A Method for Efficient NoC Test Scheduling Using Deterministic Routing

Rana Farah
Department of Computer Engineering
École Polytechnique de Montréal
Montréal, Québec, H3C 3A7

Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010, Lebanon

***Abstract*—Network-on-Chip (NoC) is an on-chip communication methodology that has been proposed as an alternative to bus-based communication in order to cope with the increased complexity in embedded designs. This paper presents a method for NoCs test scheduling using simulated annealing. The method uses a deterministic routing algorithm that minimizes test time while avoiding blocking. The method is implemented and various benchmarks are attempted.**

I. Introduction

The increase in design complexity as well as the continuos need for reducing heat and power consumption has made multi-core architectures a cost effective alternative. It has been argued by many researchers that there is a need to meet the high communication requirements of large multi-core architectures using a scalable communication methodology [1]. *Network-on-Chip* (NoC) designs are emerging technologies that consist of a number of interconnected heterogenous devices that communicate over a scalable interconnection network by bringing packet-based communication paradigms on-chip. NoCs are organized using various topologies such as *mesh/torus*, *fat-tree* and *ring*. A 2D-mesh topology consists of nodes that are arranged in a rectilinear grid where each node is bi-directionally connected to its *top*, *bottom*, *left*, and *right* neighbors [2]. Each node includes a multi-port switch that is associated with a core. NoC provides several advantages including modularity, higher performance, better structure, and compatibility with core designs and reuse [3]. The design of an efficient NoC presents various challenges that include synthesizing the network topology, modeling the traffic characteristics, and the selection of a routing algorithm. Another concern in NoC design is test time minimization which remains a challenge.

NoC routing algorithms maybe *deterministic* or *adaptive*. Deterministic routing requires less resources since the routing path is chosen before the packets are sent. However, it does not utilize the network to its full potential when there is a heavy load. Various static routing techniques have been proposed such as the *xy* routing where a packet is routed first in the *x* direction and then along the perpendicular *y* dimension. *xy* routing can be applied to regular two dimensional mesh topologies and produces deadlock-free NoC minimal paths but at the expense of available bandwidth. In addition, *xy* routing does not work with irregular meshes since some links may lead to dead-ends [4]. *Adaptive* routing techniques are used in order to alleviate the latency associated with blocking. NoC testing involves the reuse of the existing on-chip communication as a test access mechanism where test stimuli and responses are transmitted through the network. Typically, the original wrapper design is kept unchanged while the cores' functional inputs and outputs as well as the internal scan chains are concatenated into wrapper scan chains of similar length such that the channel width is enough to transport one bit for each wrapper scan chain [5].

A. Related Work

The NoC mapping, routing, and testing problems have been tackled by various researchers. For example, Murali *et al.* [3] addressed the NoC mapping problem that supports traffic splitting under bandwidth constraint with the aim of minimizing communication delay. Bolotin *et al.* [4] extended the *xy* routing with hard coded paths for deadlock free communication in an irregular mesh topology. Hu *et al.* [6] presented a non-minimal deadlock free routing algorithm for an irregular NoC mesh topology with regions. Stochastic routing for fault-tolerance in NoCs has been discussed in [7], [8]. Cota *et al.* [9], [10] proposed the reuse of the on-chip network in order to reduce hardware resources while increasing the availability of several parallel paths for transmitting test data to each core. Liu *et al.* [11] proposed a method for test time reduction. Hosseinabady et al. [12] proposed a method that reuses on-chip networks for testing NoC switches. Cota *et al.* [13] proposed a method for test access and test scheduling in NoC-based system by progressively reusing the network resources in order to transport test data to routers.

B. Problem Description

This paper presents a method for test scheduling NoCs using simulated annealing. The method uses dynamic routing path allocation during testing such that blocking is minimized. The problem has been shown to be NP-complete [11].

The remainder of the paper is organized as follows. Section II introduces the *NoC test scheduling problem*. Section III formulates *the annealing test scheduling problem* including the configuration representation, the neighborhood functions, and the cost function. The *annealing algorithm* is described in section IV. We conclude with *results* in section V.

978-1-4244-6683-2/10/$26.00 ©2010 IEEE

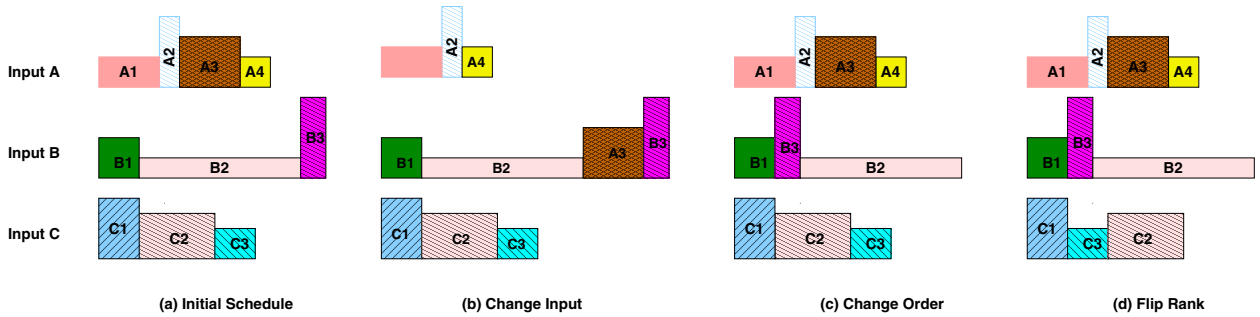

Fig. 1. p34392 SOC benchmark system implemented using a 2-D grid [14]

II. NoC Test Scheduling

We represent NoCs using a 2D-mesh topology where each cell includes a core along with a multi-port switch. The location of the cores on the grid are determined using an NoC mapping algorithm [14]. We do allow irregular meshes but in this case, the cell will only have a multi-port switch. The NoC is modeled using a *core communication graph (CCG)*, $G = (V, E)$, which is a directed graph where each vertex $v_i \in V$ represents a core, and each directed edge $e_{i,j} \in E$ represents the communication between cores v_i and v_j. The weight of edge $e_{i,j}$ represents the channel bandwidth and is treated as a flow of single commodities.

Each core has two input ports and two output ports by reusing the core's input/output or wrapper infrastructure, a set of scan chains, and a predefined set of test patterns. The number of allocated bits for each core is limited by the network capacity, 32 bits in our case. Typically, the number of pins and scan chains exceeds the number of bits allocated to a specific core. Wrappers are used to achieve serialization, and result in the use of Pareto-optimal points in test time calculation. Thus, a channel can be used by several cores simultaneously since cores cannot use the complete channel width. Each input port has a 4-bit flit buffer. Test packets are defined such that each flit arriving from the network can be unpacked in once cycle. Thus, each bit of a packet flit fills exactly one bit of a scan chain of the core. Test packets are routed to the core under test, and test responses are assembled into packets and routed back to the tester. The core test time includes the 1) time required to channel the flits from the designated input of the NOC to the core, 2) the core test time, and 3) the time needed to channel the flits from the core to the selected output of the NoC. The test patterns length is calculated as the total sum of the scan chains length, and the number of output and bidirectional pins. Each test pattern is considered as a packet. The packets are split into flits that can be accommodated by the network. A flit size is determined as the nearest Pareto-optimal point to the width of the channel bandwidth (32 bits). In this case a channel could be used by several cores simultaneously. For example, the total length of the scan chains in core 2 in *p34392* benchmark is 8,856 with 263 outputs. The length of one test pattern would be: $8856 + 263 = 9119$. In this case, core 2 has 514 test patterns and its Pareto-optimal point is 15; thus, the total number of flits is $\simeq 312,478$.

III. Test Scheduling Using Simulated Annealing

The key elements in implementing the annealing algorithm are: 1) the configuration representation, 2) the definition of a neighborhood on the configuration space and perturbation operators, 3) a cost function, and 4) a cooling schedule.

A. Configuration Representation

We propose an annealing configuration that is modeled using a two-dimensional vector where a *core* and a *switch* form a location (x,y) in the *2-D* mesh. Each core is connected to the communication network using a switch that is connected to adjacent switches.

B. Initial configuration

Cores are initially *sorted* in increasing order according to the number of flits. Cores are next assigned a priority based on the number of flits; thus, cores with the largest number of flits are assigned the highest priority. The initial configuration is next selected such that the test patterns of specific cores are randomly assigned to inputs ports of the NoC as well as to the input port of the core. All flits are next assigned either serially or in parallel if the bandwidth of the channel permits it.

C. Neighborhood Transformations

We explore the solution space based on three neighborhood transformations that we present next in reference to Figure 1.

1) Change Input: The transformation selects a core randomly and changes its input to another randomly selected input. For example in Figure 1(b), core A_3 and input B are randomly selected. Core A_3 is allocated to input B. It should be noted that core A_3 is scheduled before core B_3 since core A_3 has a higher priority.

2) Change Output: The transformation selects a core randomly and changes its output to another randomly selected output.

3) Change Order: The transformation selects a core randomly and changes its rank to another randomly selected rank. For example, core B_2 in Figure reffunctions(c) is chosen and its rank is changed to 3.

4) Swap Rank: The transformation selects two cores randomly and swap their priorities. The cores maybe assigned to the same or to different input ports. For example, cores B_2 and C_3 in Figure 1(c) are randomly selected. The priority of B_2 is changed to 3 while the priority of C_3 is changed to 2.

D. Cost Function

Given N_c cores, the objective is to find the minimum test time for all the cores. The total test time of a core is based on the time it takes flits to be routed from the input port to the core *plus* the actual test time of the core *plus* the time it takes the flits to be routed from the core to the output port. Every flit is delayed for 3 cycles at each router so that the core can process the test and the wrapper is ready to pack and deliver the response packet. Thus, the test time is given by:

$$\begin{aligned} \text{Core test time} &= IN_H \times 3 + \\ &\quad \text{The Core Actual Test Time} + \\ &\quad 3 \times OU_H + \text{Number of flits} \end{aligned} \quad (1)$$

Where IN_H is the number of hops from the input port to the core corresponding to one flit and OU_H is the number of hops from the output port to the core corresponding to one flit. The number of flits, n, is computed based on the following:

$$\text{Nb. of Flits} = \frac{\text{number of test patterns} \times \text{test length}}{\text{number of bits allocated for a core}} \quad (2)$$

E. Test Packets Routing

We use a routing algorithm that routes flits across the NoC based on the network state and queues occupancy. Thus, if congestion is detected, packets will be routed in a congestion free path. All test patterns that correspond to a certain core will follow the same route. The algorithm uses a depth-first search on a tree with a maximal degree of 4, which is the number of adjacent hops a packet can move to. The algorithm, shown in Figure 3, is based on a maze methodology where a *packet* is routed through the mesh according to a set of rules and is *re-routed* only upon reaching a *congestion*, a *broken link*, or a *missing tile*. The algorithm implies a local knowledge of the network relative to the packet's position, and requires the selection of one path upon encountering an intersection of routes in the network while "remembering" the intersection. If the selected path leads to a *dead-end*, due to congestion for example, then the packet is re-routed to the previous hop it encountered and follows another path until the final destination is reached. At each hop, a next hop is selected based on the position of the destination relative to the current position using the priorities shown in Table I. Thus, in the worst case, a router maybe visited four times before declaring the packet undeliverable. The list of priorities ensures that the packet is

TABLE I
NEXT HOP SELECTION BASED ON CURRENT AND NEXT ROUTER POSITION

Destination's position relative to source router	*Next hop priority*			
	First	*Second*	*Third*	*Fourth*
North	North	East	West	South
North East	North	East	West	South
North West	North	West	South	East
East	East	South	North	West
West	West	South	North	East
South West	South	West	North	East
South East	South	East	North	West
South	South	East	West	North

```
Annealing_TestScheduling()
{
  MaxTime ← Total allowed time for the annealing process
  CurrentS ← InitialConfiguration()
  do {
      M = M_0
      do {
          violation ← True;
          Tries ← True;
          while (violation == true and tries < limit){
            if (iteration % 5 == 0)
                NewS = Flip(CurrentS);
            else if (iteration % 6 == 0)
                NewS = ChangeInput(CurrentS);
            else if (iteration % 7 == 0)
                NewS = ChangeOutput(CurrentS);
            NewS = ChangeOrder(CurrentS);
            NewS = TraceRoutes(NewS)
            tries ++
            if all routes have been traced
              violation ← false
          }
      NewCost=Cost(NewS)
      Δ_Cost = NewCost − CurrentCost
      if (Δ_Cost < 0)
      {
       CurrentS=NewS
       CurrentCost=Cost(CurrentS);
       If (NewCost < BestCost) then
         BestS=NewS
         BestCost = Cost(BestS)
      }
      else if (Random < e^(−Δ_Cost/T) then      // T = temperature
       CurrentS=NewS
       CurrentCost = Cost(CurrentS);
      M = M − 1
      } while (M ≥ 1)          // M is time until next parameter update
      Time = Time + M_0;
      T = α × T;                          // α = Cooling rate
      M_0 = β × M_0;                      // β = Iteration multiplier
  } while (Time < MaxTime);
}
```

Fig. 2. Annealing NoC test scheduling algorithm

```
Route ()
{
  Push the source router on the stack, S_t.
  while (S_t is not empty)
    Pop a router, R_i
    If R_i = Destination then
      route is found
  else if R_i has been visited four times then
      mark R_i as blocked.
  else {
    push R_i on the S_t
    // check for next hop and make sure there are no loops in the route
    select a next hop R_j based on priorities in Table I such that R_j ∉ S_t
      if ∃R_j such that it can accommodate traffic then
        push R_j on S_t
      else
        mark R_j as blocked.
  }
}
```

Fig. 3. Test Packets Routing

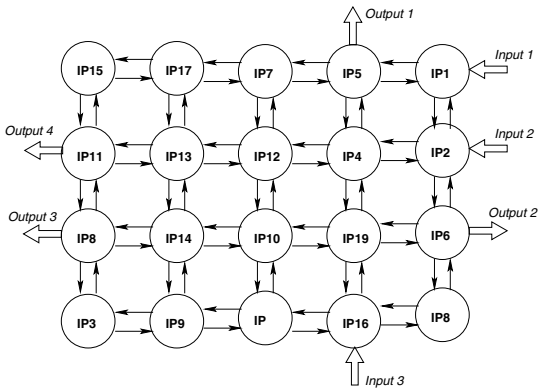

Fig. 4. p34392 SOC implemented using a 2-D mesh [14]

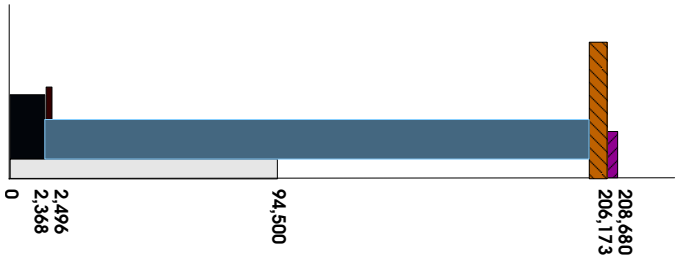

Fig. 5. Test Schedule on Input 3 of the p34392 benchmark

routed towards the final destination while guaranteeing that the route is deadlock free. The route is loop-less since each path is a depth-first search tree. The next position is determined according to a priority list that ensures that the flits are directed toward the destination position. If no blocking occurs the algorithm reduces to an XY routing algorithm.

IV. NoC Test Scheduling Algorithm

The algorithm (Figure 2) starts by selecting an initial configuration and then performs a sequence of iterations. During each iteration, a new configuration is generated by selecting the best input/output ports for the cores using the three neighborhood functions. The algorithm next performs test scheduling and allocates the routing paths in order to find the best route for the flits through the mesh. Test scheduling is performed on each input port based on the core priority. The test start time of a core is selected by finding the earliest time that can accommodate all the flits of the core. The variation in the cost between both iterations, Δ_{Cost}, is computed. If Δ_{Cost} is negative then the transition from C_i to C_{i+1} is accepted. If the cost function increases, the transition is accepted with a probability based upon the Boltzmann distribution. The temperature is gradually decreased throughout the algorithm from a high starting value where almost every proposed transition is accepted to a freezing temperature, where no further changes occur.

V. Experimental Results

We have implemented the proposed NoC test scheduling algorithm and attempted all the ITC'02 benchmark suite.

TABLE II
Test Time for the ITC'02 Benchmarks

Benchmark	# Cores	Input	Output	Test Time
u226	9	3	4	4, 194, 313
d281	8	3	4	94, 256
d695	10	1	2	56, 718
g1023	14	3	4	29, 013
f2126	4	2	2	651, 493
q12710	4	2	2	4, 015, 311
t512505	31	3	4	10, 300, 000
h953	8	3	4	228, 575
p22810	28	3	4	218, 328
p34392	19	2	2	1, 048, 739
p93791	32	2	3	625, 009
a586710	7	2	2	13, 400, 000

For each benchmark, we use the mapping and placement method in [14] in order to generate an interconnected *2-D* mesh. The benchmarks vary in connectivity, communication weight, size, and occupation. The results are shown in Table II. It was not possible to compare to other methods since testing results depend on the map as well as the blocks placement. Furthermore, most reported works did not include the core's test time in the cost function, and used the *xy* routing algorithm. The test scheduling results are shown in Table II. The simulated annealing cooling schedule was experimentally determined as follows: the temperature reduction multiplier, $\alpha = 0.99$ and T_{init} = 4000. The stopping criterion was $T \leq 0.001$ while the number of iterations, M, was set to 5 and the the iteration multiplier, β, to 100. The maximum allowed number of iterations, *MaxTime*, was set to 500.

References

[1] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, and G. D. Micheli, "Network-On-Chip Design and Synthesis Outlook," *Integration-The VLSI journal*, vol. 41, 2008.

[2] W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proc. DAC*, 2001.

[3] S. Murali and G. D. Micheli, "Bandwidth-constrained mapping of cores Onto NoC architectures," in *Proc. DATE*, 2004.

[4] E. Bolotin, A. Morgenshtein, I. Cidon, and A. Kolodny, "Automatic and Hardware-Efficient SoC Integration by QoS Network On Chip," in *Proc. ICECS*, 2004.

[5] L. Wang, C. Stroud, and N. Touba, eds., *System On Chip Test Architectures*. Morgan Kaufmann, 2007.

[6] J. Hu and R. Marculescu, "Energy and Performance-Aware Mapping for Regular NoC Architectures," *IEEE Trans. CAD*, vol. 24, no. 4, 2005.

[7] T. Dumitras and R. Marculescu, "On-Chip Stochastic Communication," in *Proc. DATE*, 2003.

[8] M. Pirretti, "Fault Tolerant Algorithms for Network-On-Chip Interconnect," in *Proc. IEEE Symp. on VLSI*, 2004.

[9] E. Cota, M. Kreutz, C. Zeferino, L. Carro, M. Lubaszewski, and A. Susin, "The Impact of NoC Reuse on the Testing of Core-Based Systems," in *Proc. VTS*, 2003.

[10] E. Cota, L. Carro, Luigi, and M. Lubaszewski, "Reusing an On-Chip Network for the Test of Core-Based Systems," *ACM TODAES*, vol. 9, no. 4, 2004.

[11] C. Liu, E. Cota, H. Sharif, and D. Pradhan, "Test Scheduling for Network-On-Chip With BIST and Precedence Constraints," in *Proc. ITC*, 2004.

[12] M. Hosseinabady, A. Banaiyan, M. Bojnordi, and Z. Navabi, "A Concurrent Testing Method for NoC Switches," in *Proc. DATE*, 2006.

[13] E. Cota and C. Liu, "Constraint-Driven Test Scheduling for NoC-Based Systems," *IEEE Trans. CAD*, vol. 25, no. 11, 2006.

[14] R. Farah and H. Harmanani, "A Method for Efficient Mapping and Reliable Routing for NoC Architectures With Minimum Bandwidth and Area," in *Proc. NEWCAS*, 2008.