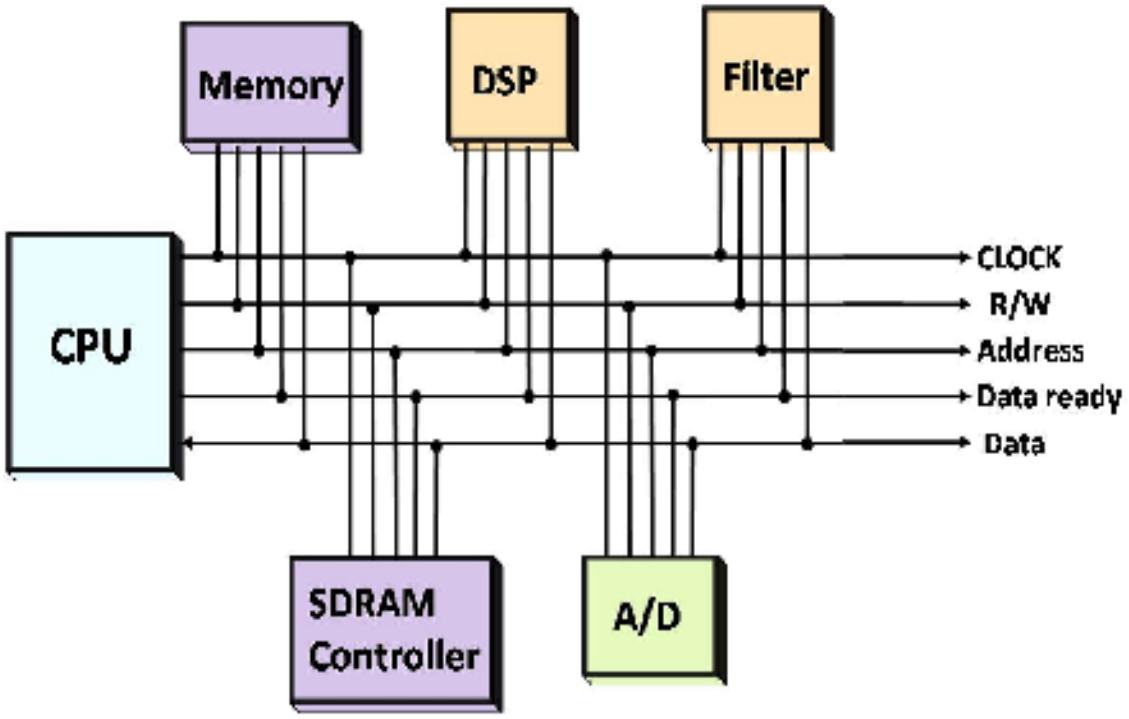

Figure 2. Conventional Bus System.

Buses have been deployed for communication since the beginning of circuit design and made their way in SoC due to their well understood concepts, their compatibility with most of the available node processors, the area taken on the chip and the zero latency after the arbiter has granted control [13].

1.4 Drawbacks of the conventional bus system.

1. Poor scalability: On the bus architecture, resources are shared between communicating cores, this limits the number of cores that can be integrated on a single bus.

2. Communication delay: On the bus architecture, only one core can send messages at a time, this constraint leads to communication delay because a core has to wait for a while before sending messages out. This leads to a lack of parallelism and communication concurrency.

3. Ad-hoc Global wire Engineering: There exists many different ways in which cores could be connected in a bus architecture, this leads to ad-hoc and non standardized global wire engineering on the chip.

4. Bandwidth is limited and shared and therefore the link speed goes down with increase in number of connected cores.

5. The bus system usually have a single bus arbiter that regulates access and activities on the bus. This central arbitration creates a single point of failure on the bus architecture.

6. Computation and communication are coupled on the Bus architecture.

7. On the bus architecture, no abstraction is done in terms of the layers of communication. Therefore any change affects the whole architecture.

8. Clock skew is an inherent synchronization problem of long links, as using the same clock for all components throughout the whole system will not be achievable with the growing future complexity [4]

As technology scales, MPSoC applications exhibit huge communication demands, scalable communication architectures are needed for efficient implementation of future complex Multicore SoCs. Due to the lack of scalability, both in terms of power and performance, traditional bus-based communication architectures fail to satisfy the tight requirements of future applications [1].

A new communication paradigm is therefore needed to solve the aforementioned problems of the bus architecture. This communication architecture should aim to provide high throughput, scalability, low power, reduced packet loss, utilize link efficiently, reduce contention and occupy less area on silicon [13].

1.5 Network-on-chip (NoC):

Network on chip (NoC) has evolved as a promising solution for efficiently interconnecting multiple cores on a single chip (MCSoC). NoC brings conventional networking theories and methods to on-chip communication and brings notable improvements over the conventional bus systems. Figure below shows a NoC architecture.

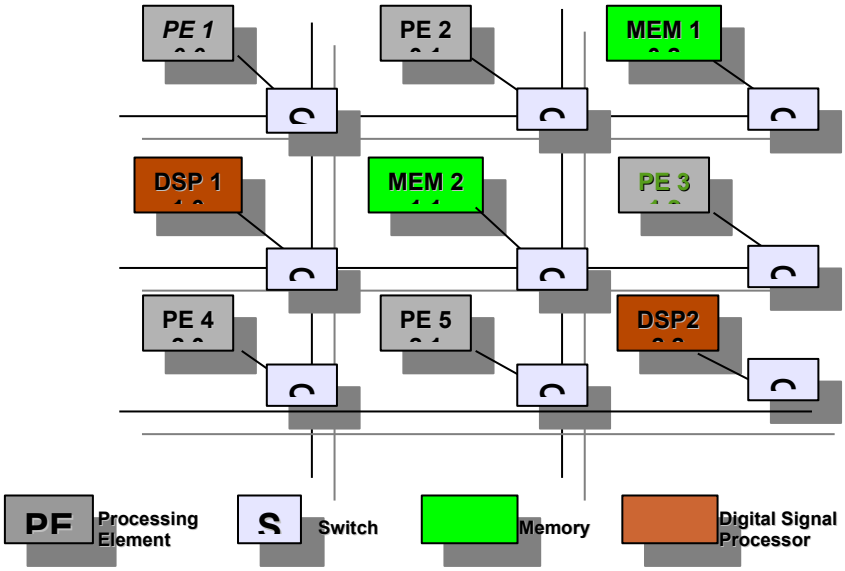

Figure 3: NoC Architecture.

The basic idea behind NoC is that processors are connected via a packet switched communication network on a single chip similar to the way computers are connected to internet. A chip based on NoC interconnection consists of several network clients (e.g. processors, memories, and custom logic) which are connected to a network that routes packets between them [2].

The NoC Architecture solves the problems associated with bus architecture. It is scalable, parallelism and concurrency are easily supported because multiple cores can send and receive packets at the same time (concurrent communication). NoC has distributed arbitration and the aggregate bandwidth grows. The link speed in NoC is not affected by the number of connected cores. NoC has separate layers of abstraction, Communication and Computation are separated, the former is achieved by the NoC while the cores are responsible for the later. On the NoC architecture, messages are transmitted via packets.

Packet switching supports asynchronous transfer of information. It provides extremely high bandwidth by distributing the propagation delay across multiple switches, thus pipelining the signal transmission. In addition, the NoC offers several promising features. First, it transmits packets instead of words. Dedicated address line like in bus systems are not necessary since the destination address of a packet is part of the packet.

Secondly, transmission can be conducted in parallel if the network provides more than one transmission channel between a sender and a receiver. Thus, unlike bus-based system on chip, NoC presents theoretical infinite scalability, facile IP core reusing, and higher parallelism [2].

In [15] the author summarises the various parts of a NoC Architecture. He described the various parts of a Heterogeneous NoC. In each tile of a heterogeneous on-chip network, the **PE** (Processing element) can flexibly be any processing unit like CPU/DSP/ARM/FPGA/ASIC core, embedded memory, application-specific component or periphery, according to the target applications. A **router** (or switch) in the tile is typically designed with a crossbar switch and arbiter unit, and five input/output channels (each of them is made up of a controller and an input buffer) which connects to the local PE and the extended routers in the four directions.

A processing element communicates with others by exchanging data with its local router (switch), and passing data packages throughout the network links to approach the target node. A **crossbar switch** in the router is used to route data packets from its input **buffers** to the appropriate output links, and a **crossbar arbiter** is used to determine the transmission priorities when traffic conflict happens inside the router. Thus packets traverse multiple links and go through routers in the NoC fabric from the source to the destination [16].

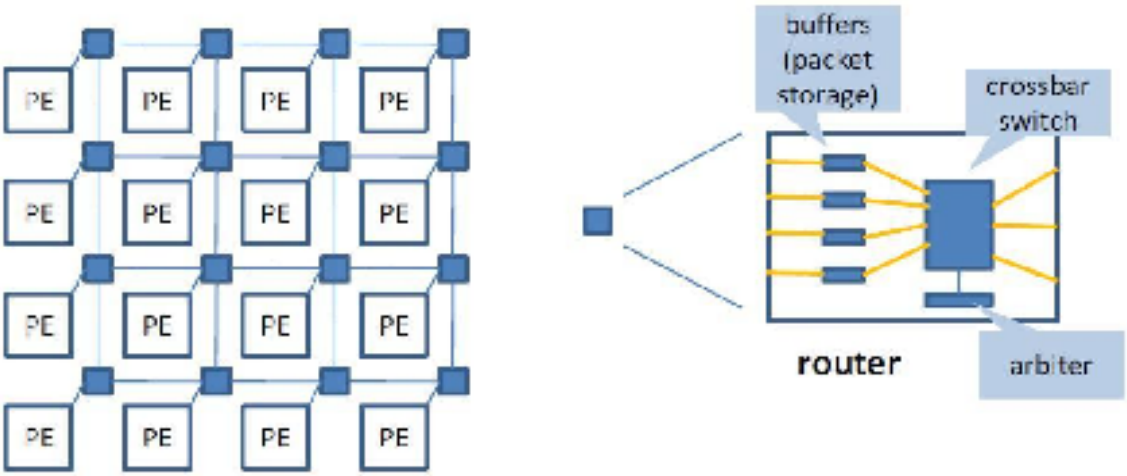

Figure 4: Mesh Network with a router(switch).

1.6 Properties of NoC:

The Diagram below depicts the various properties of NoC architecture:

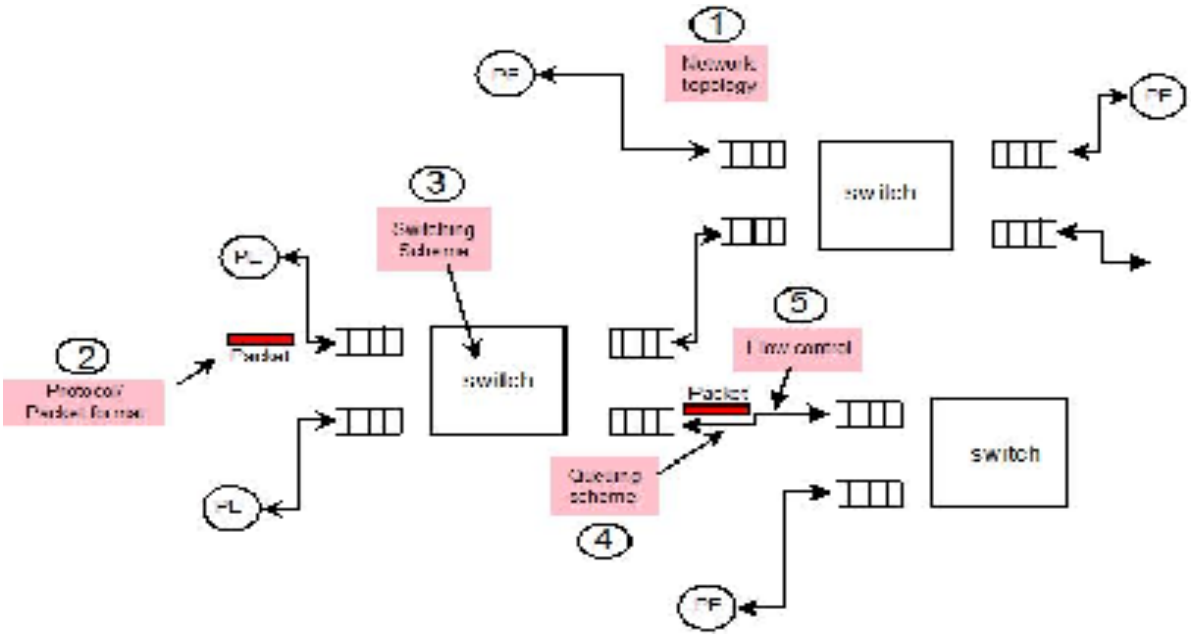

Figure 5: Properties of a NoC Architecture.

1.6.1 Network Topology:

The topology of a typical NoC system simply defines how the nodes are interconnected by links. Topologies can vary depending on system's modules sizes and placements (functional requirements).

There are several well known standard topologies on which an application can be mapped. We broadly classify them as direct and indirect topologies. In direct topology, each core is connected to a single switch some examples of direct topology are mesh, torus, ring, butterfly, octagon. For indirect topology, a set of cores are connected to a single switch. Fat tree and folded torus are indirect topologies proposed [3].

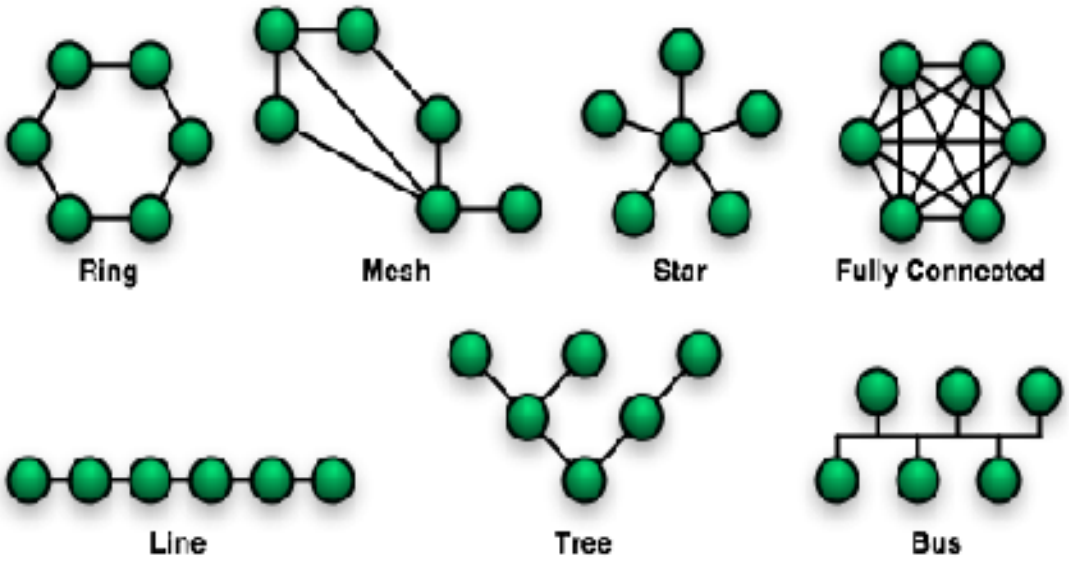

Figure 6: Different Network topologies.

1.6.2 Switching Scheme:

Simply put, switching is how a message (packet) traverses the route. There is a large protocol space to select from for NoCs. Circuit switching, packet switching, and wormhole switching are possible choices for NoC protocols. These schemes can be mainly distinguished by their flow control methodologies. When these switching techniques are implemented in on-chip networks, they will have different performance along with different requirements on hardware resources [3].

In circuit switching, a physical path from the source to the destination is reserved prior to the transmission of data. Once a transmission starts, the transmission is not corrupted by other transmission since packets are not stored in buffer as in packet switching (discussed later). The advantage of circuit switching approach is that the network bandwidth is statically reserved for the whole duration of the data.

Moreover, because circuit switching does not need packet buffers, area and power consumption can be reduced. The overhead of this approach is that the setting up of an end-to end path causes unnecessary delay. In summary, circuit switching can provide high performance but little flexibility [3].

Another alternative to circuit switching is packet switching scheme. Packet based communication has been brought to NoCs from the Internet world but loses the original advantage of reliability in the absence of dynamic routing. Currently, most of the proposals for routing in NoCs are based upon static routing mechanisms — XY-coordinate discipline [3].

In packet based communication, data is divided into fixed length packets and whenever the source has a packet to be sent, it transmits the data. Every packet is composed of a control part, the header, and a data part (also named payload). Network switches inspect the headers of incoming packets to switch the packet to the appropriate output port. In this scheme, the need for sorting entire packets in a switch makes the buffer requirement very high [3].

The last technique is the so called wormhole packet-switching, where each packet is further divided into flits (flow control unit) and the input and output buffers are expected to store only a few flits. Therefore, the buffer space requirement in the switches can be small and compact compared to the packet switching scheme. The header flit reserves the routing channel of each switch, the biddy flits will then follow the reserved channel, and the tail flit will later release the channel reservation [3].

The advantage of the wormhole routing is that it does not require the complete packet to be stored in the switch's buffer while waiting for the header flit to route to the next stages. Thus, it requires much less buffer spaces. One packet may occupy several intermediate switches at the same time. Therefore, because of these advantages, wormhole routing is an ideal candidate switching technique for on-chip multiprocessor interconnects networks [3].

The disadvantage is that by allowing a message to occupy the buffers and channels, wormhole routing increases the possibility of deadlock. In addition, channel utilization is somehow decreased if a flit from a given packet is blocked in a buffer [3].

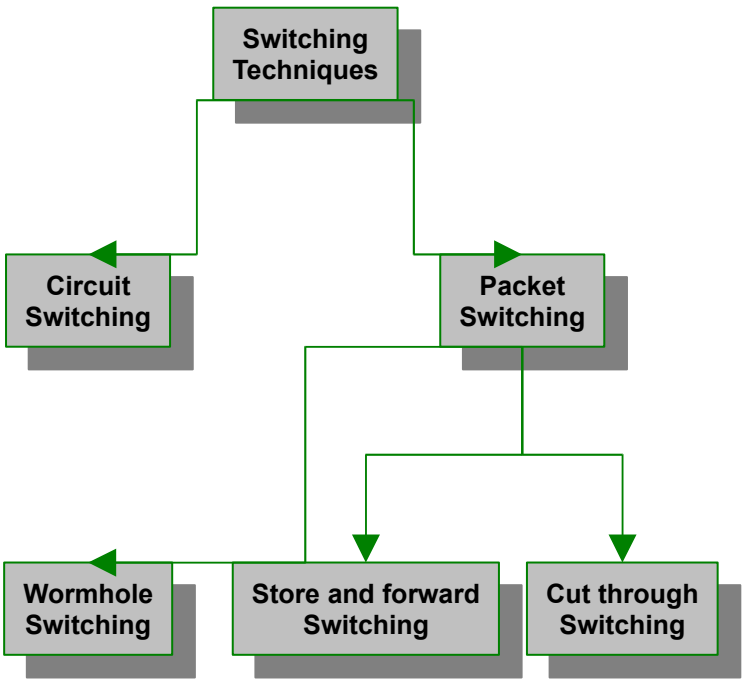

Figure 7: Switching Techniques

1.6.3 Flow control:

Flow control schedules the traversal of messages (packets) over time. This is used only for dynamic routing. Flow control determines how resources, such as buffers and channels bandwidth are allocated and how packet collisions are resolved. Whenever the packet is buffered, blocked in place, dropped, or misrouted depends on the flow control strategy. A good flow control strategy should avoid channel congestion while reducing the latency. Examples of flow control mechanisms include T-Error, stall and go and ACK/NACK. [3]

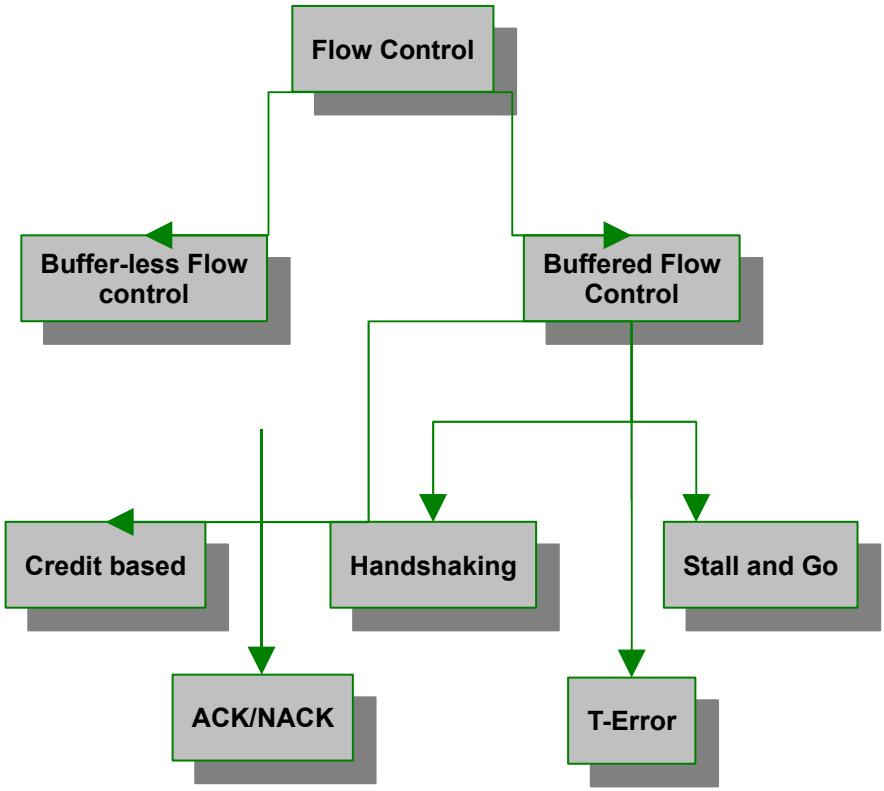

Figure 8: Flow control Mechanisms.

1.6.4 Packet Format:

Packet Size Selection: The packet size highly depends on the characteristics of the application being used (i.e. data or control dominated). If a message has to be split in too many small packets which have to be re-assembled at destination to obtain the original message, the resulted overhead will be too high. On the other case, if the packet is too large, the packet might block the link for too many cycles and potentially block other traffic with side effects on the performance of the whole system. Therefore, the correct packet size is also crucial to make optimum use of the network resources. [3]

1.6.5 Queuing scheme (Virtual Channels):

In [18], the authors describe the design of virtual channels in NoC. A virtual channel splits a single channel into two channels, virtually providing two paths for the packets to be routed. There can be two to eight virtual channels. The use of VCs reduces the network latency at the expense of area, power consumption, and production cost of the NOC implementation. However, there are various other added advantages offered by VCs [18].

Network deadlock/livelock: Since VCs provide more than one output path per channel there is a lesser probability that the network will suffer from a deadlock; the network livelock probability is eliminated (these deadlock and livelock are different from the architectural deadlock and livelock, which are due to violations in inter-process communications) [18].

Performance improvement: A packet/flit waiting to be transmitted from an input/output port of a router/switch will have to wait if that port of the router/switch is busy. However, VCs can provide another virtual path for the packets to be transmitted through that route, thereby improving the performance of the network[18].

Supporting guaranteed traffic: A VC may be reserved for the higher priority traffic, thereby guaranteeing the low latency for high priority data flits [18].

Reduced wire cost: In today's technology the wire costs are almost the same as that of the gates. It is likely that in the future the cost of wires will dominate. Thus, it is important to use the wires effectively, to reduce the cost of a system. A virtual channel provides an alternative path for data traffic, thus it uses the wires more effectively for data transmission. Therefore, we can reduce the wire width on a system (number of parallel wires for data transmission) [18].

1.6.6 Routing scheme:

Routing is the process of selecting paths in computer networking along which to send data or physical traffic. Routing algorithms are responsible for correctly and efficiently routing packets or circuits from the source to destination [16].

In [15], the author describes two types of routing techniques: static (deterministic) routing and dynamic (adaptive) routing. Deterministic routing means routing paths are completely determined offline, while adaptive routing is that paths are online determined depending on dynamic network conditions. Deterministic routing has design simplicity and low latency under loose network traffic, but performs throughput degradation when network congestion happens. Adaptive routing uses alternative paths when network is congested, which provides higher throughput, while it will experience higher latency if network congestion is low.

In NoCs, the routing scheme usually selects candidates among the routing paths that have minimum distance between the source and destination nodes. There are many routing algorithms available e.g. XY routing and odd-even routing [17]. They are both theoretically guaranteed to be free of deadlock and livelock.

The XY routing strategy can be applied to regular two-dimensional mesh topologies without obstacles. The position of the mesh nodes and their nested network components is described by coordinates, the x-coordinate for the horizontal and the y-coordinate for the vertical position. A packet is routed to the correct horizontal position first and then in vertical direction. XY routing produces minimal paths without redundancy, assuming that the network description of a mesh node does not define redundancy.

The odd-even turn model is a shortest path routing algorithm that restricts the locations where some types of turns can take place such that the algorithm remains deadlock-free. More precisely, the odd-even routing prohibits the east → north and east → south turns at any tiles located in an even column. It also prohibits the north → west and south → west turns at any tiles located in an odd column.

1.7 NoC Design Flow:

A typical heterogeneous NoC design flow is described in Figure 9. First of all, types of needed PEs are selected according to the natures of target applications **(PE selection)**. Tasks are then mapped to the PEs **(task allocation)**, and topology synthesis is applied. The mapping procedure decides the position of the selected PEs in the topology **(tile mapping)**. Then, path selection for communications between the application tasks is performed **(routing path allocation)**, and the scheduling policy or result of each PE is decided **(task scheduling)**. Some optimizations are performed throughout the entire flow, like channel width selection and buffer sizing.

The figure below shows the NoC design flow. Each step is guided by protocols and algorithms. A survey of various NoCs was made in [18]. Various techniques and Algorithms that have been employed in NoC synthesis are discussed there.

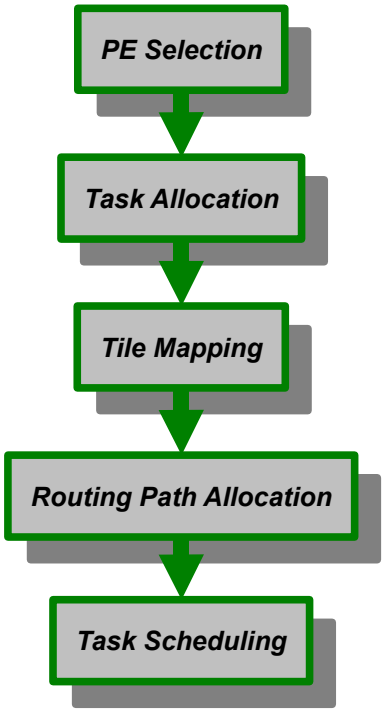

Figure 9: NoC Design Flow.

1.8 OASIS NoC:

The OASIS NoC was developed at the *Adaptive systems laboratory, The University of Aizu, Graduate School of Computer Science and Engineering. Aizu, Japan.*

OASIS NoC is a complexity effective on-chip interconnection network. it uses a 4x4 mesh network and adopts wormhole switching. The size of one flit is 76 bit, and it has information of destination address, next port direction information and payload. Only input ports have buffers in OASIS. The maximum number of ports is five is the OASIS NoC (NORTH, SOUTH,EAST, WEST & SELF) Each input port has FIFO which is 76 bits

Fig. 10: OASIS flit structure. DATA(64bit) is payload, TAIL(1bit) means last flits of a packet.

NEXTPORT(5bit) is direction of output, XDEST(3bit) is destination x-address, YDEST(3bit) is destination y-address. The next port direction in next address is decided in input port. By comparing the destination address with the next address. The algorithm is as follows:

◦ IF Y destination address > Y next address

then next port == NORTH

ELSE next port == SOUTH

◦ IF X destination address > X next address

then next port == EAST

ELSE next port == WEST

◦ IF destination address == next address

then next address == SELF

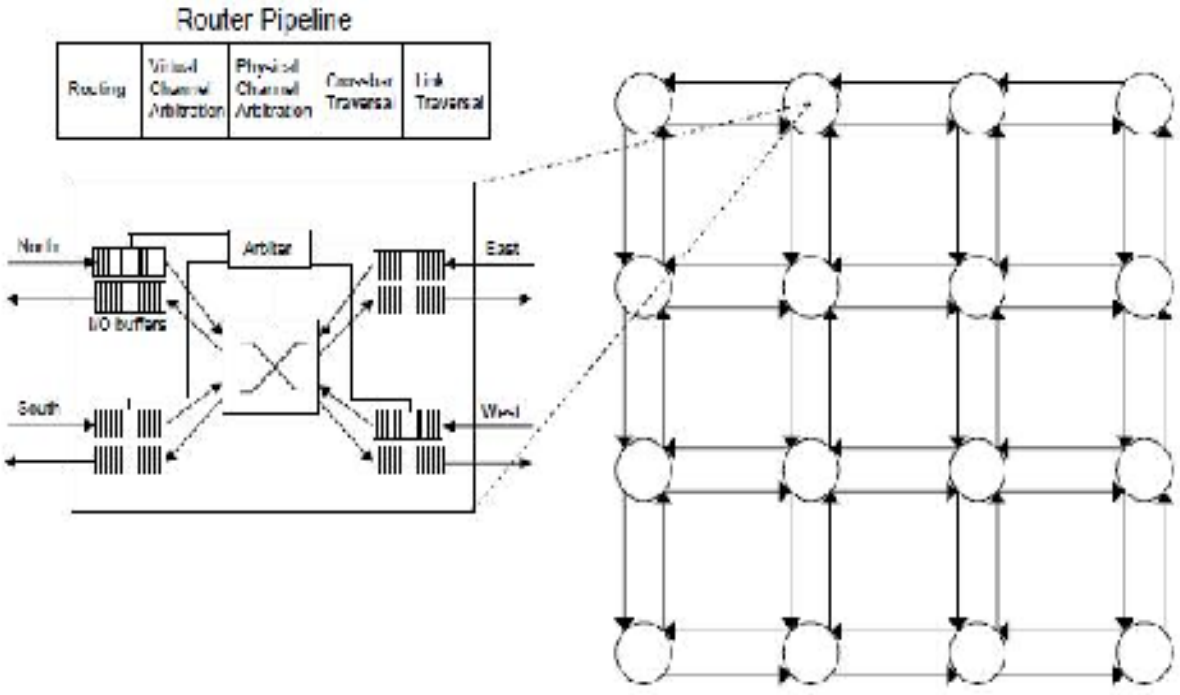

Fig 11: OASIS Switch.

OASIS switch has an Allocator and an Arbiter. The switch allocator decides the direction of flit in input port and when the flit should be sent to output. The Arbiter is used to resolve deadlock. It determines high priority flit in input port, OASIS arbiter schedules in round-robin scheme. The flow control mechanism in OASIS is called the STALL-GO.

.

Chapter II

2.1 Objective of the Research:

The aim of my research will be to study power consumption in NoC and propose *an efficient dynamic re-mapping algorithm* to reduce power consumption on NoC architecture. This will be achieved by monitoring the NoC at run time and dynamically re-mapping the cores (PE) to bring frequently communicating cores as close as possible.

Although today's processors are much faster and far more versatile than their predecessors using high-speed operation and parallelism, they also consume a lot of power. Moreover, an interconnection network dissipates a significant fraction of the total system power budget. For instance, the MIT Raw on-chip network consumes 36% of the total chip power and Alpha 21364 microprocessor dissipates 20% of total power in interconnection network [7].

As observed in [5], As the number of routers/switches between communicating cores decreases, the power consumption and communication delay between the cores also decreases. Dynamic re-mapping will be aimed at reducing the link length (by implication the number of switches/routers crossed by packets from source to destination) between frequently communicating cores.

2.2 Energy model of NoC

As the characteristic of NoC, power dissipation is a critical issue throughout the design. We only consider communication power dissipation which is consumed on the router/switch and the interconnect wires. Computation power dissipation on the PEs is not an issue in NoC. Most researchers use power modeling with bit energy, which is the power consumption of transferring a bit of data from the source node to the destination, as the energy model of NoC [15].

Ye et al. [28] proposed a model for power consumption of network routers/switches. The bit energy *(Ebit)* metric is defined as the dynamic energy consumed when one bit of data is transported through the router/switch:

$$Ebit = ESbit + EBbit + EWbit \quad (2.1)$$

Where *ESbit , EBbit* and *EWbit* represent the dynamic energy consumed by the switch, buffering and interconnection wires inside the switching fabric, respectively.

Since in Eq. (2.1), *EWbit* is the dynamic energy consumed on the wires inside the switch fabric, for NoC, the dynamic energy consumed on the links between tiles (*ELbit*) should also be included. Thus, the average dynamic energy consumed in sending one bit of data from a tile to its neighbouring tile can be calculated as:

$$Ebit = ESbit + EBbit + EWbit + ELbit \quad (2.2)$$

Let *ERbit* to be the average energy consumption of transferring one bit of data through a router, that is:

$$ERbit = ESbit + EBbit + EWbit \quad (2.3)$$

Consequently, in a new model, the average energy consumption of sending one bit of data from tile *ti* to tile *tj* is:

$$E_{bit}^{t_i, t_j} = nhops \times ERbit + (nhops - 1) \times Elbit \qquad (2.4)$$

Where *nhops* is the number of routers/switch the bit passes. *ERbit and Elbit* in Eq. (2.4) are constants for a given design. Their values depend on the router architecture, inter-tile link geometry (width, length, shielding, etc.) and can usually be derived through profiling/simulation or analysis.

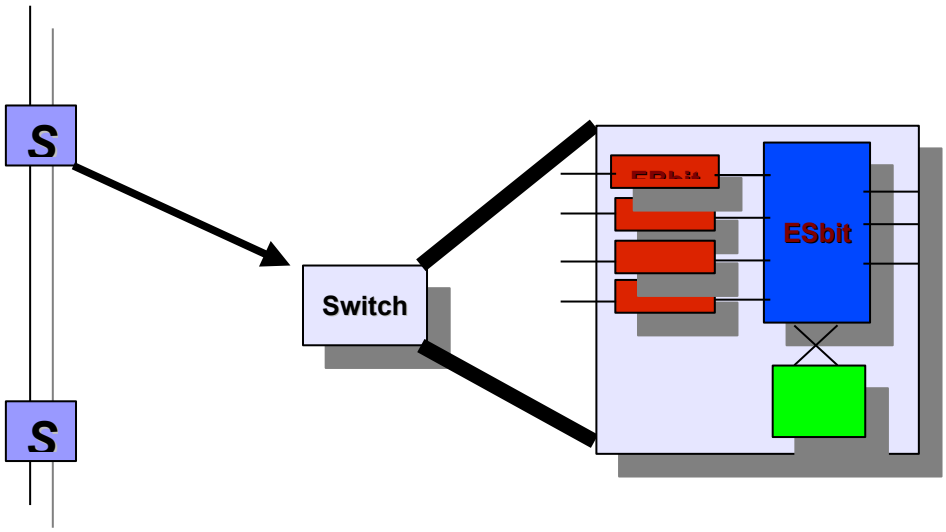

Figure 12: The NoC *Ebit* Energy Model.

Thus, using Eq. (2.4), the communication energy consumption can now be analytically calculated, provided that the communication volume between any communicating tile pair is known. For regular 2D mesh networks with minimal routing, Eq. (2.4) shows that the average energy consumption of sending one bit of data from tile *ti* to tile *tj* is determined by the Manhattan distance between them [27].

The power is mainly consumed by three sources: the internal node switches, the internal buffer queues and the interconnect wires [19, 20]. The degree of switch power consumption is decided by the choices of process technologies, voltage levels and operating frequencies. The internal buffer power consumption mainly comes from the occurrences of destination contentions and interconnect contentions in the buffer memories. The interconnect wires power consumption is directly related to the link lengths.

From the theories and experiments in [19, 20], the following conclusions were derived which can guide our designs with energy saving in mind. Storing a packet in buffer consumes far more energy than transmitting the packet on interconnect wires. This “buffer penalty” indicates that energy consumed in buffers is a significant part of total energy consumption of switch fabrics, and the buffer energy will increase very fast as the packet flow throughput increases [15].

How to minimize the power dissipation of a NoC system is basically the first and foremost criteria that embedded system designs focus on, and it is discussed in almost every paper in this area. Regarding to this, The contribution of this research is propose an *efficient dynamic re-mapping algorithm* that will significantly reduce interconnect wire length, number of switches/routers crossed (hence reduce “buffer penalty”).

2.3 Causes and Types of power consumption on NoCs:

To emphasize on the discussions above, types and causes of power consumption in a NoC based SoC architecture are summarized in this section. NoC source of power consumption can be classified into two broad classes:

1. Node centric and
2. Network centric.

The node centric power consumption is the power consumed by the IP cores. The processing elements (processor and memory) and custom logic cores consume power in SoCs, but they mostly have power management tools and policies implemented in them (cores). One of such policies described in [6]; involves dividing the states in which a node (IP core) can be (Active, sleep and idle).

The *active* state is the state in which the IP core is making some computation while the *idle* state is the state in which no computation is being made. When a core is in the *idle* state for some time it transits to a *sleep* state where the IP core is turned off and no power is consumed.

The power consumed during communication on the NoC, is refered to the network centric power consumption. In [8]; it has been observed that NoCs' buffers and links can consume near 75% of the total NoC power, thus there is a significant benefit to optimizing buffer size, link length and bandwidth of a NoC design. Parts of the NoC that consumes power include:

2.3.1 The Routers or Switch Fabric:

These are the interconnection devices through which packets of data are routed to their destination. The router/switch fabric consume power during routing/switching activity. The switching protocol and queuing scheme employed by the router/switch is also a determining factor of how much power is consumed on a NoC.

2.3.2 Buffers on Routers:

These are the input and output buffers on the switches/routers. The size of the buffer is proportional to the amount of energy consumed in keeping packets of data before they are routed out.

2.3.3 Link length:

The length of link between communicating cores. This is length of wire connecting a core to a switch/router and the length of wire connecting switch/router to another switch/router. This is the length which packets have to travel and is also proportional to amount of power consumed on the links. The NoC topology determines the link length.

2.3.4 Size of flits:

Large flits take up buffers space and take a long while to send thereby consuming power. Buffer size is directly proportional to the size of the flit in NoC.

2.3.5 Routing protocol:

The routing protocol may contribute to the power consumption in NoC. If the routing protocol uses longer routes to destination thereby increasing the hop count (number of switches/routers crossed to packet destination).

2.4 Desirability of Low Power:

Low power consumption is desirable in MCSoC because:

1. High power increases capacitance,electro magnetic interference (EMI) and dissipates more heat, thereby reducing performance and usability.
2. Most devices built using MCSoC are hand-held devices and therefore might not have access to continuous power supply (Battery powered Embedded systems).
3. High power is generally expensive, leads to early wear and tear of processors and it is unfriendly to the environment.

NoCs can be broadly classified into two categories, the Application specific NoC and the Adaptive NoCs. The Application specific NoCs are design parametrised and are generally tailor made for a certain application or an application domain and fail in scenarios of hard-to-predict system behaviour and/or in situations where reliability is a concern [4].

On the other hand, adaptive NoCs are general purpose NoCs that can be used for a broad class of applications. The adaptive NoCs therefore have the ability to self adapt to changes in an application requirement, user behaviour, system constraints and/or reliability issues by, for instance, dynamically (re-)mapping a running application at runtime or changing other NoC properties like the buffer size and routing protocol at runtime.

2.5 Advantages of the Adaptive NoC over the Application-specific NoC:

The table below highlights the advantages of an adaptive NoC over Application Specific NoC:

	(Design-time) Application-specific NoC		**(Runtime) Adaptive NoC**	
Flexibility	All the design related parameters, i.e. number of virtual channel, buffer depth are fixed at design time. No flexibility.	(-)	All the design-related parameters may be configured at runtime to meet the requirements of the different tasks. Therefore,high flexibility.	(+)
Routing & Mapping Algorithm	Routing and mapping are static and performed offline, therefore they are only efficient in specific scenarios.	(-)	Dynamic routing and runtime mapping are used. Therefore, may support varying workloads and/or constraints in systems.	(+)
Hardware Resources	Hardware resources are allocated to each part of NoC at design time. Therefore, no resource multiplexing (lower resource utilization).	(-)	Hardware resources may be shared and reused. Therefore, higher resource utilization and lower silicon area.	(+)
Efficiency	Design-time decisions only cover certain efficiency scenarios and fail in efficiency when hard -to-predict system scenarios occur.	(-)	The system may be adapted both at system-level as well as architecture-level during unpredicted scenarios.	(+)
Reliability	Static design and therefore, may not be able to recover from faults.	(-)	The system-level adaptation allows to move a complete task to another PE and therefore, may recover from faults. Architecture-level adaptation supports resource reuse to other parts. Higher resource utilization during faults.	(+)
Complexity of Control logic	Simple implementation and the control management is easy to realise	(+)	Increased complexity. The control requires additional hardware and consumes more power. Over-designed, hard-to-predict the initial hardware budget.	(-)

Table 1: Advantages/disadvantages of the adaptive and the application-specific NoCs (symbol “(-)” stands for showing the disadvantages and “(+)” for showing the advantages) [4]

2.6 Power Management Policies and Methods

Various power management policies and methods have been proposed to reduce power consumption on NoC architecture to the possible minimum. Some of the methods are:

2.6.1 Reconfigurable buffers/routers:

In [9]; the authors propose a method which allows an increase in the buffer size through a borrowing/lending process of buffers among neighbouring routers. They show how a NoC built with reconfigurable routers allows the use of buffers with smaller depths having similar performance with the one obtained in a NoC using a fixed router with a large buffer depth. This in turn saves power, and improves energy usage in the NoC.

2.6.2 Multi-Mode Switch:

In [10]; the authors propose a multi-mode switch, the basic idea is that if they can predetermine the mode of switches from the normal mode to the lease-line mode or the off mode, it can save the power consumption by passing signals through lease lines or turning off the idle switches.

2.6.3 Dynamic Voltage scaling Closed-loop control concept:

The power management optimization problem is formulated and solved using a closed-loop control model with a combination of node and network centric power management approaches. Each communicating core has its local power manager that consists of an estimator and a controller. The estimator observes changes in the state of the local core, incoming traffic to the core (node-centric) and the special requests for power management coming from the other cores on the network (network-centric). Based on the changes detected, it recalculates the optimal control. The optimal controller selects the appropriate power and performance states of the local core [6].

2.6.4 Voltage island Shut-down:

In many Systems on Chips (SoCs), the cores are clustered into islands. When cores in an island are unused, the entire island can be shutdown to reduce the leakage power consumption [10].

2.6.5 Reconfigurable Architecture:

Reconfigurable architecture that selects the best topology, switching policy and packet size at run time to best suit the running application. In essence you have a variable packet length, packet and circuit switching co-existing. Adaptive routing is also employed.

2.6.6 Reconfigurable Topology:

This are methods of power management achieved through re-mapping of cores. Remapping bring communicating cores closer and thereby shorten communication link length and reducing the number of buffers/routers crossed by packets between two communicating cores. The main contribution of this paper will be to propose *an effective dynamic remapping algorithm* to reduce power consumption on NoC. This is done by monitoring the NoC at run time and dynamically re-mapping the cores to reduce power consumption.

2.7 Why Dynamic Re-mapping:

Tasks are mapped onto the various cores on a MPSoC, different tasks/processes exhibit different communication requirements. Dynamically re-mapping the cores to bring frequently communicating core at runtime reduces the link length traversed by packets, the number of switches/routers are also reduced. In [14]; the authors found out that less switching favours low power consumption as switching of packets consumes energy.

Remapping cores to reduce distance between frequently communicating cores also reduces routing delays and the channel occupancy. Communication becomes faster and therefore energy consumed due to inter-IP communication is significantly reduced.

2.8 NoC Monitoring:

Run-time observability (Monitoring) is a prerequisite for runtime adaptivity (re-mapping) as it is providing necessary system information gathered on-the-fly [11].

A runtime observability infrastructure with small hardware and communication overheads would be more than compensated by the degree of freedom achieved using adaptation. The prime challenges for runtime observability are scalability, flexibility, non-intrusiveness, real-time capabilities and cost. For the monitoring components to be as non-intrusive as possible, they need to keep their interference with system execution (probe effects) at a minimum [11].

To achieve dynamic re-mapping or cores in a NoC architecture (OASIS in this case), a monitoring infrastructure is needed to facilitate run time observability. For the Oasis NoC what needs to be monitored is the communication between cores (frequency of communication). Each packet coming out of a core has information about its destination. This information(destination address of packets) is what needs to be monitored for all the packets coming out of all cores.

2.8.1 Network Probes Architecture.

The Network probes are small hardware modules that have the task of capturing destination address of packets and sending that information to a Central Monitoring Unit (CMU). The probes can be viewed as small switches through which packets cross. Of course the functionalities of the probes are minimal compared to that of switches.

The probes will have the following parts

1. Input port for incoming packets from switches
2. Input buffers to buffer incoming packets

3. Output port for out going packets
4. A cross bar switch that determines the destination address of each packet, the cross bar switch also has a packet sending mechanism with which it will send a packet (containing destination address of packets crossing the probe) to the CMU.

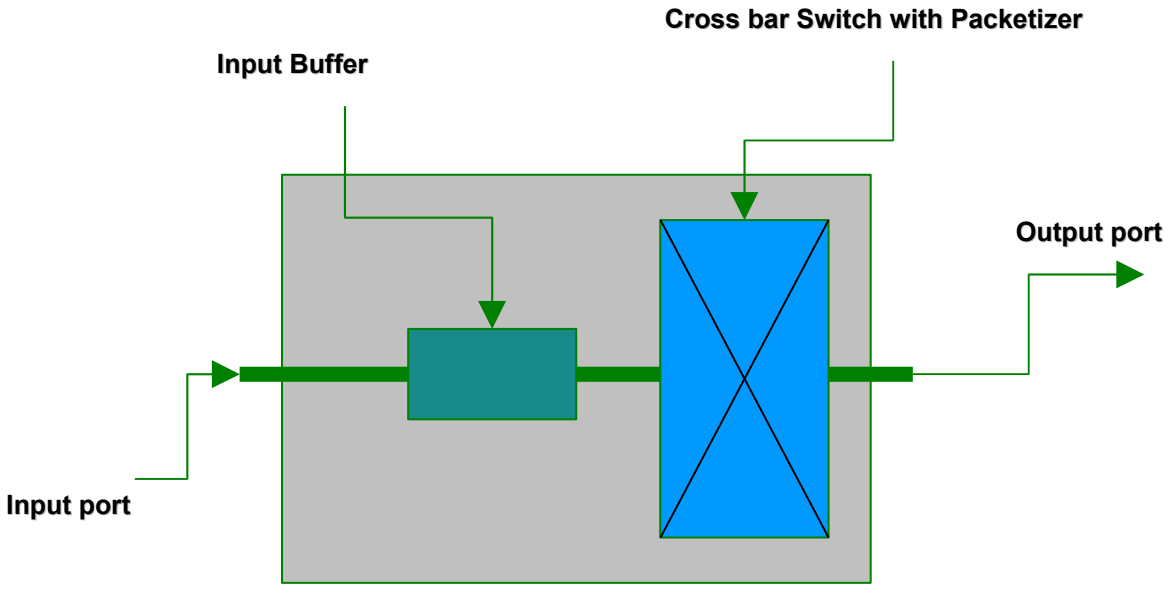

Figure 13 : Probe Architecture.

To minimize traffic generated by the probes, only header flits will be analysed and destination information contained in the header flits is added to the probe packet and forwarded to the CMU. The monitoring packet must be of a higher priority to allow it to pre-empt regular connections in the virtual channel buffers (VCB). When using only two priorities, one for regular traffic and one for monitoring traffic,a packet's priority requires a one-bit field in the header flit.

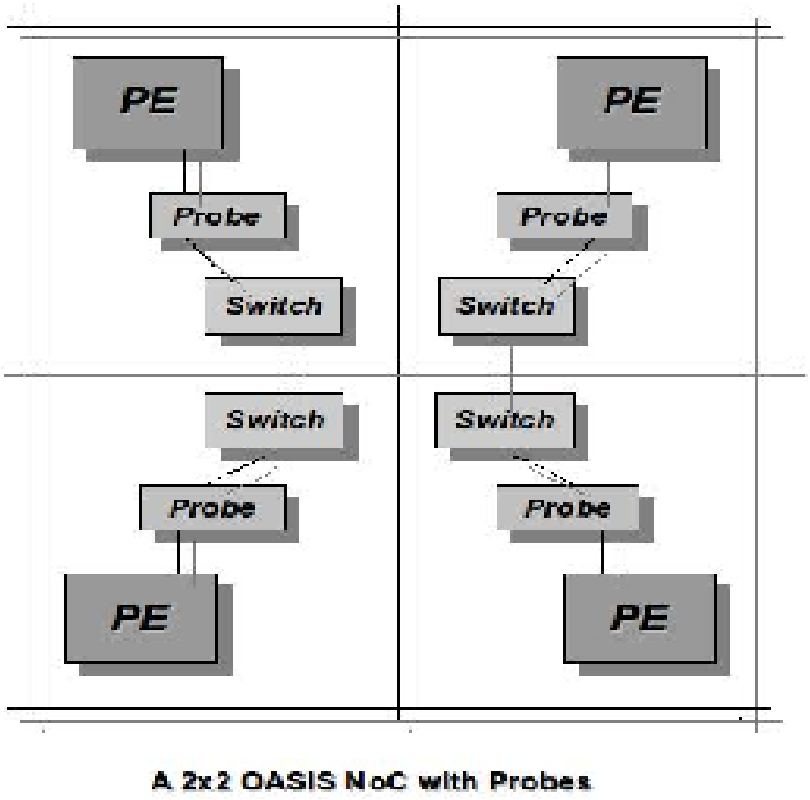

Figure 14: A 2x2 OASIS NoC with 4 cores 4 switches (S) and 4 probes (P).

Chapter III

3.0 Application Mapping:

Each Application that runs on Multiprocessor System on chip (MPSoC) can be partitioned into different tasks. This tasks in turn are assigned to the different processors on the MPSoC, this is called Application Mapping and it can be done using Task graph. Task graphs are graphs that show the relationship and sequence of execution between different tasks of an application.

The author of [15] summarized mapping as a problem of "determining how to map an application onto the NoC platform, while certain metrics of interest (energy, performance, etc.) are optimized. Depending on the flexibility of the given NoC platform, the mapping problem may have different flavours [21]. More precisely, given an application task graph, already selected IP cores for each task, the problem is to determine the topological mapping of the IP cores (also the tasks) onto proved to be NP-hard[15].

The author in [15] went ahead to further discuss application mapping in general and also discussed other factors affecting application mapping (routing strategy, and also scheduling policy). He also observed that most papers address both mapping and routing problems together and also states that the problem size of application mapping is too large if putting the three matters together to solve.

The following observations were also made is [15] " Hu et al solve the problem of mapping a group of applications (denoted by task graphs) onto a regular NoC architecture, as well as the selection of the communication routing paths based on the mapping, with the constraint of runtime performance of the applications, and the objective of optimized energy consumption [22].

The problem is solved by branch-and-bound approach: try all possible positions of mapping a task and evaluate the energy cost on the way. The state space explosion problem is solved by limiting the length of the entry queue for possible mappings.

Manolache et al address the communication mapping problem in [23] and propose a fault and energy-aware mapping strategy that provide guaranteed latency and packet delivery probability, and minimized energy at the same time. The guaranteed packet delivery is achieved by distributing redundant copies. The number of copies and the routing paths are carefully selected according to the system requirement and the energy consumption analysis. Two metrics are defined to quantify the packet delivery probability and energy consumption. The algorithm is designed based on the greedy principle.

Lei et al propose an application mapping technique to minimize the overall execution time of the application in [24]. The approach uses a 2-step genetic algorithm for application mapping and uses as-soon-as-possible (ASAP) or as-late-as-possible (ALAP) heuristics for application scheduling. The system delay is addressed by the running time of the critical path, and it is optimized by two steps: network assignment and application mapping. The authors design a genetic algorithm to solve each problem respectively. This paper specially addresses the problem of network assignment, which is not advised in this report.

There are some other researches focusing on application mapping. Ykman-Couvreur et al propose a static mapping algorithm that is combined with run-time management consideration [25]. Li et al propose a software pipelining model to improve software mapping flexibility and execution efficiency [26].”

How mapping of the application onto the hardware platform is done is important for the end result. The ideal application mapping should minimize memory usage, processing requirements, communication, and power consumption to a global optimum for the application. Mapping is a known NP-hard problem so there is little hope to find the optimum solution for other than the smallest problems [27].

The problem can be formulated as follows. Given a platform with processing elements ***E***, a set of tasks ***T*** , and a communication graph ***C***, find a mapping ***M*** that fulfils the constraints.

$$M = Fmap\ (E, T, C) \qquad (3.0)$$

The mapping function ***Fmap*** must be somewhat tailored to the application area for near optimal results as stated above. The result of the mapping is basically a set of tuples (<***Ei , Tj*** >) that describes which task(s) go on which processing element.

Definition 1: An application Task Graph (TG) is a directed graph ***G = (T, F)*** where ***T*** is the set of all tasks (computational module), ***ti ∈ T*** describing a task and ***fi,j ∈ F*** represents the data transmission from task ***ti*** to ***tj*** . Each ***fi,j*** has associated a value ***V (fi,j)*** describing the communication volume in bits transmitted between the tasks ***ti*** and ***tj***.

Moreover, each ***ti*** may be annotated with several other information, e.g. execution time on different types of processing elements, energy consumption depending on the type of the processing elements, task deadline, etc. The task graph set G is the set of all task graphs ***Gk*** [4].

Definition 2: The application mapping function is a function ***M Pt : G → P N*** which maps an application onto the physical NoC network either at design time for the application-specific NoC or at runtime for the adaptive NoC. The function changes in discrete time intervals [4].

3.1 Static Mapping:

According to [24] A mapping is called static if the resource on which it is going to be executed is decided before its execution and is not changed thereafter. Static mapping is the mapping done at design time. It is the default mapping with which the MPSoC initially executes the different tasks of application. All mapping algorithms are aimed at bringing frequently communicating cores as close as possible so as to reduce the communication overhead.

But due to different communication demands by different applications, static mapping is unable to capture all the communication characteristics of the many application that could be executed on today's MPSoC.

With the above shortcomings of static mapping, it is therefore important to propose a monitoring infrastructure that can observe the communication requirements of an application at runtime and an algorithm that will dynamically re-map the cores to reduce communication overhead and ultimately reduce power dissipated due to communication on the NoC.

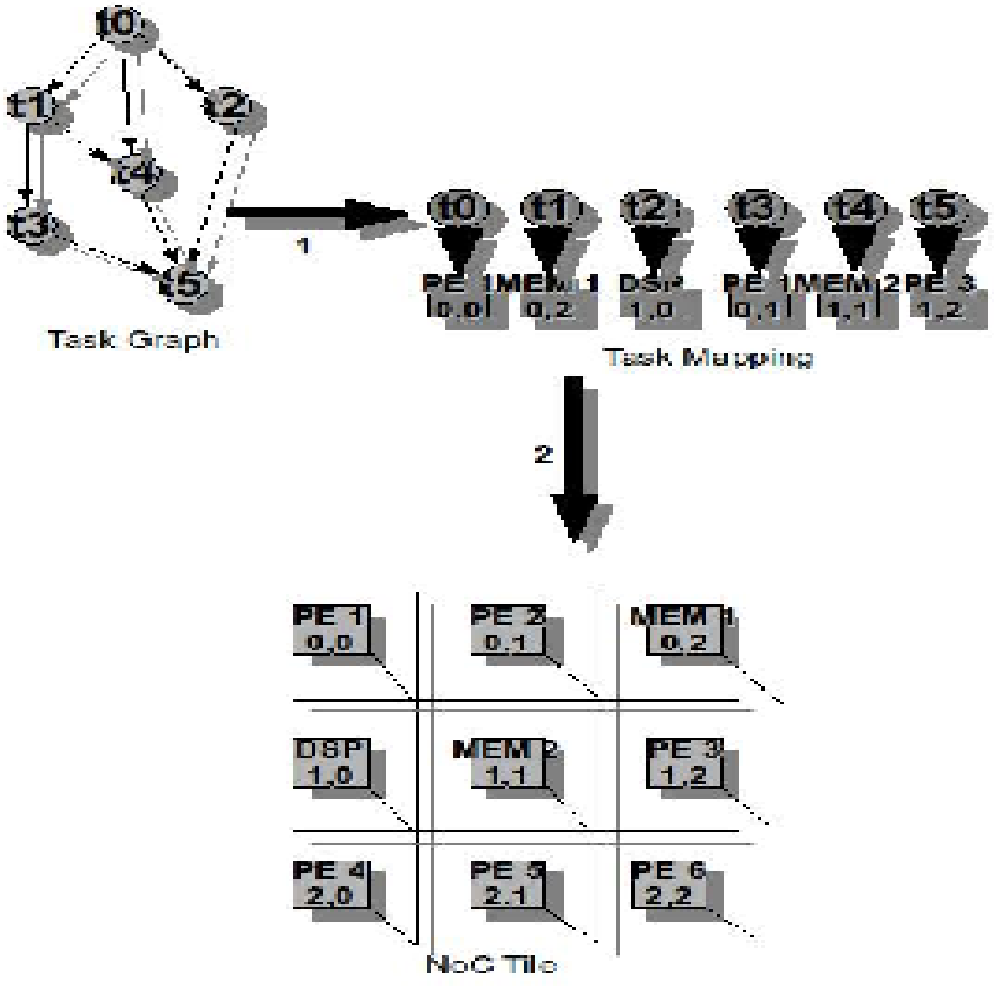

Figure 14: Application Mapping

3.2 Dynamic Mapping:

A mapping is called dynamic if the placement of task could be changed during execution of the application [24]. The Dynamic Mapping is the mapping done at run time, this targets the adaptive NoCs. As mentioned before, different applications exhibit different communication demands at runtime, this makes applications behaviour hard to predict before they are actually executed on SoCs. It is therefore important to have a way of mapping the cores at runtime.

Applications running in heterogeneous MCSoCs, as multimedia and networking, normally contain a dynamic workload of tasks. This implies a varying number of tasks simultaneously running, with their number possibly exceeding the available resources. This may require the execution of task mapping at run time, to meet real-time constraints [32].

Task migration has also been used in heterogeneous MPSoCs to optimize the performance at run-time. Task migration relocates tasks either when a performance bottleneck is detected or to distribute the work-load more homogeneously among the MPSoC processors. Differently from task migration, dynamic mapping can insert new tasks into the system at run time [32].

3.3 Dynamic Remapping Algorithm:

The dynamic remapping algorithm which is the main contribution of my thesis is aided by a monitoring infrastructure. The monitoring infrastructure make use of monitoring probes (which was discussed in the previous chapter). The monitoring probes which are attached to switches (in Our OASIS NoC) gather information about packet destination and sent to the central monitoring unit (CMU).

In the CMU, the Lookup table (LUT) is populated with traffic information monitored by the probes. To discuss the Dynamic remapping algorithm proposed in this thesis work, I will first go over the look up table (LUT) in the Central monitoring unit. An example of the LUT is shown below:

Cores	**Core Address**	**Recipients**	**Total packets sent**
Core 1	0,0	Core 1| Core 2| Core 3.....| Core n * 10 3 12	25
Core 2	0,1	Core 1| Core 2| Core 3.....| Core n 3 * 8 1	12
Core 3	1,0	Core 1| Core 2| Core 3.....| Core n 5 10 * 6	21
…..	…..	….....	…...
Core n	…...	Core 1| Core 2| Core 3.....| Core n 2 4 3 *	9

Table 2: Look up table

The first column (Cores) shows all the IP cores connected in a MPSoC. The second column (Core address) shows the address of each core on a mesh/torus/folded torus topology. The third column shows all other cores communicating with the core in the first column, it also shows the number of packets that are sent from the core in the first column to these recipient cores (* indicates self). The last column (total packets sent) shows the total number of packets that have been sent by the core in the first column.

The above LUT can also be represented in terms of communication weighted model*(CWM).*

Definition 3: A communication weighted graph ***(CWG)*** is a directed graph ***<C, W>***. The set of vertices ***C = {c1, c2..., cn}*** represents the set of cores in one application. Assuming ***wab*** is the number of bits of all packets sent from a core ***ca*** to a core ***cb***, then the set of edges ***W*** is ***{(ca, cb) | ca, cb ∈ C and wab != 0}***, and each edge is labeled with the value ***wab***. ***W*** represents all communications between application cores, while ***CWG*** reveals information of application relative communication volume. [12]

3.2.1 The Algorithm:

1. Let ***C = {c1,c2,c3.....,cn}*** be the set of all cores.
2. Let ***wab*** be the communication volume between core *a* and core *b* and ***W*** be a set of all communication volumes such that ***W*** is ***{ (ca, cb) | ca, cb ∈ C and wab != 0 }.***
3. Let ***N = {Na, Nb, Nc.....Nn}*** be the total number of packets sent by core ***a,b,c****.....and* ***n*** respectively.
4. Select a core ***ci*** such that ***ci*** has an ***Ni*** greater than all other cores. This core is now the root core and the root of of task graph.
5. Repeatedly select a core ***ci*** from ***C*** which has the highest ***w*** *(communication volume)* with the root and make it a neighbour of the root.
6. Repeat step 5 for all the neighbours of the root until no core remains. The system will remain with this current mapping for a particular time period ***t***.
7. After the time period ***t*** has elapsed, the LUT is checked to see if the core with the highest ***Ni*** has changed,
    1. if it has changed, a new root is selected
        1. if the most frequently communicating cores are already neighbours of the new root, then no remapping is done. Else step 4-6 are repeat to get a new mapping.
    2. Else if the root is still the same, then the system is not remapped.

Below is a flow chat of the above algorithm.

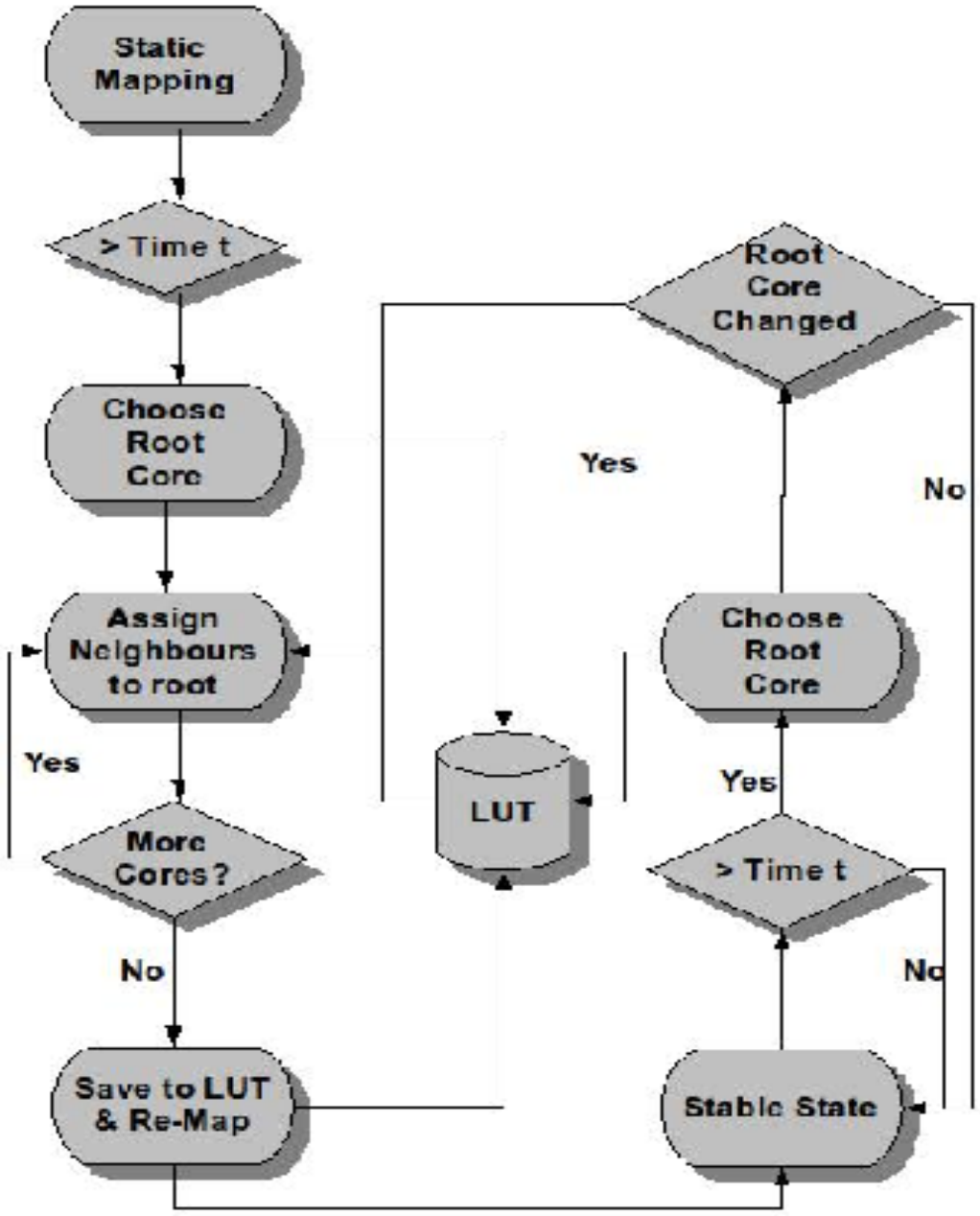

Figure 15: Flow chat of Dynamic remapping algorithm.

3.3.2 Algorithm Discussion:

The SoC based NoC will start with the default mapping. After an elapsed time:

1. The PE sending the highest number of packets is selected as a root PE.
2. All other PEs that receive packets most frequently from the root are placed as the neighbours of the root PE.
3. For each neighbour of the root PE, it will have PEs that it frequently communicates with as its own neighbours (this is not including PEs already considered).
4. Step 3 is repeated for all the Neighbours of the root PE until no PE is left.
5. The system (SoC Oasis in our case) will remain at this state until a PE(I will call it potential root) other than the root is sending more packets out than the root.
6. The CMU will now check the packets being sent out by the potential root, if the destination address of the packets are already neighbours of the potential root, then no re-mapping is done.

   Else if the potential root has a core it communicates with which is not its immediate neighbour then a re-mapping is done by selecting the neighbour of the potential root whose communicating frequency with the potential root is minimal or non at all and replace it with the core communicating with the potential root.
7. Hence the potential root now becomes the new root.
8. To avoid an unstable system that will require too much re-mapping. A time interval will be defined in which re-mapping is not done.
9. After the time period has elapsed, then the CMU will check the LUT to find out if there is any new potential root.

Chapter IV

4.0 Problem Statement:

Simply stated, for a given application, The objective is to decide on which tile should each IP core be mapped to, based on the communication volume of the IP core, such that the total energy consumption during communication is minimized. Effective re-mapping brings frequently communicating cores closer and reduces the number of links and routers crossed by packets.

4.1 Simulation Set up and Algorithm Implementation:

We assume the platform under consideration is a n×n mesh NoC. We further assume that minimum deterministic routing algorithm (XY- routing) is used to direct the packets across the network in the NoC. In addition, we assume that each IP-core has a probe monitoring the packets coming out of it. Base on the destination address of the header flit, the probe send a packet with that information to the Central monitoring unit (CMU). Under such an architecture, the *Ebit* energy model as presented in Section 2.2 can be used to derive the energy consumption of the system analytically.

The first step is to run an application using the static mapping. Using Noxim [29] and worm_sim [30] which are two NoC simulators an application is executed. ***The total received packets, total received flits, global average delay (cycles), global average throughput (flits/cycle), throughput (flits/cycle/IP), Maximum delay (cycles) and Total energy (J)*** consumed are generated by the simulators.

The second step is to analytically apply the re-mapping algorithm to the statically mapped application, the resulting mapping is now used to re-map the application.

The third step is to execute the re-mapped application on the NoC simulators. ***The total received packets, total received flits, global average delay (cycles), global average throughput (flits/cycle), throughput (flits/cycle/IP), Maximum delay (cycles) and Total energy (J)*** consumed for the re-mapped application are also generated by the simulators.

The result of the two simulations for the static mapping and dynamic mapping are compared.

4.2 Validation of Result and Performance Evaluation (Case Study):

To validate the result of the simulations, I will use real examples. The NoC to be used is the OASIS NoC, I will implement a 3x3 OASIS NoC and initially use TGFF [31] a task graph generating tool to generate my tasks graph, this task graph is then mapped to IP cores statically. Then Using a hypothetical Lookup Table (LUT), I will use the proposed re-mapping algorithm to re-map the IP cores.

Next I will use the *Ebit* energy model as presented in Section 2.2 to show that when communicating cores are brought closer by dynamic remapping, the total energy consumed due to communication is significantly reduce.

The figure below is an example of a hypothetical Lookup table (LUT) in the CMU. The LUT is randomly generated. Using the data in this table, I will generate the corresponding communication weighted graph<CWG>. Subsequently I will map the cores according to the proposed re-mapping algorithm.

4.2.1 The Lookup Table:

Cores	Address	Recipients Cores									Total
Core 1	***0,0***	***Core 1***	***Core 2***	***Core 3***	***Core 4***	***Core 5***	***Core 6***	***Core 7***	***Core 8***	***Core 9***	34
		***	*5*	*15*	*11*	*2*	*0*	*0*	*1*	*0*	
Core 2	***0,1***	***Core 1***	***Core 2***	***Core 3***	***Core 4***	***Core 5***	***Core 6***	***Core 7***	***Core 8***	***Core 9***	29
		3	***	*4*	*15*	*2*	*0*	*2*	*2*	*0*	
Core 3	***0,2***	***Core 1***	***Core 2***	***Core 3***	***Core 4***	***Core 5***	***Core 6***	***Core 7***	***Core 8***	***Core 9***	25
		12	*1*	***	*5*	*3*	*1*	*1*	*1*	*1*	
Core 4	***1,0***	***Core 1***	***Core 2***	***Core 3***	***Core 4***	***Core 5***	***Core 6***	***Core 7***	***Core 8***	***Core 9***	29
		9	*8*	*3*	***	*1*	*0*	*2*	*5*	*1*	
Core 5	***1,1***	***Core 1***	***Core 2***	***Core 3***	***Core 4***	***Core 5***	***Core 6***	***Core 7***	***Core 8***	***Core 9***	29
		1	*2*	*1*	*1*	***	*3*	*4*	*5*	*11*	
Core 6	***1,2***	***Core 1***	***Core 2***	***Core 3***	***Core 4***	***Core 5***	***Core 6***	***Core 7***	***Core 8***	***Core 9***	19
		0	*0*	*2*	*0*	*2*	***	*2*	*5*	*8*	
Cor	***2,0***	***Core 1***	***Core 2***	***Core 3***	***Core 4***	***Core 5***	***Core 6***	***Core 7***	***Core 8***	***Core 9***	18
		0	*2*	*1*	*2*	*5*	*1*	***	*4*	*3*	

e 7			
C or e 8	***2,1***	***Core 1|Core 2|Core 3|Core 4|Core 5|Core 6|Core 7|Core 8|Core 9*** *2 1 2 7 7 1 3 * 2*	25
C or e 9	***2,2***	***Core 1|Core 2|Core 3|Core 4|Core 5|Core 6|Core 7|Core 8|Core 9*** *0 0 2 2 1 5 1 1 *	12

Table 3: An arbitrary Lookup table for 3x3 NoC Architecture.

4.2.2 Step 1 Selecting The Root:

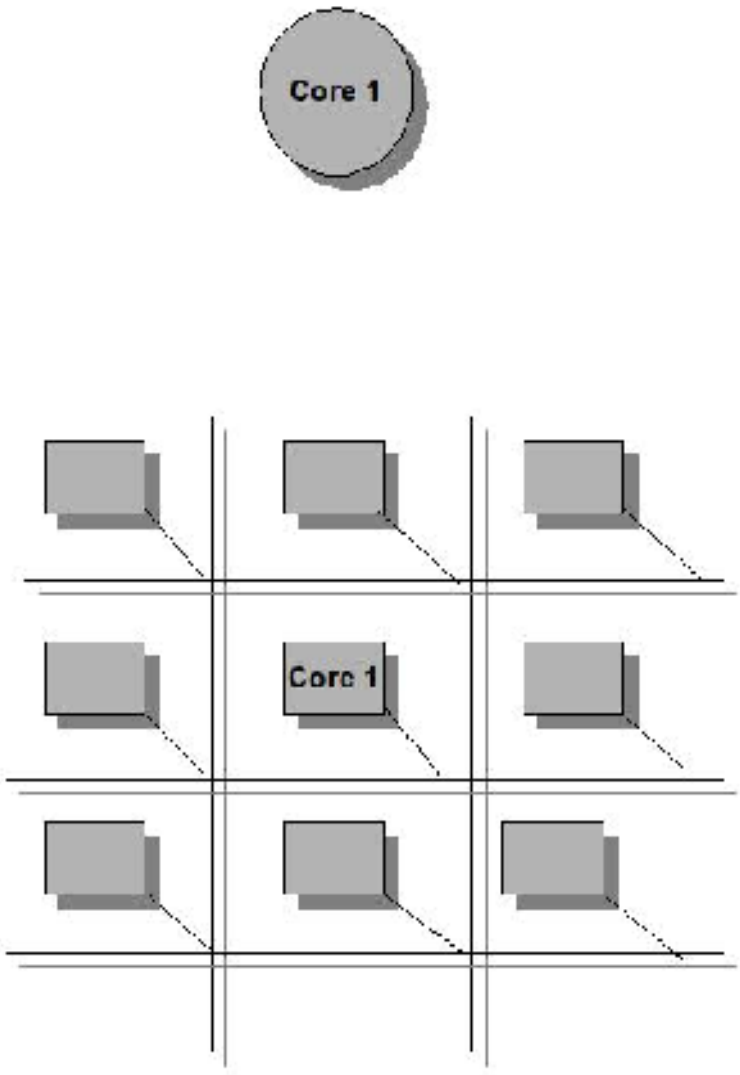

Figure16: Selecting the Root core (Based on the above LUT).

4.2.3 Step 2 Selecting Neighbours of the Root:

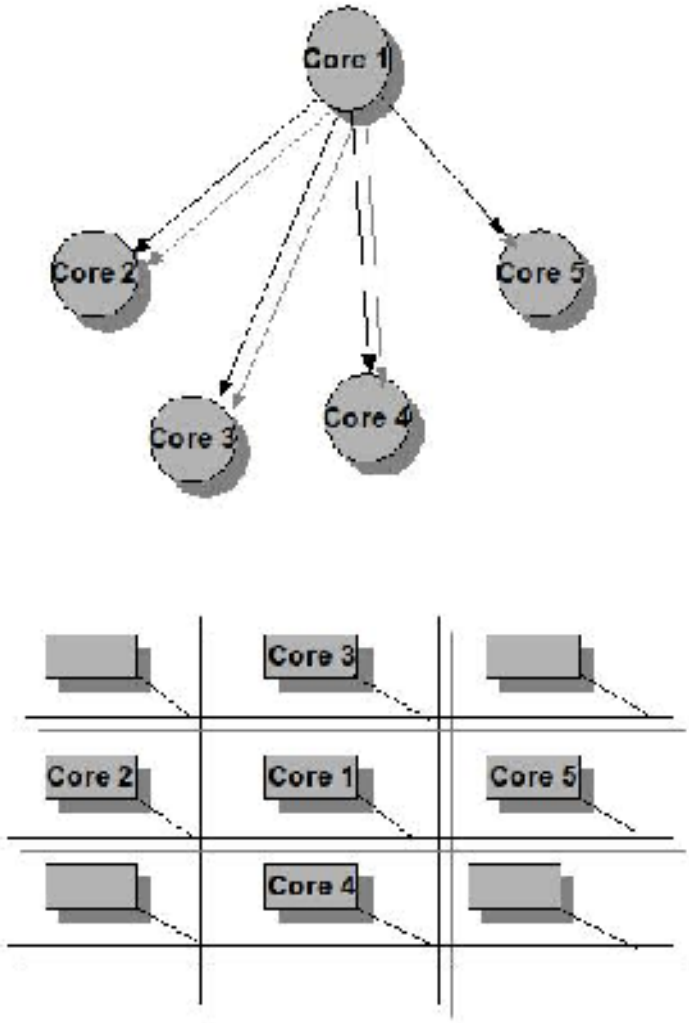

Figure17: Step 1: Selecting the neighbour of the root core: based on the above LUT.

4.2.4 Selecting Neighbours of the Neighbours:

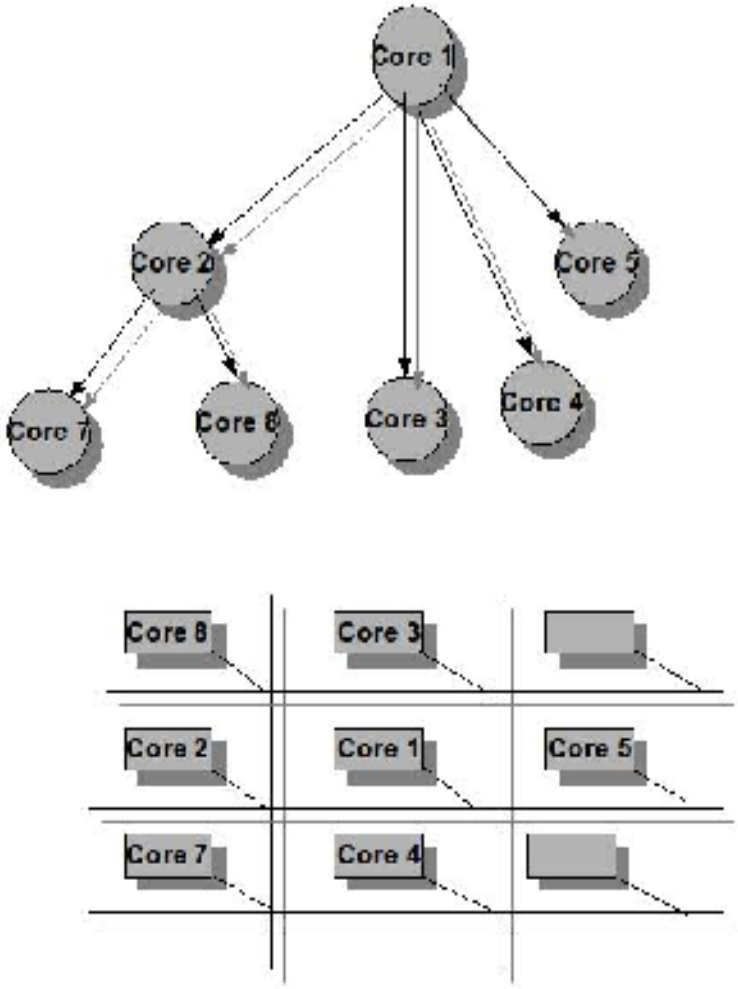

Figure18: Step 2: Selecting the neighbour of the root core: based on the above LUT.

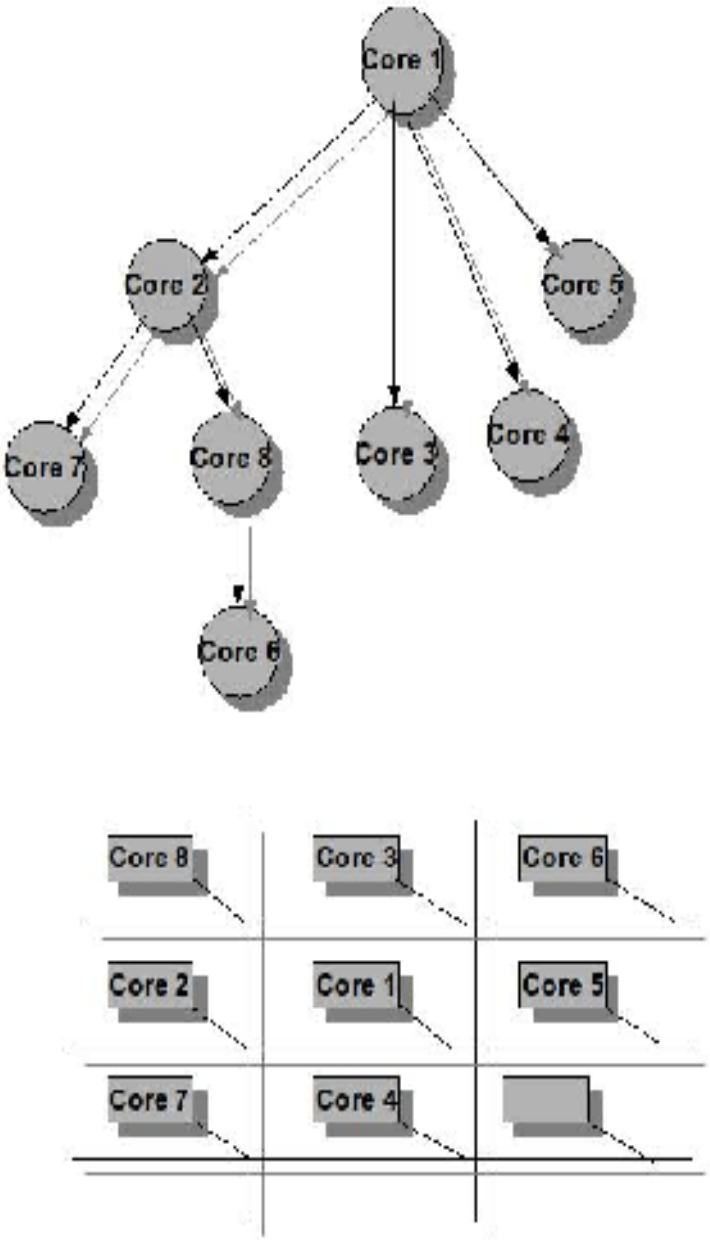

Figure19: Step 3: Selecting the neighbours of the root core: based on the above LUT.

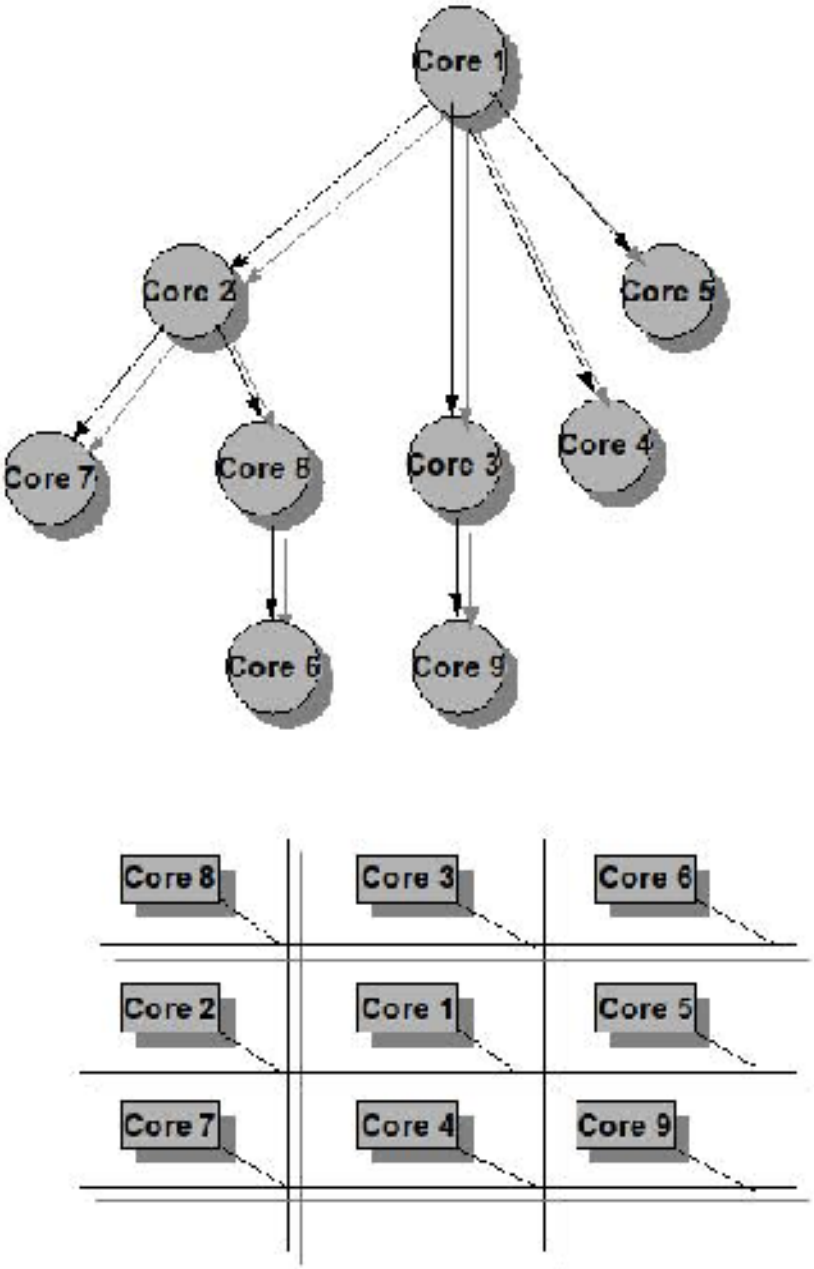

Figure 20: Step 4: Selecting the neighbour of the Root core: based on the above LUT.

The energy model described in section 2.2 is based on the number of switches crossed by data packets from source to destination (*nhops*), the energy consumed in the switches *(ERbit)* and the energy consumed by the number of interconnecting links *(Elbit)* from source to destination.

$$E_{bit}^{ti,tj} = nhops \times ERbit + (nhops - 1) \times Elbit$$

It is clearly visible that due to dynamic re-mapping, this three factors *(nhops,ERbit,Elbit)* that contribute to energy consumption on the NoC are reduce to the minimum among frequently communicating cores. This can also be seen from simulation results (Noxim) that show improvements in the ***global average delay (cycles), global average throughput (flits/cycle), throughput (flits/cycle/IP), Maximum delay (cycles) and Total energy (J)*** consumed when the re-mapping algorithm is used to simulate an application.

Before Dynamic Remapping of Cores	**After Dynamic Remapping Cores**
Total energy (J): 1.47477e-05	*Total energy (J): 1.10477e-05*
Total received packets: 1424	*Total received packets: 1724*
Total received flits: 8475	*Total received flits: 10,475*
Global average delay (cycles): 10.4284	*Global average delay (cycles): 7.4284*
Global average throughput (flits/cycle): 0.0629742	*Global average throughput (flits/cycle): 0.0929742*
Throughput (flits/cycle/IP): 0.0588542	*Throughput (flits/cycle/IP): 0.0888542*
Max delay (cycles): 72	*Max delay (cycles): 51*

Table 4: Noxim Simulation results for cores before and after remapping.

4.3 Conclusion and Future Work:

Research has shown that significant amount of energy is consumed by the communication infrastructure: The NoC. But with the algorithm presented, the amount of energy consumed on the NoC can be significantly reduced. A comprehensive analysis and comparison with other low power techniques in NoC is needed to show that the re-mapping algorithm proposed above provides a better power reduction mechanism.

After using the re-mapping algorithm, it can be established that the link length between frequently communicating cores has been reduced, the number of buffers crossed from source to destination is also reduced. Reduced switching requirements in switches due to decrease in the number of switches crossed is also achieved.

It can also be concluded that the extra area consumed by the monitoring probes is negligible compared to the flexibility and performance improvement that is achieved. The Algorithm proposed is based on a reconfigurable architecture and therefore targets reconfigurable devices like the FPGA board, therefore a reconfigurable device will be targeted to implement the algorithm, this will entail developing the following:

1. A soft-core processor to implement the LUT and re-mapping algorithm (the CMU).
2. Monitoring probes that can be added to a NoC and connected to the CMU.

The re-mapping algorithm performs better with an average number of cores, this is because communication increases with increase in the number of cores. But when the number of cores is really high, there is a huge overhead associated with re-mapping the entire cores at runtime, this overhead can be alleviated by grouping cores into clusters. In the future, the algorithm should take into consideration cores in a cluster, re-mapping should be done for a particular cluster, this is in an effort to reduce the overhead of dynamically remapping large number of cores at runtime.

Appendix

Noxim:

Noxim is a Network-on-Chip Simulator developed at the University of Catania (Italy) using C++ and SystemC. Noxim has a command line interface for defining several parameters of a NoC. In particular the user can customize the network size, buffer size, packet size distribution, routing algorithm, selection strategy, packet injection rate, traffic time distribution, traffic pattern, hot-spot traffic distribution.

The simulator allows NoC evaluation in terms of throughput, delay and power consumption. This information is delivered to the user both in terms of average and per-communication results. In detail, the user is allowed to collect different evaluation metrics including the total number of received packets/flits, global average throughput, max/min global delay, total energy consumption, per-communications delay/throughput/energy etc.

Worm_sim:

Worm_sim is a cycle-accurate simulator which was developed from scratch in C++ using standard template library. Worm_sim is written with flexibility and modularity in mind. It can be used to simulate a wide range of NoC architectures (e.g. NoCs with different topologies and different routing algorithms, etc.), using user controllable performance parameters (e.g. channel buffer size, routing engine delay, crossbar arbitration delay, etc.). Moreover, due to the flexibility of worm_sim, it can be easily extended to simulate NoCs which worm_sim does not support at the current stage.

Worm_sim's built-in traffic generator allow users to simulate the system under several popular traffic patterns, such as uniform traffic pattern, hot spot traffic pattern, transpose traffic pattern, etc. Worm_sim also provides an efficient way to allow user specify arbitrary traffic condition for the NoC under simulation. More precisely, user has the control over the packet generating rate at individual IP node, the size of the packet and its distribution, etc.

Worm_sim allows user to attach a trace file for each individual IP node, so that the system under simulation can mimic the exact traffic condition of realistic applications by reusing the trace files retrieved from those applications.

TGFF:

Task graphs for free (TGFF) was designed to provide a flexible and standard way of generating pseudo-random task-graphs for use in scheduling and allocation research. This includes the areas of embedded systems, hardware/software co-design, operating systems (both real-time and general-purpose), parallel or distributed hardware or software studies, flow-shop scheduling, as well as any other area which requires problem instances consisting of partially-ordered or directed acyclic graphs (DAGs) of tasks, i.e. task graphs.

References

1. M. Huebner & J. Becker. “Multiprocessor System-On-Chip: Current Trends and the Future” *Internet*: http://www.dac.com/events/eventdetails.aspx?id=95-251 [Sep. 23 2010].

2. Wikipedia, the free encyclopaedia. “Multi-core processor” *Internet*: http://en.wikipedia.org/wiki/Multi-core Sep.21 2010 [Sep. 23 2010].

3. B. Abderazek and M. Sowa. “Basic Network-on-Chip Interconnection for Future Gigascale MCSoCs Applications: Communication and Computation Orthogonalization” *Proceedings of the TJASSST2006 Symposium on Science, Society and Technology,* Sousses, Dec. 2-4, 2006. pp. 1-7.

4. M. A. A Faruque. “Runtime Adaptive System-on-Chip Communication Architecture” PhD. Dissertation, University of Karlsruhe, Karlsruhe, Germany, 2009

5. Calin Ciordas et al. “NoC Monitoring: Impact on the Design Flow”. *In Proceedings of International Symposium on Circuits and Systems (ISCAS),* May 2006.

6. T. Simunic and S. Boyd “Managing Power Consumption in Networks on Chips” *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition,* 2002.

7. Soteriou V, Peh L-S. “Design-space exploration of power-aware on/off interconnection networks”. *In: ICCD’04: Proceedings of the IEEE international conference on computer design;* 2004. p. 510–7.

8. Srinivasan, K. and K. S. Chatha. “A low complexity heuristic for design of custom network-on-chip architectures”. *Proceedings of the 2006 Design, Automation and Test in Europe Conference and Exhibition,* 2006.

9. Caroline Concatto, Debora Matos, Luigi Carro, Fernanda Kastensmidt, Altamiro Susin and Marcio Kreutz. “NoC Power Optimization Using a Reconfigurable Router”. 2*009 IEEE Computer Society Annual Symposium on VLSI.* 2009.

10. Ciprian Seiculescu et al. “NoC Topology Synthesis for Supporting Shutdown of Voltage Islands in SoCs” *Design Automation Conference, 2009. Proceedings.* July 26 – 31 2009

11. M.A.A Faruque et al: “Runtime Observability for an Adaptive Network on Chip Architecture (ROAdNoC)”. *DAC’08: Procedure of the 45th Conference on Design Automation*, 2008.

12. C. Marcon et al. “Exploring NoC Mapping Strategies: An Energy and Timing Aware Technique”.*Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE’05),* 2005.

13. Ahmad, B.; Erdogan, A.T.; Khawam, S. “Architecture of a Dynamically Reconfigurable NoC for Adaptive Reconfigurable MPSoC” *First NASA/ESA Conference on Adaptive Hardware and Systems* 2006 (AHS-2006), pp. 405- 411, 15-18 June 2006.

14. Luca Benini, Giovanni DE Micheli , “Powering Networks on Chips: Energy Efficient and Reliable interconnect for SoCs.”. *System Synthesis, 2001. Proceedings. The 14th a International Symposium* on 2001 Page(s) : 33 - 38.

15. Weichen Liu. “ Efficient Application Mapping and Scheduling for Networks-on-Chip” Department of Computer Science and Engineering Hong Kong University of Science and Technology April 10, 2008

16. S. Pasricha and N. Dutt. “On-Chip Communication Architectures: System on Chip Interconnect (Systems on Silicon)”. Morgan Kaufmann, April 18, 2008.

17. G.-M. Chiu, “The odd-even turn model for adaptive routing,” *Parallel and Distributed Systems, IEEE Transactions* on, vol. 11, no. 7, pp. 729–738, Jul 2000.

18. Ankur Agarwal, Boca Raton: “ Survey of Network on Chip (NoC) Architectures & Contributions”. *Journal of Engineering, computing and architecture* Volume 3, Issue 1, 2009.

19. A. Hansson, M. Coenen, and K. Goossens, “Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip,” *Design, Automation & Test in Europe Conference & Exhibition*, 2007. DATE ’07, pp. 1–6, 16-20 April 2007.

20. A. Hansson and K. Goossens, “Trade-offs in the configuration of a network on chip for multiple use-cases,” *Networks-on-Chip, 2007. NOCS 2007. First International Symposium* on, pp. 233–242, 7-9 May 2007.

21. R. Marculescu, J. Hu, and U. Ogras, “Key research problems in noc design: a holistic perspective,” *Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS ’05. Third IEEE/ACM/IFIP International Conference* on, pp. 69–74, Sept. 2005.

22. J. Hu and R. Marculescu, “Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures,” *Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 688–693, 2003.

23. S. Manolache, P. Eles, and Z. Peng, “Fault and energy-aware communication mapping with guaranteed latency for applications implemented on noc,” *Design Automation Conference, 2005. Proceedings.* 42nd, pp. 266–269, 13-17 June 2005.

24. T. Lei and S. Kumar. “A two-step genetic algorithm for mapping task graphs to a network on chip architecture,” *Digital System Design, 2003. Proceedings. Euromicro Symposium*.

25. C. Ykman-Couvreur, V. Nollet, T. Marescaux, E. Brockmeyer, F. Catthoor, and H. Corporaal, “Design-time application mapping and platform exploration for mp-soc customised run- time management,” *Computers & Digital Techniques*, IET, vol. 1, no. 2, pp. 120–128, March 2007.

26. F. Li, M. Kandemir, and I. Kolcu, “Exploiting software pipelining for network-on-chip architectures” *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium* on, vol. 00, pp. 6 pp.–, 2-3 March 2006.

27. Daniel Wiklund. “Development and Performance Evaluation of Networks on Chip” Ph.D Dissertation, Department of Electrical Engineering Linköping University, Sweden, 2005.

28. T. Ye, L. Benini, and G. De Micheli. “Analysis of power consumption on switch fabrics in network routers”. *In Proceedings of Design Automation Conference. (DAC)*, June 2002.

29. Noxim, A Network-on-Chip Simulator developed at the University of Catania (Italy).

30. Worm_Sim: A cycle accurate simulator for Networks-on-Chip developed at System Level Design Group Carnegie Mellon.

31. Robert Dick. “Task Graph for Free” *Internet:* http://ziyang.eecs.umich.edu/~dickrp/tgff/ [Sep. 23 2010]

32. E. Carvalho N. Calazans and F. Moraes. “Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs”. *18th IEEE/IFIP International Workshop on Rapid System Prototyping(RSP'07)*. 2007