NOC RESOURCE ALLOCATION BASED ON PHYSICAL DESIGN TECHNIQUES

A Thesis

by

GONGMING YANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jiang Hu
Committee Members,	Weiping Shi
	Anxiao Jiang
Head of Department,	Krishna Narayanan

May 2014

Major Subject: Computer Engineering

ABSTRACT

Networks-on-Chip (NoC) has been recognized as a scalable approach for on-chip communication. Quality-of-Service (QoS) is a fundamental part of application specific NoCs. This thesis focuses on resource allocation on NoC, to improve the capability of NoC for Guaranteed Service (GS). A graph model is adopted to describe physical and temporal sources of a NoC. Based on the graph model, an RRR-based algorithm is proposed for simultaneous routing and time slot allocation. In addition, a negotiation-based algorithm is suggested for achieving power-efficient QoS for application-specific NoCs. Last, a hybrid NoC architecture, which combines circuit switching and packet switching, is developed and investigated. Experimental results show that our techniques outperform previous works.

DEDICATION

To all sentient beings

ACKNOWLEDGEMENTS

First, I am grateful to Buddha.

Second, it is my pleasure to express my sincere gratitude to my advisor, Prof. Jiang Hu, for his continuous support and encouragement in the past two years. His guidance during our numerous discussions gave me a strong motivation to finish this work.

I particularly thank Mr. Hao He for his great help.

I am particularly grateful to my parents, my wife, my brother and my brother in-law, for their continuously encouraging. My gratitude also goes to all beings for their supporting in all the time.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION

The performance of many modern digital systems is limited by their interconnection instead of their logic circuits or storage elements. In a high-performance system, a large amount of power is consumed to drive signals on wires and most of clock cycle time is attributed to wire delay rather than gate delay. Networks-on-chip (NoC) has been recognized as a scalable approach to cope with the increasingly large demand for on-chip communication. In NoC designs, Quality of Service (QoS) and power-efficiency are of paramount importance. This thesis focuses on application-specific NoC that has somewhat traceable traffic patterns. This is in contrast to NoCs in microprocessors where the traffics are largely random. In specific, this thesis studies three subjects: (1) simultaneous packet routing and time slot assignment for QoS, (2) power-efficient QoS, and (3) hybrid NoC architecture for power-efficient NoCs.

1.1 Background

Networks-on-Chip is an inter-communication system for an integrated circuit between intellectual-property (IP) blocks in a System-on-Chip (SoC). It applies networking theory and methods to bring notable improvements over conventional point-to-point connection, bus and crossbar interconnections.

Bus on chip is an inter-communication system that connects all components in a chip. Modern bus can be wired in either a multi-drop or daisy chain topology or

connected by switched hubs. NoC improves the scalability and the power efficiency of complex SoCs compared with bus and other design.

Quality-of-Service (QoS) is characterized by diverse parameters, such as reliability, delay, jitter, bandwidth, packet loss, and throughput [1]. In this paper, the QoS on NoC is characterized by guaranteeing bandwidth and the maximal latency allowed for delivery for Guaranteed Service traffics.

1.2 Related Works

1.2.1 Simultaneous Packet Routing and Time Slot Assignment

When latency constraints for packets are tight, time-division multiplexing (TDM) is able to provide guaranteed performance. The TDM implementation divides time into a series of slots. The duration and the number of time slots govern the granularity of the resource to be allocated. The virtual circuit of TDM is a set of contiguous time slots spanning a routing path from the source intellectual-property block (IP block) to the destination IP block. Finding the "optimal" time slot allocation is generally NP-hard, so most realistic implementations are heuristic approaches.

There are several early works studying the time slot allocation problem. A. Hansson, et al. [2], performs packet routing and time slot assignments separately. The work of Z. Lu, et al. [3] defines a new concept of a logical network (LN) and investigates how to allocate VC with LN to avoid conflict of paths, and performs time slot allocation and routing simultaneously using a brute-force method, which is computationally very expensive for large size NoCs.

A graph model is proposed in [4] to describe the physical resources and time slot based on the notion of time plane. With this model, routing and time slot assignment can be conducted simultaneously by finding disjoint paths on the graph. Then, the problem is similar as global routing in chip layout. In this regard, J. A. Roy, et al. [5] proposes a Fairly Good Router (FGR) based on the framework of Ring-up-and-Re-Route (RRR). RRR is a straightforward yet very effective approach to solving contentions in routing. FGR improves RRR penalty function over its previous works and hence increases opportunities of finding feasible solutions.

1.2.2 Power-Efficient QoS for Application Specific NoCs

Bandwidth utilization inevitably affects power-efficiency, which is crucial yet largely neglected in prior NoC QoS methods. NoC capacity optimization is studied in [6]. It is an iterative greedy heuristic for minimizing link capacities. It iteratively routes a flow among the minimal cost paths in the network. However, QoS is not handled in this work. Perhaps the only work that seems to touch both power-efficiency and NoC QoS is [7]. It mainly solves task mapping and scheduling assuming fixed routing as well as fixed link and buffer capacity.

1.2.3 Hybrid Architecture for NoC

The book by W. Dally et al. [8] is a classic literature on computer interconnect network. Many NoC techniques, such as packets switching, wormhole routing and virtual-channel flow control, are largely borrowed from techniques described in this book. Later, the work of K. Goossens, et al. [9], proposes a network on silicon (NoS) to implement the communication among IP blocks. This work tries to design a router

network with packet switching techniques to reduce wire congestion between IP blocks and enable scalable integration of IP blocks. Also, this work proposes a combination of circuit and packet switching in the spirit of ATM [10], and used TDM circuit switching for Guarantee Services and virtual output queuing for Best Effort traffics. Based on those previous works, the TDM technique is better to support circuit switching; however, the packet switching is more effective on handling Best Effort traffics.

1.3 Problem Formulation

1.3.1 Simultaneously Resource Allocation and Routing

Given a resources Graph $G(V,E)$ of a NoC and a set of GS traffic flits $\Phi = \{f_1, f_2, \ldots f_p\}$.

Objectives: Generate paths $p = \{p_0, p_1, \ldots, p_m\}$ and allocate time slot simultaneously for each flit f_i.

Constraints: The utilization ue for any edge $e \in E$ is no greater than Ce, the capacity of this edge, latency constraint of each flit is satisfied, and all flits of the same packet should be delivered in order.

1.3.2 Power-Efficient QoS for Application Specific NoCs

Given a resources Graph $G(V,E)$ of a NoC, a set of GS traffic flits $\Phi = \{f_1, f_2, \ldots f_p\}$, and dynamic/static power model $pm(V,E)$.

Objectives: Generate paths $p = \{p_0, p_1, \ldots, p_m\}$ and allocate time slot simultaneously for each flit f_i and decide link/buffer capacity, such that the power consumption of all routers and net links in NoC is minimized.

Constrains: Satisfy the bandwidth and deadline requirement of every f_i, and delivered in order.

1.3.3 Hybrid Architecture for NoC

Objectives: Seamlessly combine circuit switching for GS traffics and packet switching for Best-Effort (BE) traffics, and design a practical algorithm to route and allocate time slots simultaneously.

CHAPTER II

APPROACHES

2.1 Algorithm for Simultaneous Routing and Time Slot Assignment

2.1.1 Resource Graph Model

The resource allocation problem can be modeled in a directed graph $G(V,E)$ where V is a set of nodes modeling routers and E is a set of edges indicating communication links. To allocate time slots and routes simultaneously, graph $G(V,E)$ is expanded into $G'(V',E')$, called expanded graph, which includes time slots information.

We define *time plane* as a set of edges in a specific time slot. Since we cannot infinitely long time horizon, we assume the traffic patterns are repeated in periodic time windows.

In order to capture the temporal aspect of the problem, the graph is expanded along time axis by duplicating the nodes V to $\{V^0,V^1,...V^{tw-1}\}$, at each *time plane* $tp=\{0,1,...,tw-1\}$ where tw is the number of *time plane*, and without loss of generality, V^i is the set of vertex exist in *time plane* i.

The edges in base $G(V,E)$ are expanded along time axis, by changing the destination of the edge to corresponding node in the next *time plane*. Transmitting one flit from V^i to V^j must be done in the only one time slot; otherwise the flit may conflict with other flit to V^j. The time difference between two adjacent *time planes* is one time slot.

For any $e^i(si,di) \in E(e^0,e^1,...,e^i,...)$ where si is the source vertex of e^i and di is the destination node of e^i. The edges based on e^i are $E_i' = \{e^0(s_0,d_1),e^1(s_1,d_2),...,e^{tw-1}(s_{tw-1},d_0)\}$, the last edge $e^{t-1}(s_{tw-1},d_0)$ wraps back to time plane 0. Overall, the edge set in expanded graph is defined as $E' = \{E_0',E_1',...,E_{tw-1}'\}$.

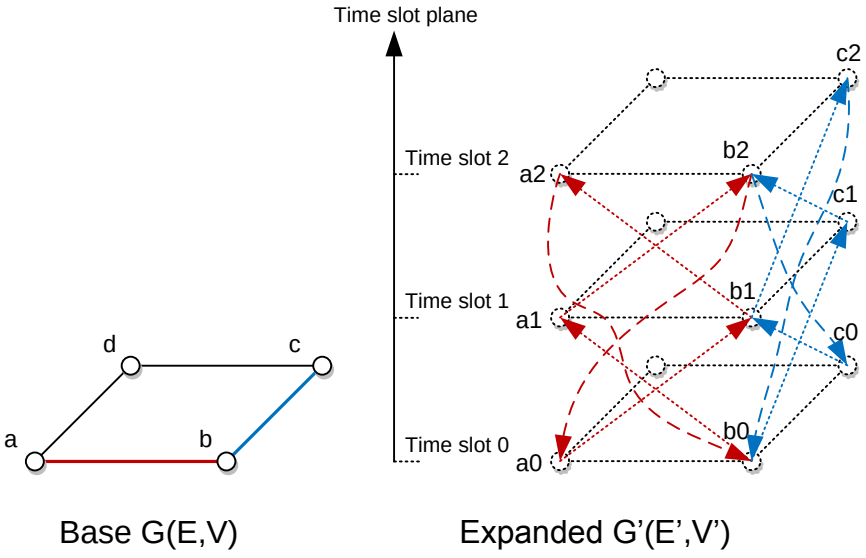

Figure 1. Example of expanded graph model

For example, in the Figure 1, the base graph $G(V,E)$ contains 4 nodes and 4 edges. The red edge $e^0(a,b)$ in the base graph $G(V,E)$ is expanded to edge set $E_0' = \{e(a_0,b_1),e(a_1,b_2),e(a_2,b_0)\}$ and $e^1(b,c)$ in the other direction is expanded to edge set E_1'=$\{e(b_0,c_1),e(b_1,c_2),e(b_2,c_0)\}$. Both the red dot line edge $e(a_2,b_0)$ and the blue dot line edge $e(b_2,c_0)$ are wrapping around edges. The left side is the base graph $G(V,E)$,

the right side is the expanded graph where the blue edges are the expansion of $e(a,b)$ and red edges are expansion of $e(b,c)$.

2.1.2 Mapping Bandwidth to NoC Resource Graph

In TDM networks, the time is divided into many time slots with the same length. The data capacity of each time slot ($Cslot$) is decided by $Cslot = Ce \times Lslot$, where Ce is the bandwidth of the edge and $Lslot$ is the length of the time slot. For example, an edge with bandwidth of $1Gbps$, and the length of time slot is 1 millisecond, then in theory, the data capacity of the time slot will be $1Gbps \times 1ms = 1Mbps$.

In realistic environment, the data capacity of each time slot is less than that in theory, because it takes time to perform clock synchronization and link negotiation. Moreover, there could be additional overhead in packets. Thus, a realistic mapping of bandwidth can be defined as:

$Cslot = Ce \times Lslot \times \beta$ where $0 < \beta < 1.0$

2.1.3 Time Window Size

Time window size is a parameter of repetitive traffic patterns. If there are two or more traffics, the window size is set to the least common multiple of all traffics. However, the time axis or time window can be extremely long and the graph size would be prohibitively large. To solve this issue, NoC traffic is often abstracted to repeated periodic patterns, which can approximately represent aperiodic cases if the period is sufficiently large.

2.1.4 Capacities of Edges in Expanded Graph

The capacities of edges in expanded graph $G'(V',E')$ are the same as the capacity of the time slot for this edge. To avoid time slot confliction in routings, the capacity of edge in expanded graph is assigned with minimal bandwidth and doesn't allow to be shared with other flits.

2.1.5 Mapping from Expanded Graph to Base Graph

Assume a set of paths $P' = \{p_0, p_1, ..., p_k\}$ are found in an expanded graph, where $p_i = \{e'(s'_{jm}, d'_{kn}) \mid s'_{jm} \in V',\ \mathrm{d}'_{kn} \in V' ,e' \in E'\}$, and the subscript j and k are the ID of nodes in base graph, the subscript m and n are the ID of *time plane* in the expanded graph. Mapping P' back to $Q = \{q_0, q_1, ..., q_k\}$ where $q_i = \{e(s_j, d_k, sid) \mid s_j \in V,\ \mathrm{d}_k \in V, e \in E, 0 \le sid < Cslot\}$ is calculated as bellow:

For any $e'(s'_{jm}, d'_{jn}) \in p_i$, then the edge in base graph $e(s_j, d_k) \in q_i$, and time slot ID $sid = m$.

2.1.6 Routing on Expanded Graph

Observation 1: Assume a set of paths $P = \{p_0, p_1, ..., p_k\}$ in an expanded graph satisfy the capacity constraints. Then, after mapping P back to the base graph, the mapped new traffics will satisfy the constraints of capacity in base graph and the slot assignment is confliction free.

Suppose there are two paths for two flits, p_1 and p_2, the two paths conflict in slot k of edge e_i in the base graph. Then, p_1 and p_2 share time slot k of edge e_i. This

conflicts with the routing constraint that any edge in the expanded graph does not allow to be shared with multiple flits.

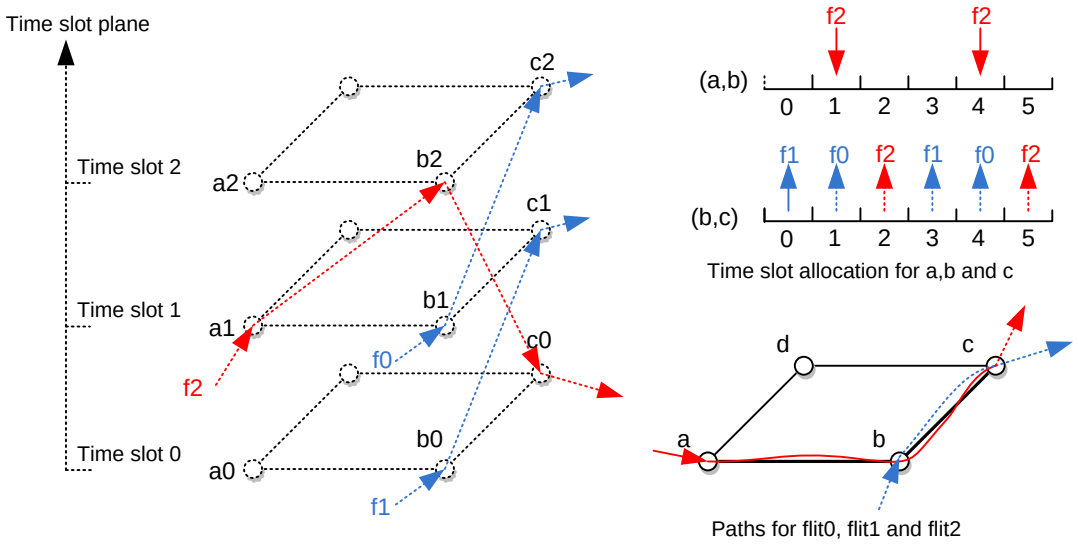

Figure 2. Example of routing and slot mapping

In Figure 2, flit0 and flit1 transmit from node b to node c, and flit2 transmits from node a to node c. After routing, path $p_0 = \{b_1, c_2\}$ is for flit0, path $p_1 = \{b_0, c_1\}$ is for flit1, the path $p_2 = \{a_1, b_2, c_0\}$ is for flit2, and the slot uses of edge $e(a,b)$ and $e(b,c)$ are shown in the right part of Figure 2.

2.1.7 Flit Injection Uncertainty

Even though this thesis research is targeted to application specific NoCs, where traffic patterns are somewhat traceable, it is still difficult to predict the exact flit injection time. This problem can be solved by adding super source nodes in the

expanded graph. For example, in the right part of Figure 3, a super source node is connected to nodes a0, a1 and a2. This is to model a time range of flit injection time. Similarly, adding a super destination node like the left part of Figure 3 provides flexibility for flit arrival deadline.

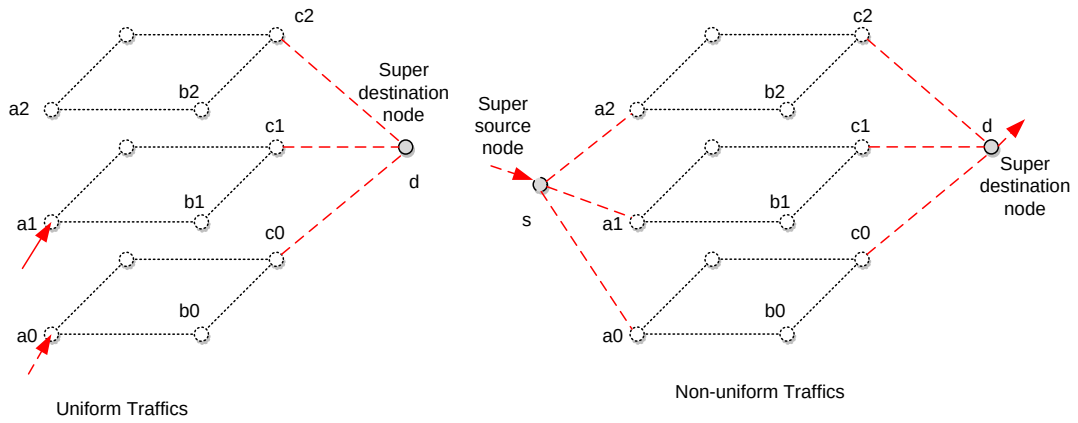

Figure 3. Injection uncertainty by adding super nodes

2.1.8 In-order Delivery

In-order delivery is very important for TDM networks. The data are chopped into packets, but the length of a packet is still too long to be efficiently transmitted in NoC. Thus, a packet is further divided into flits, which are smaller data segments with limited routing information. To better utilize network bandwidth, we allow flits of the same packet to be routed along different paths. Such multi-path routing requires that these flits arrive the destination in-order. The simplest solution for in-order delivery is to choose only the shortest paths for all flits. If all paths are shortest paths, then all flits take the

same amount of time to travel in the NoC, and therefore they reach the destination node in order.

2.1.9 Deadline Constraint

Deadline constraint is the maximal hops (Mh) allowed for a flit to transmit across NoC. If a deadline is not tight, routing detour is allowed.

We implement A* search algorithm for routing on the expanded graph. A* search computes the function $f(n) = g(n) + h(n)$ for every node n; $g(n)$ is the actual cost or hops from start node to node n; $h(n)$ is the estimated cost or hops from node n to destination node. If we initialize the graph $G'(V', E')$ with $h(n) = 0$, then A* algorithm is the same as Dijkstra's shortest path algorithm. We calculate $Dts(n)$ and $Dtd(n)$ beforehand, where $Dts(n)$ is the minimal hop count from n to source node and $Dtd(n)$ is the minimal hop count from n to destination node. Thus, $Dts(n) + Dtd(n)$ is the minimal hop from source node to destination node if the flits pass node n.

To calculate $Dts(n)$ and $Dtd(n)$, we firstly calculate $g(n)$ for every node n, by setting $h(n) = 0$. After routing across the entire graph, we obtain $Dts(n) = g(n)$ for every node n. Then, we reverse the direction of all edges in the expanded graph by reversing $e(s_i, d_j)$ to $e(d_j, s_i)$, and set $h(n) = 0$. After routing across the entire graph by A* search, we assign $Dtd(n) = g(n)$ for every node n.

2.1.10 Enforcing Deadline Constraint

We add an additional constraint to A* search, that we relax node x, if and only if $Dts(x)+Dtd(x)<Mh$.

Observation 2: Let Mh be the maximal number of hops allowed for a specific flit, if A* only select nodes with $Dts(x)+Dtd(x)\le Mh$ then the new A* algorithm can find paths whose hops are no greater than Mh, and if this path exist, the new A* algorithm at least can find one.

To satisfy the in-order delivery constraints together with the constraint of deadline is a difficult task. If there is only one flit in a time window, we do not need to worry the in-order delivery constraint. If there are multiple flits, we relax all edges using A* search, and then record the total number of hops for each path, and then choose paths which satisfy in-order delivery constraint.

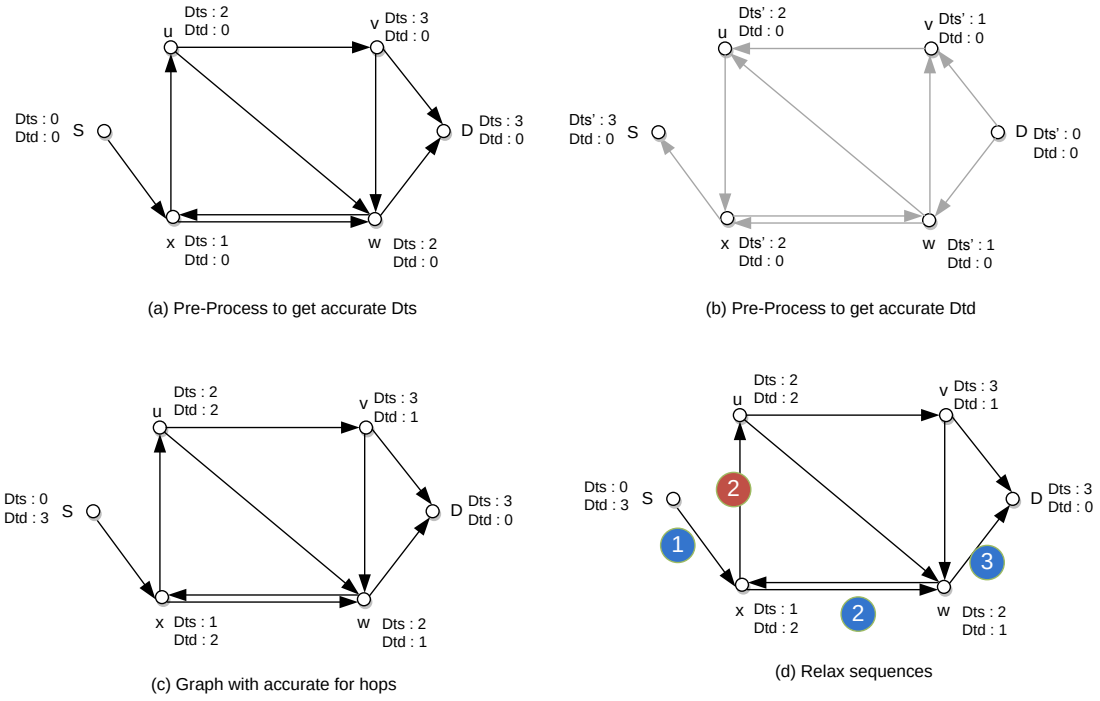

Figure 4. Relaxing steps with deadline constraints

Figure 4 is an Example for A* search algorithm with deadline constraint. Figure 4(a) is a graph with 6 nodes, and a flit from S to D. The deadline constraint is 3 hops. First we run A* search in (a) to obtain *Dts* for each node. Next, we construct the reversed graph and run A* search in (b) to assign *Dtd* to each node. Figure 4 (c) is the graph with both *Dts* and *Dtd*. Figure 4(d) shows the relaxation steps of A* search.

In step (1), edge $e(S,x)$ is relaxed. In step (2), Dts+Dtd of node u exceeds the max hot count constraint (deadline constraint) and therefore node u is not relaxed. But, node w is relaxed in step (2). In the step (3), the destination node D is reached. Then, the path $p\{S,x,w,D\}$ is correctly found and runs faster than Dijkstra's algorithm by skipping some nodes, such as u and v.

2.1.11 RRR-based Algorithm

First we briefly review rip-up-and-reroute (RRR) [5], a very popular approach of wire routing. Wire routing for circuit layout has been studied for decades. It is similar to the NoC resource allocation problem because both of them need to allocate limited resources on a graph.

Then, it assigns cost for every edge, and then starts routing for all flits one after another and ignores the capacity constraints. Therefore the paths after the initial routing may have many overflows. If *Ce* is the capacity of an edge e and *Ue* is the utilization of the edge, we define the overflow is as $\max(Ue-Ce,0)$. The overflowed edges violate the capacity constraint, so some flits need to be re-routed to less congested regions. Typically, a shortest path algorithm, such A* search, is employed for the rerouting while the congestion is captured by edge costs.

The key of RRR routing algorithm is to decide which paths need to be re-routed and how to change edge cost to make the routing in the next iteration avoid congested regions. An efficient technique to avoid congested regions is to increase an edge cost at each iteration as long as this edge continuously has overflow. Edge cost ∂e depends on both edge's length be and its congestion penalty pe, and we define the cost as:

$$\partial e = be + he \times pe \tag{2.1}$$

where he is the history factor. If i is the index of iterations, he is updated iteration by iteration as:

$$h_e^{i+1} = \begin{cases} h_e^i + h_{inc} & \text{if } Ue > Ce \\ h_e^i & \text{otherwise} \end{cases} \tag{2.2}$$

For a constant δ, pe is the congestion penalty term defined as:

$$pe = \begin{cases} \exp(\delta \times (Ue/Ce - 1)) & \text{if } Ue > Ce \\ pe & \text{otherwise} \end{cases} \tag{2.3}$$

By increasing the cost of edges which have overflow, A* search algorithm avoids choosing overflowed edges in the next iteration. RRR terminates when there is no overflowed edge or a given limit on the number of iterations is reached.

Before introducing the RRR-Based routing algorithm, we discuss an important concept: path flexibility. Path flexibility is the number of distinctive shortest paths from its source to its destination. When several flits are re-routed, a conventional method often re-routes them in an arbitrary order. We observe that flits with greater flexibility should be re-routed later than flits with less flexibility. In our RRR-based routing algorithm, we first sort the flits and re-route them in non-decreasing order of flexibility.

Another concept we used is the dead-loop. RRR may keep re-routing forever even there are feasible solutions, and we call this kind of loop as dead-loop. The dead-loop can happen in conventional RRR algorithm. We design a technique called dead-loop detector, and after so many experiments, we found that it reduces the number of iterations before the algorithm finds a feasible solution. The dead-loop detector attempts to route the flits that need to be re-routed based on the residual graph. For example, the capacity of edge e is Ce, the utilization of this edge is ue, then, in residual graph, the capacity of this edge is $Ce-ue$, and the utilization of this edge is 0. If the flits to be re-routed in the residual graph cannot find feasible solution, we say a dead loop may occur. Once the risk of dead-loop is detected, we rip up more flits on edges where overflow exists. Increasing the number of flits for rerouting may reduce the chance of dead-loop.

2.1.12 The Pseudo-code of RRR-based Algorithm

Input: Resource Graph $G(E,V)$, maximal iterators k
Output: A set of flits ϕ to be rerouted with confliction free

```
1  Sort flits φ in non-decreasing order of path flexibility
2  Construct G'(V', E') and φ_r ← φ and φ_c = ∅
3  While (--k && φ_r ≠ ∅) do
4    For ∀e ∈ E with ue > ce do
5      φ_r ← {φ_i | e ∈ p_i} where pi is the path of flits ∈ φ
6      Rip up φ_r
7    End
8    For each f ∈ φ_r do
9      φ_c ← {∀f ∉ φ_r}
10   End
11   For each f ∈ φ_r do
12     Trying to Re-route f
13     If f conflict with φ_c
14       φ_x ← f ∪ {φ_i | e ∈ p_i of this flit and ue > ce}
15     End
16   End
17   φ_r ← φ_x
18   For each f ∈ φ_r do
19     Re-route f and remove f from φ_r
20   End
21 End
```

Figure 5. Pseudo-code of RRR-based algorithm

Figure 5 shows the Pseudo-code of the RRR-Based Method. The first line of code is to sort flits in a non-decreasing order of path flexibility. The second line constructs expanded graph and puts all flits into φ_r to route all flits in the first iteration. From line 3 to line 21 is the body of RRR-Based Method. From line 4 to line 7, all flits

that going through overflowed edges ($ue > ce$) are found and ripped up, and the edge costs are recalculated. The codes from line 8 to line 10 put all flits that do not involve overflow into φ_c. Though flits in φ_c are overflow-free, they may not necessarily be the optimal solution. The codes from line 11 to line 16 attempt to route all flits in the residual expanded graph and find out whether a dead loop may exist or not. The code of line 12 tries to reroute the flits. If there is any flit conflicting with flits in φ_c, the corresponding flits in *phi_c* is moved into φ_x. There is no dead loop if φ_x is empty. The code of line 17 assigns all flits in φ_x to φ_r. The code of line 19, re-routes all traffics need to be re-routed and updates the cost of affected edges.

2.2 Power-Efficient QoS for Application Specific NoC

This section discusses power-efficient NoC QoS for Multi-Processor System-on-Chip (MPSoC). An MPSoC often has multiple operation user-cases. For example, a smart phone processor may perform text editing, voice recognition or video streaming at different times. Each user-case entails a specific traffic pattern on NoC. The objective is to minimize average energy dissipation among all user-cases, including dynamic and static energy. Ideally, a flit is routed along the shortest physical path in the network without waiting in a buffer. If there is resource contention between different flits, the decisions face three options: (1) increasing link capacity, (2) waiting in a buffer, (3) routing detour. The first two options increase static energy while option (3) causes more dynamic energy. Hence, there is a tradeoff among these options and we can use the three

options to find routing and time slots for each flit such that the total power consumption is reduced.

In previous works, the IP blocks and the net links are designed before the routing and time slot allocation, yet, some links or NoC routers can be removed without degrading the QoS and some other links may need to increase their bandwidth, and some nodes need to add buffers. All decisions can be made along with routing and time slot assignment. Please note that link and buffer capacity can be shared among different user cases. Therefore, our approach includes simultaneous routing, time slot assignment and link/buffer capacity optimization.

2.2.1 Power Consumption Model

Power consumption includes two parts, static and dynamic. The dynamic power of each node is estimated as:

$$Dn = \begin{cases} dn \times fn & \text{if used by any flit} \\ 0 & \text{if unused} \end{cases} \qquad (2.4)$$

where *dn* is the dynamic power of one flit and *fn* is the number of flits.

The static power of an edge is defined as

$$\begin{aligned} Se &= Su \times Cb && \text{if } su \le M \\ Se &= \infty && \text{if } su > M \\ Se &= 0 && \text{if } su = 0 \end{aligned} \qquad (2.5)$$

where *su* is the number of physical links implementing one edge, *Cb* is the static power of a single link and *M* is the upper bound on the number of physical links. Please note the infinite power is not realistic, but is to forbid the number of links from being

greater than the upper bound. The link capacity increase is equivalent to adding additional physical links. This is illustrated in Figure 6.

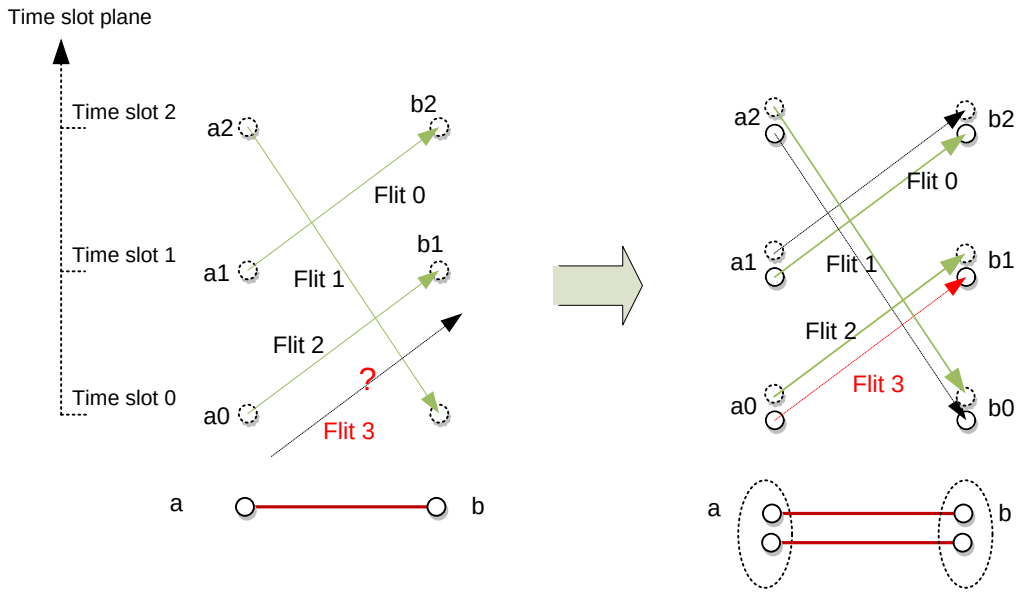

Figure 6. The method to increase the capacity of an edge

Figure 6 is an example of increasing the capacity. The edge $e(a,b)$ should carry 4 flits if time window size is 3. However, this edge can transport only 3 flits at a time. Instead of increasing the capacity of $e(a,b)$ to 4 flits in slot window, we add additional wire for $e(a,b)$, so the capacity of $e(a,b)$ increases up to 6 flits.

The dynamic power for an edge is defined as

$$De = \begin{cases} fu \times Cf & \text{if used by any flit} \\ 0 & \text{if unused} \end{cases} \qquad (2.6)$$

where fu is the number of flits passing through the edge and Cf is the dynamic power dissipation of one flit.

The static power for a buffer edge is defined as:

$$\begin{aligned} Sf &= su \times Cbs && \text{if } su \le M \\ Sf &= \infty && \text{if } su > M \\ Sf &= 0 && \text{if } su = 0 \end{aligned} \qquad (2.7)$$

where *su* is the buffer depth, *Cbs* is the static of buffer for one flit and *M* is the upper bound for buffer depth.

The dynamic power for a buffer edge is defined as

$$De = \begin{cases} fu \times Cbe & \text{if used by any flit} \\ 0 & \text{if unused} \end{cases} \qquad (2.8)$$

where *fu* is the number of flits entering (or exiting) the buffer and *Cbe* is the dynamic power dissipation of a single flit.

2.2.2 Graph Model for Power-Efficient QoS

The problem of Power-efficient QoS for Application Specific Networks-on-Chip can be described upon a directed base graph $G(V, E)$ too, where V is a set of nodes modeling routers and E is a set of edges indicating communication links. To allocate time slot and route simultaneously, the base graph $G(V, E)$ is expanded into $G'(V', E')$, called expanded graph, which contains time slot information. This graph model employs the same concepts of slot window size, *time plane*, and graph expansion as the resource

graph described in Section 2.1.1. However, graph model here considers link and buffer capacity change.

Suppose time window size is tw, buffer edges can be constructed as follows. For every node $v \in V$, there is a set of nodes $V' = \{v_0, v_1, ..., v_{tw-1}\}$ in expanded. Then the buffer edges are $Be = \{e_0(v_0, v_1), e_1(v_1, v_2), ..., e_{tw-1}(v_{tw-1}, v_0)\}$.

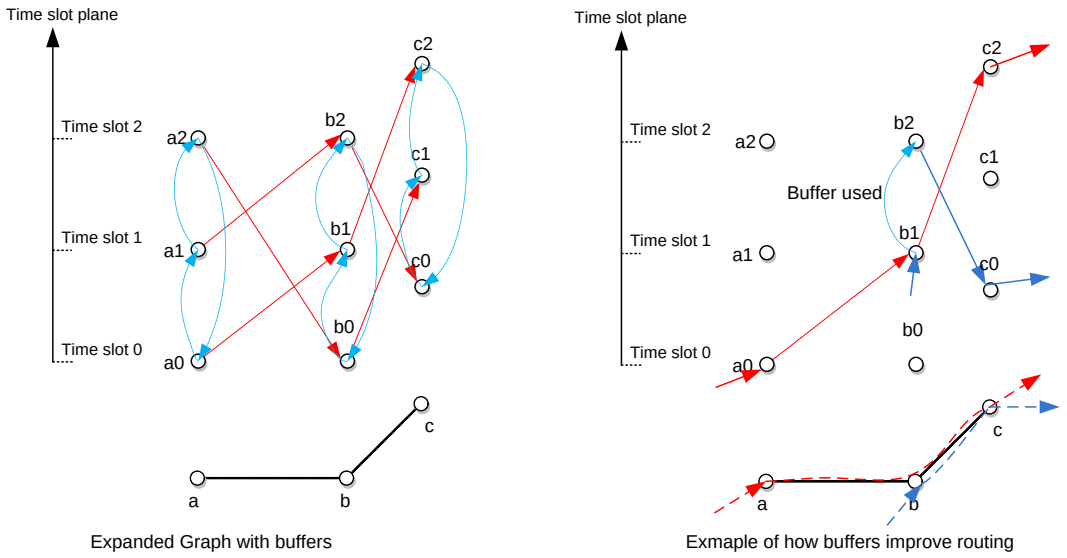

Figure 7. Constructing power-efficient expanded graph

Figure 7 shows one example of buffer edges. Since a buffer holds flits at the same router instead of transmitting the flits to other routers, a buffer edge is always incident to nodes corresponding to the same router. The right side of Figure 7 shows how buffer can improve the utilization of links. There are two flits, one traveling from a to c, injected at time plane 0, and the other one going from b to c, inserted at time plane

1. When we route the second flit, the time slot 1 of node b is occupied by the first flit. In this case, the second flit can be stored at router b for one time slot, and reach node c along path $\{b_1, b_2, c_0\}$. Although the second flit needs to wait, the link capacity from b to c does not need to be increased.

2.2.3 Cost Functions for A* Search Algorithm

The negotiation-based heuristic is based on RRR algorithm. The cost function and how to find non-preferred edge are two key concepts of our negotiation-based heuristic. We use A* search algorithm to route in the expanded graph. The cost for A* search is the incremental power consumption for nodes or edges.

The routing is carried out sequentially, one flit after another. Before routing a flit, Negotiation-Based Heuristic pretends to add one flit for every edge and node, calculate the incremental power consumption as the cost for edge or node. Then A* search is applied to find a minimal power consumption path. Although the negotiation-based method finds the minimal power path for every flit, it is still greedy for the overall problem and cannot guarantee the overall optimality.

Cost ∂e for edges or nodes are calculated as bellow:

$$\partial e = \Delta p_s + (\lg(he) + 1) \times \Delta p_d \qquad (2.9)$$

where Δp_s is the increment of the static power if a flit is added in the next iteration. It is calculated as bellow:

$\Delta p_s = p_s^{i+1} - p_s^{i}$, where i is the index for iterations.

Δp_d is the increment of the dynamic power if a flit is added in the next iteration and can be calculated as bellow:

$\Delta p_d = p_d^{i+1} - p_d^{i}$, where i is the index for iterations.

The history penalty term is increased when this edge is found to be non-preferred.

$$h_e^{i+1} = \begin{cases} h_e^i + h_{inc} & \text{if edge e is non-preferred} \\ h_e^i & \text{otherwise} \end{cases}$$, where i is the index for the iterations.

2.2.4 Non-preferred Edges

Non-preferred edges are those that may cause flits to stuck at local optimal routes. Next, we will elaborate what they are and show how avoiding can help to improve solution quality.

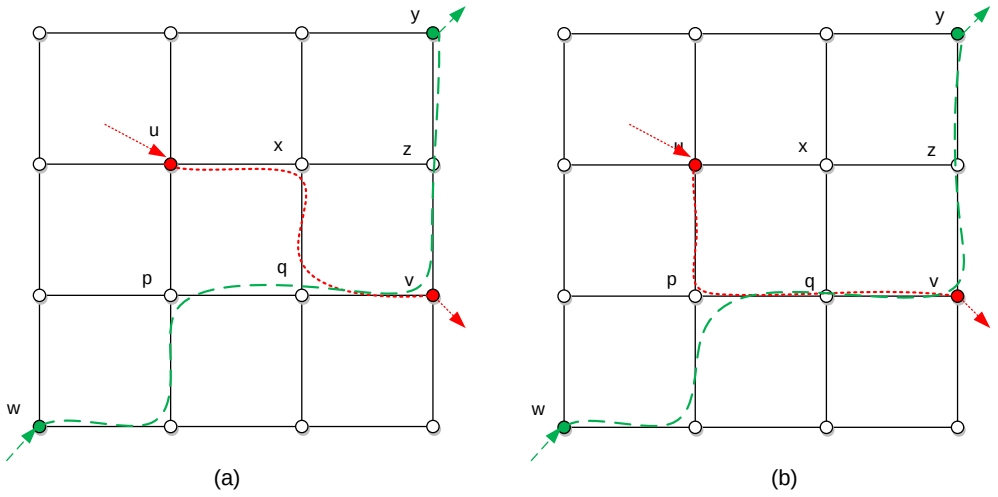

Figure 8. Example of a non-preferred edge

We describe by an example in Figure 8. Negotiation-based heuristic routes one flit at a time. In this example, the red flit with less path diversity is routed first. It can find 3 shortest paths, and without loss of generality, we assume the algorithm chooses path $\{u, x, q, v\}$. Then the algorithm attempts to route the green flit from w to y. Because the static power consumption of edges $E_0\{e(u,x), e(x,q), e(q,v)\}$ will not be increased if one flit is newly added, the routing of the second flit tries to reuse E_0. Then it turns out that (a) in Figure 8 is one of the optimized paths. However, the solution of (b) in Figure 8 is better; because the red traffic and the green traffic have two edges shared and can save more energy. In this case, *(x,q)* is a non-preferred edge. If we increase the cost of this edge, the A* search would select path $\{u, p, q, v\}$. The key characteristic of edge (x,q) is that the direction of the red flit along it is the opposite of the direction the green flit. To be more specific, the direction from node x to node q is not an edge of any edge in the shortest path from w to y.

We define non-preferred edges as the edges that are within other flits' shortest paths but have direction opposite to the direction of the shortest paths. If any non-preferred edges are detected, the cost of this edge increased and the flits are re-routed in the next iteration.

2.2.5 The Pseudo-code of Negotiation-based Heuristic

```
Input:  Resource Graph G(E,V), GS flits φ, iteration count k
Output: Power-efficient routing for flits φ
 1  Calculate Dts and Dtd for every flits φ
 2  Sort flits φ in non-decreasing order of path flexibility
 3  Φ ← ∀f ∈ φ
 4  While (--k && Φ ≠ ∅) do
 5      For every f ∈ Φ do
 6          Calculate costs of edges and nodes for adding one flit
 7          Route f using A* Algorithm and remove f from Φ
 8      End
 9      For each non-preferred edge e ∈ {p_i ∈ f} do
10          Increase history factor for e
11          Put any f ∈ { f_i | e ∈ p_i of f } to Φ
12          Rip up f ∈ Φ from G(E,V)
13      End
14      Count power consumption
15      Save the solution with minimal power consumption
16 End
```

Figure 9. Pseudo-code of Negotiation-based Heuristic

In line 1, the algorithm calculates Dts and Dtd for all flits, which are useful to achieve in-order delivery. Line 5 to line 8 routes all flits to be re-routed. From line 9 to line 13, we check all the non-preferred edges to find which flits need to be re-routed in the next iteration. Line 14 is to calculate the overall power consumption for the current iteration. Line 15 saves the best solution.

This heuristic does not need to handle the dead loop issue, because one can always finds a feasible solution by increasing link/buffer capacity. If the capacity of some edges has to be enlarged more than the system allowed, we simply designate the corresponding solution with infinite power consumption. Another issue we need to point out is that the iteration k is a flexible parameter. Because there is no sufficient information for us to tell in which iteration we can find the best result, we run the code for a fixed number of iterations.

2.3 Hybrid Architecture for NoC

It is well known that communication links occupy a large amount of space in chip. Most NoC techniques are borrowed from computer interconnection network, which share ideas with internet technology. In a NoC, GS traffic and BE traffic coexist. Thus, typical NoCs designs should take both of them into consideration. GS traffics imply service guarantee and top service priority. BE traffics can be categorized to different priority levels. Our observation is that GS can be easily achieved with circuit-switching while BE traffics fit better in packet switching. We consider to integrate these two in a seamless manner. The circuit switching can be implemented in a way similar as FPGA reconfiguration. Therefore, we design NoCs as a network with two different types of routers – one is GS router where switches that are configured by SRAM like in FPGA, the other is BE router like in conventional NoC designs.

2.3.1 High Level Design of Hybrid NoC

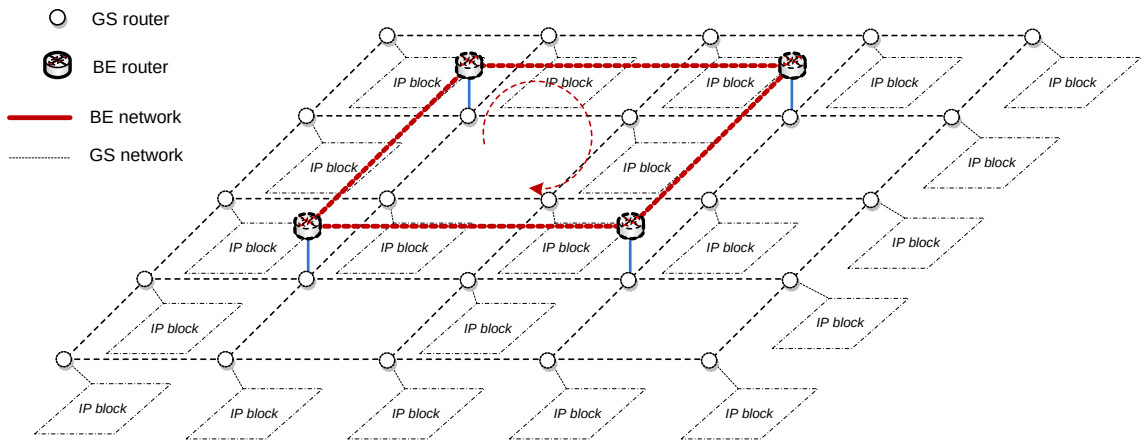

Figure 10. Hybrid structure of NoC

Figure 10 is an example of the proposed hybrid NoC architecture. The GS routers connect with IP blocks, other GS routers and BE routers. The routers and links at the bottom layer form the GS network, which operates according to TDM. The routers and links in the up plane form the BE network, which is a packet switching network. Every BE router is assigned to 5 or 6 IP blocks, which form a local network.

2.3.2 GS Network

GS routers form the circuit-switching network, and connect both the IP blocks and the GS routers. From the point view of GS routers, BE routers and IP blocks are equivalent. For the sake of convenience, we call both of them Data Handle Point (DHP).

GS routers can be designed as 4-ports routers, 6-ports routers, or even more ports routers. The structure is depicted in Figure 11.

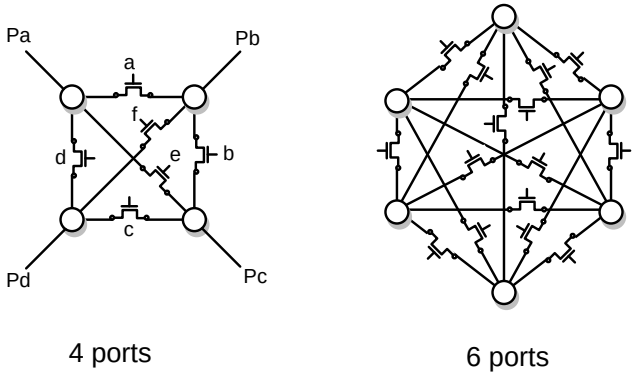

Figure 11. Design of 4 ports and 6 ports GS router

GS network is purely TDM network where time is divided into equal time slots. At the beginning of each time slot, the GS routers in the path from source DHP to destination DHP are all re-configured by local SRAM such that the physical connections form a circuit. Then, source DHP links destination DHP directly, and transmit data point to point in the rest of the time slot.

This approach has several advantages. First, such circuit switching allows very low latency each hop as the virtual channel allocation and switch allocation are skipped. By avoiding these allocation logics, the power dissipation of each GS router is also very low. If the time window size is *k* and a GS router has 4 ports, we need only 1 byte to configure the pass transistors for each time slot and *k* bytes for *k* time slots. Hence, the

area, power and complexity of a GS router are much lower than a conventional BE router.

2.3.3 Graph Model

The resource allocation problem for Hybrid NoC design can be modeled in a directed graph $G(V,E)$, where V is a set of nodes modeling routers and E is a set of edges indicating net links. The base graph $G(V,E)$ is expanded into expanded graph $G'(V',E')$. The definitions of *time plane* and time window size are the same as previous sections.

The edges in base $G(V,E)$ are expanded along time axis by duplicating and connecting the vertices. For any $e^i(si,di) \in E(e^0,e^1,...,e^i,...)$ where si is the source vertex of e^i and di is the target of e^i. The edges based on e^i are $E_i' = \{e^0(s_0,d_0),e^1(s_1,d_1),...,e^{tw-1}(s_{tw-1},d_{tw-1})\}$. The edges in expanded graph are defined as $E' = \{E_0',E_1',...,E_{tw-1}'\}$.

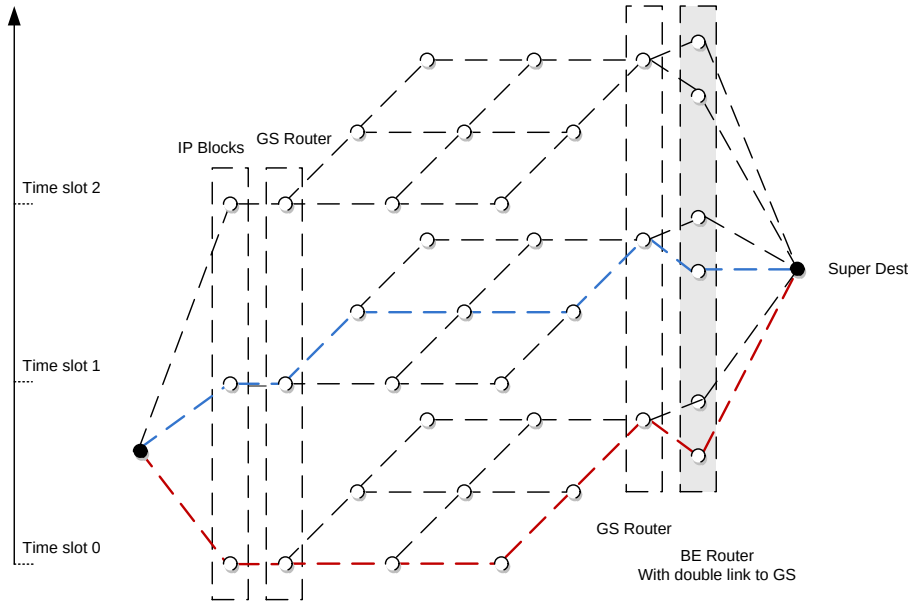

Figure 12. Constructing expanded graph for Hybrid NoC design

Figure 12 is an example on how to construct expanded graph for Hybrid NoC architecture. Super-source and super-destination are created, and the edges of expanded graph only link each other in the same *time plane*.

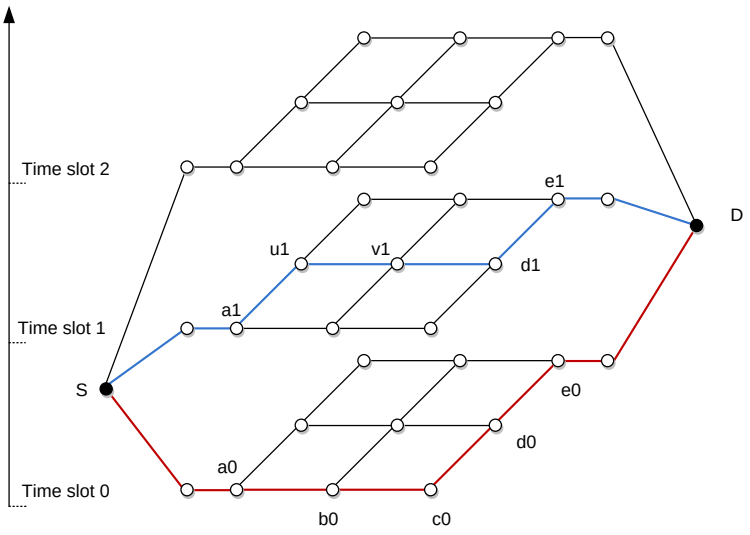

Figure 13. Routing in the graph model of Hybrid NoC architecture

Figure 13 is an example of routing in expanded graph for the hybrid NoC architecture. In this example, the time window size is 3, and there are two flits from node a to node e. In time slot 0, GS routers link $\{a_0, b_0, c_0, d_0, e_0\}$ together, and make node a communicate with node e directly. In the time slot 1, GS routers link $\{a_1, u_1, v_1, d_1, e_1\}$ together, and make a communicate with e directly. If the IP block sends flits strictly in time slot $\{0,1,3,4,...\}$ the latency of flits can be at most one clock cycle. If the IP block generates flits randomly, in the worst case, every flit will be delayed for 1 time slot.

2.3.4 BE Network

BE traffics are transmitted in either GS network or BE network. In GS network, BE traffics are delivered between IP block and BE router using GS router. If IP block is designed to enable BE traffics, the block reserves one or more time slot for BE traffic, and routes them to its nearest BE router. In BE network, BE traffics are routed and

delivered by BE routers. BE routers only connect BE routers and GS routers. BE routers are designed with a TDM interface to receive from or send to GS routers and store data in buffer. BE routers do not handle GS traffics, thus, the NoC system can decrease the number of BE routers. A ring or double ring topology can be adopted in a hybrid NoC architecture according to the size of NoC.

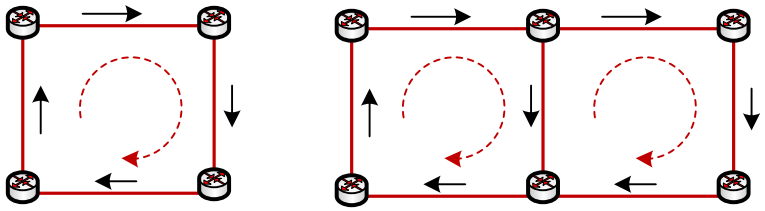

Figure 14. Topology of BE routers in Hybrid Design

Figure 14 is the example of single ring and double ring structure for the BE network.

2.3.5 Time Slot Allocation and Routing

The GS network performs the simultaneously routing and time slot allocation in the expanded graph, using the RRR-based algorithm.

CHAPTER III

EXPERIMENT SETUP AND RESULT

All the algorithms are implemented in C/C++ and the experiments are performed on AMD Opteron processor with 2.2GHz frequency and Linux operating system.

3.1 Simultaneously Resource Allocation and Routing

We compare RRR-based method with the method of Z. Lu's work [3].

3.1.1 Experiment of Success Rate

Table 1. Experiment result of success rate for resource allocating and routing

Cases	No. Packets	Previous Work[3]			RRR-Based		
		Success	Runtime(s)	TimeOut	Success	Runtime(s)	TimeOut
Mesh 6*6	25-90	20.0%	11	64.0%	60.0%	53	40.0%
Mesh 8*8	30-160	23.1%	1	73.1%	65.4%	268	34.6%
Mesh 10*10	35-250	18.2%	4	72.7%	59.1%	1162	40.9%
Random 36	25-110	20.8%	1	79.2%	66.7%	42	33.3%
Random 64	30-270	17.4%	1	82.6%	73.9%	324	26.1%
Random 100	35-450	16.7%	1	83.3%	79.2%	1228	20.8%
Average		19.4%		75.8%	67.4%	513	32.6%

The leftmost column of Table 1 is the size of the test cases. The mesh 6×6 is 2D mesh topology with $6\times6=36$ nodes, and similarly for mesh 8×8 and mesh 10×10. The Random N where $N=\{36,64,100\}$ is the random topologies with N nodes. The second column tells the number of GS packets in each time window. In each case, about 85% packets are single-flit and 15% are multi-flits.

In the experiment, we set a timeout limit of 4 hours. If a method running out of time, the run is counted as a failure. The previous work of [3] has only average 19.3% success rate, and RRR-based is average 67.3%.

3.1.2 Stress Test

The stress test is to increase the packet injection to a certain network design till the point where feasible solution of routing and time slot assignment cannot be found. Then, the maximum number of packets injected to this point is an indication of the capability of the algorithm. The results are shown in Table 2.

Table 2. Experiment result of stress test

case	Previous Work[3]	RRR-Based
	Maximal No. of traffics	Maximal No. of traffics
1	40	101
2	40	81
3	61	100
4	43	92
5	51	93
6	31	67
7	33	140
8	37	80
9	37	100
10	11	29
Average	38.4	88.3

In the Table 2, there are 10 test cases. From the table we can tell that the maximal numbers of GS traffics that can be accommodated by previous work [3] are only half of the numbers of our RRR-based algorithm.

3.1.3 Experiment for Conventional RRR and Our RRR

The inputs of this comparison test are generated randomly to show whether our new techniques on RRR-based algorithm work. The main differences between the two methods are, (1) Conventional RRR does not have dead-loop detection and (2) Our routing follows non-decreasing order of path diversity. We compare the two methods on 22 cases and the results are shown in Table 3.

Table 3. Comparison of conventional RRR and our RRR

	Conventional RRR	New RRR
No. of Success	18%	91%
Average No. of Iterations	15	28
Average Runtime	47s	996s

The RRR-based proposed by this thesis solves 91% test cases whe the conventional RRR solves only 18%. The average runtime of our RRR-based method is much longer than that of the conventional RRR method. There are two reasons. One is that the average runtime does not consider the unsuccessful test cases, which take longer than 4 hours, and the other reason is that the conventional RRR is simpler than our RRR.

3.2 Experiment for Power-Efficient QoS

To the best of our knowledge, there is no previous work dedicated to power-efficient QoS for application specific NoCs. Hence, we compare with extensions of one related but different work. The Iterative Greedy is extended from the related work [6], which is an heuristic for minimizing link capacities only. It iteratively routes a flow

along the minimal cost path in the network. If a link's capacity has already been used in previous iterations, its cost is regarded as zero to encourage reuse.

There are two types of test cases. One is random benchmarks generated by TGFF [11] which has been used in many other NoC works.

3.2.1 Experiment for Large TGFF Cases

The experiment is on a set of relatively large TGFF cases. The second column is the total number of nodes $|v|$ for the expanded graph. So $|v|=|V|\times tw\times|p|$, where $|V|$ is the number of physical nodes in base graph $G(V,E)$, and $|p|$ is the number of user cases and tw is the window size. In this situation, the Negotiation-based heuristic demonstrates its value on large cases. In the result of Table 4, negotiation-based heuristic obtains 14% energy reduction compared to the iterative greedy heuristic. The runtime is increased but still at a manageable level.

Table 4. Experiment and result of large TGFF cases

TestCase	Total |v|	Iterative Greedy		Negotiation-Based	
		Energy	RunTime(s)	Energy	RunTime(s)
Case 1	9000	844	3347	718	7239
Case 2	9900	1694	9126	1358	17461
Case 3	10010	1336	6018	1178	12712
Case 4	10080	1541	7556	1289	14744
Case 5	10200	1275	5680	1135	12337
Case 6	10200	1331	6990	1117	13725
Case 7	10260	1458	7487	1217	14961
Case 8	10400	1216	6095	1088	13215
Case 9	10500	798	3292	697	7097
Case 10	10500	1269	6173	1090	12253
Case 11	10500	1050	5618	966	11090
Normalized Total		1	1	0.86	2.03

3.3 Experiments for Hybrid NoC Architecture

3.3.1 Stress Experiment for Success Rate

In stress comparison experiment, we build 10 test cases, each case with 118 traffics in the base graph of mesh 6*6 and time window size of 8. Then, we compare the maximal number of traffics can be successfully routed between TDM-based NoC and the hybrid NoC architecture. In the result, we count the total number of hops. In the hybrid architecture, each GS packet takes only one hop as all flits are transmitted in one time slot.

Table 5. Comparison between TDM NoC and Hybrid NoC

cases	Maximal # of traffics		Average No. hops	
	RRR-Hybrid	RRR-TDM	RRR-Hybrid	RRR-TDM
1	110	112	89	435
2	103	81	88	437
3	109	108	85	414
4	94	94	91	448
5	103	103	94	435
6	100	101	91	417
7	110	111	86	414
8	110	110	93	431
9	107	106	92	451
10	93	93	94	428
Average	103.9	101.9	90.3	431

In the result of Table 5, the solution space for TDM-based NoC design is similar as the hybrid NoC design. The average of the maximal number traffics can be deployed in TDM-based is 101.9 and that in Hybrid-based design is 103.9. The total number of hops needed for the hybrid design is only 20.9% of that for TDM-based NoC design.

This means that the hybrid design significantly decreases traffic latency without decreasing the capability of routing.

CHAPTER IV

CONCLUSIONS

The RRR-based algorithm for simultaneous time slot assignment and routing increases the success rate compared to previous approaches. The negotiation-based algorithm saves more energy for large TGFF cases in an acceptable runtime. As the size of NoC increases, the negotiation-based algorithm shows greater advantages. The hybrid NoC architecture successfully combines the circuit switching architecture with packet switching architecture, decreases the latency for traffics, simplifies the design of the architecture decreasing the number of Best-Effort routers, and reduces power consumption by the simple GS router design.

REFERENCES

[1] M.D. Harmanci, N.P. Escudero, Y. Leblebici, and P. Ienne, "Quantitative modelling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip ", in proceedings of IEEE International Symposium on Circuits and Systems, IEEE, pp.1782-1785 vol.2, 2005.

[2] A. Hansson, K. Goossens, and A. Rãdulescu, "A unified approach to constrained mapping and routing on network-on-chip architectures", in proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM, pp. 75-80, 2005.

[3] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network on-chip", Very Large Scale Integration (VLSI) Systems, IEEE 16(8):1021–1034, Aug. 2008.

[4] R. Stefan, K. Goossens, "A TDM slot allocation flow based on multipath routing in NoCs", Microprocessors and Microsystems 35(2): 130-138, 2011.

[5] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale", Computer-Aided Design of Integrated Circuits and Systems, IEEE 27(6):1066–1077, June 2008.

[6] I. Walter, E. Kantor, I. Cidon, S. Kutten, "Capacity optimized NoC for multi-mode SoC", in proceedings of the 48th Design Automation Conference, ACM, pp.942-947, 2011.

[7] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints", in proceedings of Design,

Automation and Test in Europe Conference and Exhibition, IEEE, pp.234-239 vol.1, 2004.

[8] W.J. Dally, and B.P. Towles, "Principles and practices of interconnection networks", Elsevier, 2004.

[9] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage, "Networks on silicon: combining best-effort and guaranteed services", in proceedings of Design Automation and Test in Europe Conference and Exhibition, IEEE, pp.423-425, 2002.

[10] ATM Forum Technical Committee, "ATM User-Network Interface Specification", The ATM Forum, 1994.

[11] D. Rhodes, R. Dick, and K. Vallerio, "Task graph for free", http://ziyang.eecs.umich.edu/ dickrp/tg_/.