International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 7, July - 2013

Noc Implementaion Using Generalized De Bruijn Graph

Sumalatha. N [1] , Smt. Divya Prabha [2] , Dr. M. Z.Kurian [3]

[1]Post Graduate M.Tech student, SSIT, Tumkur,Karnataka, India;

[2]Asst.prof, ECE Department, SSIT, Tumkur,Karnataka, India;

[3]Dean Academics, Registrar, Dr. & Head, ECE Department,SSIT, Tumkur,Karnataka, India;

Abstract -- NoC is a efficient on-chip communication architecture for SoC architectures. It enables integration of a large number of computational and storage blocks on a single chip. The Network-on-Chip (NoC) provides a scalable interconnection scheme. The concept uses a set of buses connected to routers or switches that interchange packets, much in the same way as traditional computer networks or multiprocessor machines do. This project proposes the generalized binary de Bruijn (GBDB) graph as a reliable and efficient network topology for a large NoC. A reliable routing algorithm to detour a faulty channel between two adjacent switches is proposed. A GBDB-based NoC in which the number of channels is less than that of Torus which has the same number of links is proposed.. The low energy consumption of a de Bruijn graph-based NoC makes it suitable for portable devices which have to operate on limited batteries. Also, the gate level implementation of the proposed reliable routing shows small area, power, and timing overheads due to the proposed reliable routing algorithm.

Key words: System-on-chips (SoCs), Generalized de Bruijn graph, network-on-chip (NoC).

1. INTRODUCTION

A system on a chip or system on chip (SoC or SOC) is an integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip. It may contain digital, analog, mixed-signal, and often radio-frequency functions—all on a single chip substrate.

Multiprocessor systems-on-chips (MPSoCs) have emerged in the past decade as an important class of very large scale integration (VLSI) systems. An MPSoC is a system on-chip,a VLSI system that incorporates most or all the components necessary for an application that uses multiple programmable processors as system components. MPSoCs are widely used in networking, communications, signal processing, and multimedia among other applications.With the growing complexity in consumer embedded products, new tendencies forecast heterogeneous Multi-Processor Systems-On-Chip (MPSoCs) consisting of complex integrated components communicating with each other at very high-speed rates. Intercommunication requirements of MPSoCs made of hundreds of cores will not be feasible using a single shared bus or a hierarchy of buses due to their poor scalability with system size, their shared bandwidth between all the attached cores and the energy efficiency requirements of final products.

To overcome these problems of scalability and complexity, Networks-On-Chip (NoCs) have been proposed as a promising replacement to eliminate many of the overheads of buses and MPSoCs connected by means of general-purpose communication architectures. However, the development of application-specific NoCs for MPSoCs is a complex engineering process that involves the definition of suitable protocols and topologies of switches, and which demands adequate design flows to minimize design time and effort.

IJERT

The network consists of wires and routers. Processors, memories and other IP-blocks (Intellectual Property) are connected to routers. A routing algorithm plays a significant role on network's operation. Routers make the routing decisions based on the routing algorithm. Different devices with different purposes have different requirements for routing algorithms. Thus there have been designed several routing algorithms with various features and purposes.

There are a couple of requirements that every Network on Chip implementation has to meet. Performance requirements are small latency, guaranteed throughput, path diversity, sufficient transfer capacity and low power consumption. Architectural requirements are scalability, generality and programmability. Fault and distraction tolerancy as well as valid operation are major on Quality of Service.

The generalized binary de Bruijn (GBDB) graph is a reliable and efficient network topology for a large NoC. The latency and energy consumption of the generalized de Bruijn graph are much less than those of Mesh and Torus network topologies.

2. SYSTEM DESIGN

In order to design an NoC a topology to connect switches together, and a routing algorithm is to be proposed**.**

2.1 Topology: Generalized binary de Bruijn graph is the topology used. Generalized binary de Bruijn graph prepares a high speed network to perform the communication among cores in a NoC. This graph can be defined as follows.

Definition*:* A generalized binary de Bruijn graph, GBDB, has (n) nodes,where (n) can be any desired natural number. Each two nodes i and j are connected together if they satisfy one of the following equations:

$i=2*j+ r \pmod n$, r= 0 or 1 (1)

$j=2*i + r \pmod n$, r= 0 or 1 (2)

The generalized de Bruijn graph has a lot of features that make it suitable for implementation of reliable networks. The most important feature, which is denoted in Theorem 2, is the logarithmic relationship between the diameter of a generalized de Bruijn graph and the number of its nodes.

Theorem 2: The diameter of a generalized binary de Bruijn graph GBDB , which is defined as the maximum among the lengths of shortest paths between all possible pairs of nodes, is not greater than $[\log_2 n]$.For example, the diameter of the generalized binary de Bruijn graph GBDB[14] is 4 i.e $[\log_2 14] \sim 4$.

In this subsection, some features of the generalized binary de Bruijn graph that are used in the rest of the paper is explained. Without loss of generality, the generalized de Bruijngraph with 14 nodes is used (see Fig. 2.1(a) as an example to explain the methods). Therefore, all the features, explained using this example, are applicable to other generalized binary de Bruijn graphs.

As shown in Fig. 2.1, using (1) and starting from Nodes 0 and n-1(i.e., Node 13), two link-disjoint binary spanning trees of generalized binary de Bruijn graph can be constructed .To construct the tree of Fig. 2.1(b), we start from Node 0 and using the modular equation (1), the connected nodes to Node 0 (i.e., Node 0, Node 1) is obtained. This equation shows that there is a self-loop around Node 0.For the sake of generality and simplicity, this self-loop in the tree structure is kept. Node 0 is a parent for Node 1. The remainder part in (1) (i.e.r) is used as a label for the link connected to corresponding Nodes. Using (1) for Node 1, its children (i.e., Nodes 2 and 3) is obtained. Note that in this tree (Tree 1 of Fig. 2.1(b), the node number of a child node is greater than the node number of its parent. this tree is constructed until all children nodes have a node number less than their parent's node number. Therefore, all leaf nodes in this tree have a number not less than n/2 (where n is the number of nodes in the corresponding de Bruijn graph). Tree 2 of Fig. 2.1(c) can be constructed by using the similar technique. Note that there is a kind of analogy between Tree 1 and Tree 2. Each Node i in one tree (e.g., Tree 1) is correspond to Node (n-i-1) in the other tree (e.g., Tree 2). For example two nodes i=2 and j= 5 are connected together in Tree 1, and their corresponding nodes that are Nodes 14-2-1=11 and 14-5-1=8 are also connected together in Tree 2.

Based on Fig. 2.1, each node has at most two children and at most two parents. A directed link that connects a node to its parent represents a *Parental* relation (P-relation); and a directed link that connects a node to its child represents a *filial* relation (F -relation). For example, the link between Nodes 12 and 10 in Fig. 2.1(a) shows a P-relation for Node 10, and an F -relation for Node 12.

IJERT

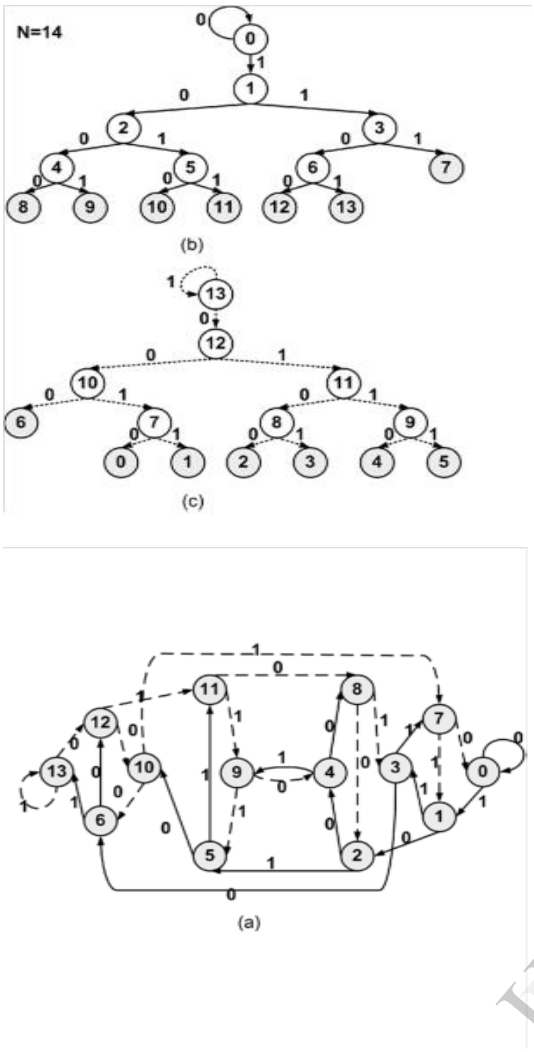

Fig. 2.1. Two link-disjoint spanning trees for GBDB (14). (a) Generalized de Bruijn graph with 14 nodes: (b) Tree-1 and (c) Tree-2.

2.2. Deadlock-Free Shortest Path Routing

This subsection explains how to find the shortest deadlock free path between two nodes in the GBDB. The following simple method explains how to find a path with length m (which is the diameter of the graph) between each two nodes.

Let us assume s and d be two nodes in GBDB , n and m the diameter of the graph, then

$T=d-s.2^m \pmod n$ is a binary number$(t_{m-1},\ldots\ldots.t_1,t_0)$ which defines path from s to d with length of m so that

$S=u_m \longrightarrow u_{m-1} \longrightarrow \ldots\ldots \longrightarrow u_1 \longrightarrow u0 = d$

Where for $u_i=2 * u_{i+1}+t_i \pmod n$ for $i =0,\ldots m-1$.

For example t=0100 for s=3 and d=10 in GBDB of fig 2.2. therefore ,the corresponding path is

$S=3 \xrightarrow{0} 6 \xrightarrow{1} 13 \xrightarrow{0} 12 \xrightarrow{0} 10=d$

Using virtual channels, the deadlock-free routing scheme is presented. For this purpose, each channel needs m-[(m-1)/2] virtual channels in which m represents the diameter of the GBDB. A unique ordered channel number is assigned to each channel so that in each tree all outgoing channel of a node are greater than the incoming channel. The algorithm is used to select virtual channels at each node to avoid a possible deadlock. The algorithm run in node x_{curr} gets three inputs x_{prev}, x_{next}, s and d which are the previous, next, source, and destination nodes.(Note that flit headers contain this information.) Then, the algorithm selects the next virtual channel based on its inputs.

2.3. Routing Implementation

This section briefly explains how to implement the proposed deterministic source routing algorithms. Based on Theorem 2, the diameter of a generalized de Bruijn graph has at most a logarithmic relationship with the number of nodes. Considering this logarithmic relationship, a simple routing algorithm that can be implemented in switches of an NoC with low area overhead is proposed. The packet format of the proposed routing algorithm, which utilizes the wormhole switching, consists of a few flits, starting with head flit, continuing with data flits, and ending with tail flit. The head flit is comprised of four fields which are: source address, destination address, tags, and flit-type. Flit-type bits determine the head, data, and tail flits. The tag bits that are grouped into two subfields (i.e., Tag-1 and Tag-2) determine the exact route from a source node to a destination node. Because our network uses source routing algorithm, the source core generates the tag bits which go through certain minor modification at each intermediate node along the path to destination.

When a packet arrives at node s, there are two possible actions that may be taken by . If the packet is desired for the core connected to , the switch at must deliver the packet to that core. On the other hand, if the packet is destined for another core, the switch at must forward the packet to one of its neighbor switches which is the next hop in the path. The selection of the next switch is based on the corresponding tag bits in the tag field of the head flit.

The afore mentioned Tag-1 and Tag-2 in the head flit are used to implement the routing algorithm in each switch. These two tags indicate to each intermediate node the next neighbor in the shortest path. Each bit in the Tag-1 field determines the F- or P-relation in a switch along the path, and each bit in the Tag-2 field determines the label assigned to each link in Fig.2.1. Source address, Destination address, Tag-1, and Tag-2 fields have the same length of bits (is the diameter of the generalized binary de Bruijn

IJERT

graph which for our example is [log $_2$ 14]=4). Therefore, the length of head flit is(k+k+k+k+2)=4k+2 bits (the two first terms are numbers of bits in source and destination addresses, respectively; the third and forth terms are numbers of bits in Tag-1 and Tag-2 and the last term shows the two bits needed to determine the type of flits in each packet). In other words, the length of head flit is 4*[log $_2$ n]+2 where n is the number of nodes in the generalized de Bruijn graph.

When a packet is received at switch, the following steps are carried out by the switch.

Step 1) If the destination address is , the packet is delivered to the core connected to the switch; otherwise, Steps 2 and 3 are performed.

Step 2) The first bit of Tag-1 is used for selecting either P-relation or F-relation. Because there are two P-relations and two F-relations, the LSB bit of Tag-2 is used for selecting the specific gender in P-relation or F-relation.

Step 3) When the next node in the path has been determined, Tag-1 and Tag-2 are circulated one bit to the right. Then the head flit is forwarded.

The following example elucidates the proposed routing algorithm with more details.

Example *2*: Fig.2.3 shows the GBDB (14) for a NoC with directions and link labels which are used to determine Tag-1 andTag-2, respectively. If we want to send a packet from Node 0 to Node 13 through path 0 → 7 → 10 → 6 → 13 then the head flit of this packet which is generated by core connected to Node 0 is"0000-1101-0011-0010-00," which based on Fig.2.1, "0000" is the Node 0 address, "1101" is the address of Node 13, 0011 isTag-1 (we have coded F-relation with 0 and P-relation with 1),"0100" is Tag-2 which are link labels along the path, and "00" determines the head flit. Fig. 2.3 shows the path and head flit in deferent places along the path for this example. When the head flit established the path, the data and tail flits traverse along this path using the wormhole routing without any changes. The tail flit will close the established path and release all channels along the path. In our implementation, a bit attached to each channel shows the state of the channel whether it is free or reserved.

2.4 Reliable Routing Algorithm Implementation

Each port in a node has a status bit, *f* , that determines the status of the R-channel connected to that port. If this bit is "1," the R-channel is faulty; otherwise, the link is functional. A diagnostic circuit periodically checks the R-channel and asserts the corresponding bit, *f*, in the presence of a fault in the R -channel.

Using the status bit (i.e.,*f*) and bits in the tag fields of the head flit, each switch can detour a faulty R-channel using another R-channel connected to a member of its family, as shown in Fig. 2.4. Replacing the faulty R-channel with another R-channel in the family is a local decision that each switch can handle it easily. Using the family members to detour the faulty R-channel needs to add two extra hops to the original path. As a result, two extra bits in each tag are enough to detour a faulty R-channel. Therefore, in the fault-tolerant switch, we add two bits to each tag field, i.e., a tag field has [Log_2 n]+ 2 bits.

In a node which changes the route, let us assume that the LSB of Tag-1 and Tag-2 are a , and b, respectively. If an R-channel through which the switch sends a packet is faulty\(i.e., *f*=1), by expanding the LSB of Tage-1 (i.e.,a) to three bits(a,$\bar{a}$,a) and expanding the LSB of Tage-2 (i.e.b,) to three bits($\bar{b}$,b,$\bar{b}$), the switch can detour the packet around the faulty R-channel using other members of the family. After this two-bit tag expanding, the switch connected to the faulty R-channel performs the normal routing algorithm.

The following example elucidates the proposed reliable routing algorithm with more details.

Example3: In Example 2 in which a packet traverses from Node 0 to Nodes 13, assume that theR-channel between Nodes 7 and 10 is faulty. The family covering R-channel from Node 7 to Node 10 consists of Nodes 7, 3, 6, and 10. Therefore, the new path from Node 0 to Node 13 is 0 → 7 → 3 → 6 → 10 → 6 → 13.Fig. 2.5 shows the modified head flit during this transfer.

IJERT

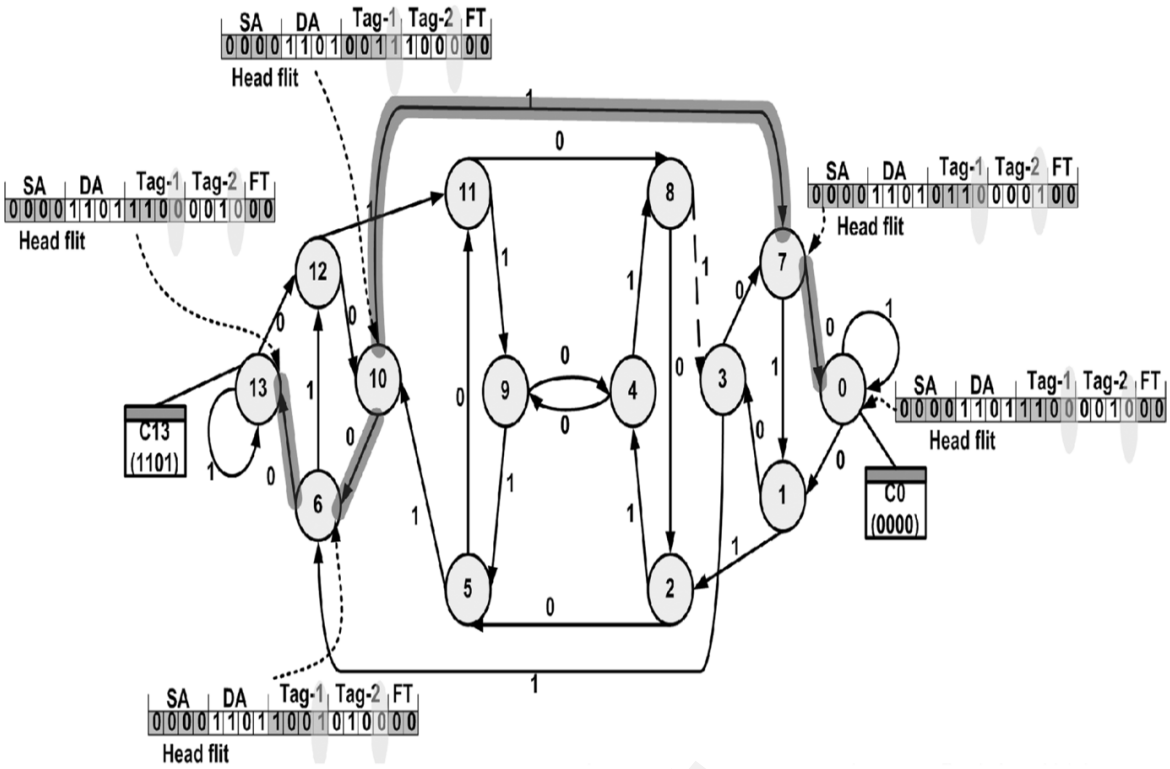

Fig 2.3 Head flit from Node 0 to Node 13 in GBDB(14)

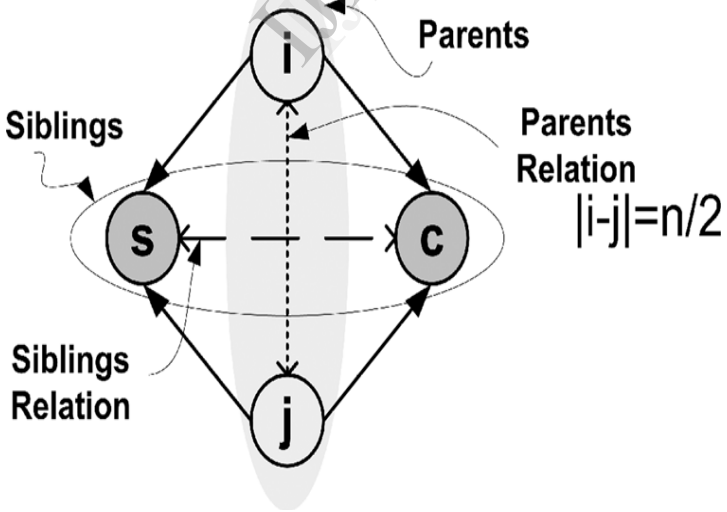

Fig. 2.4. Parents, siblings, and their relations

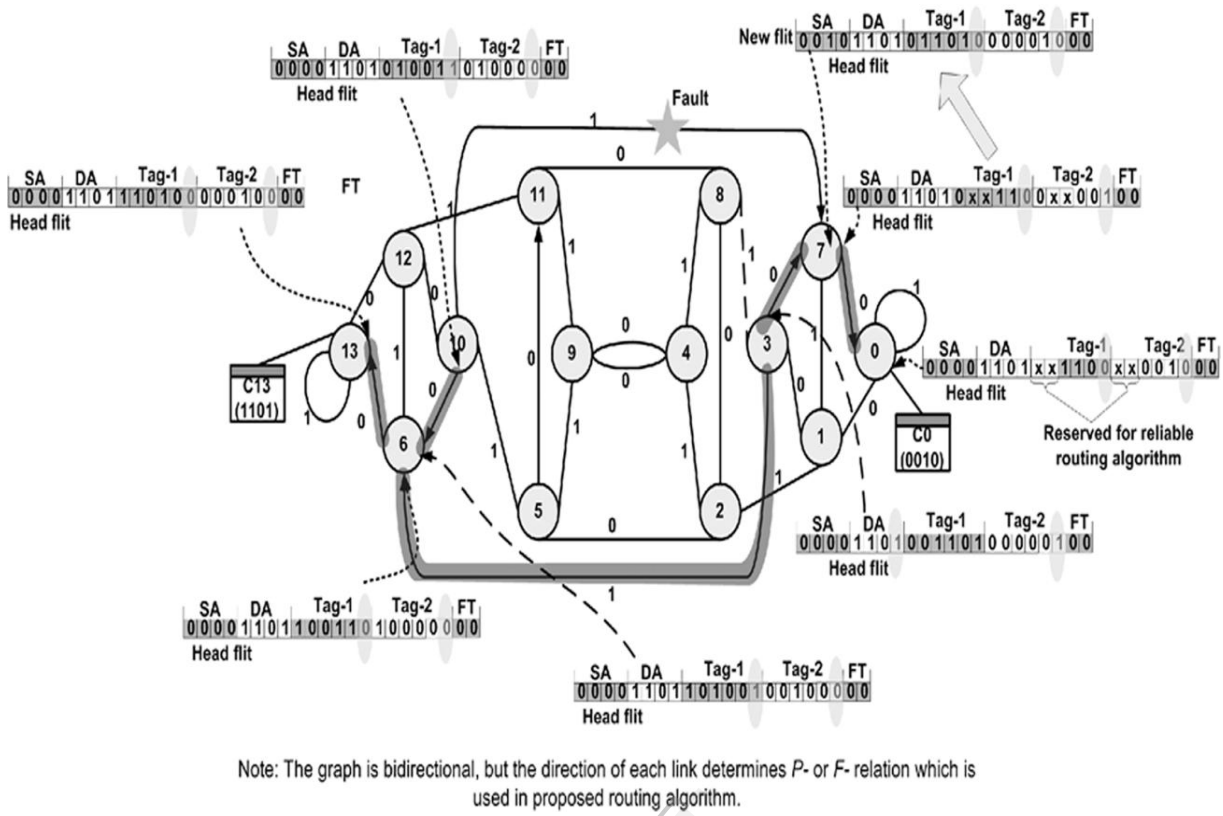

Fig. 3.5. Head flit from Node 0 to Node 13 in GBDB, when link between Nodes 7 and 10 is faulty.

4 EXPERIMENTAL RESULTS

The design was simulated using ModelSim and was tested for functionality by giving various stimuli.

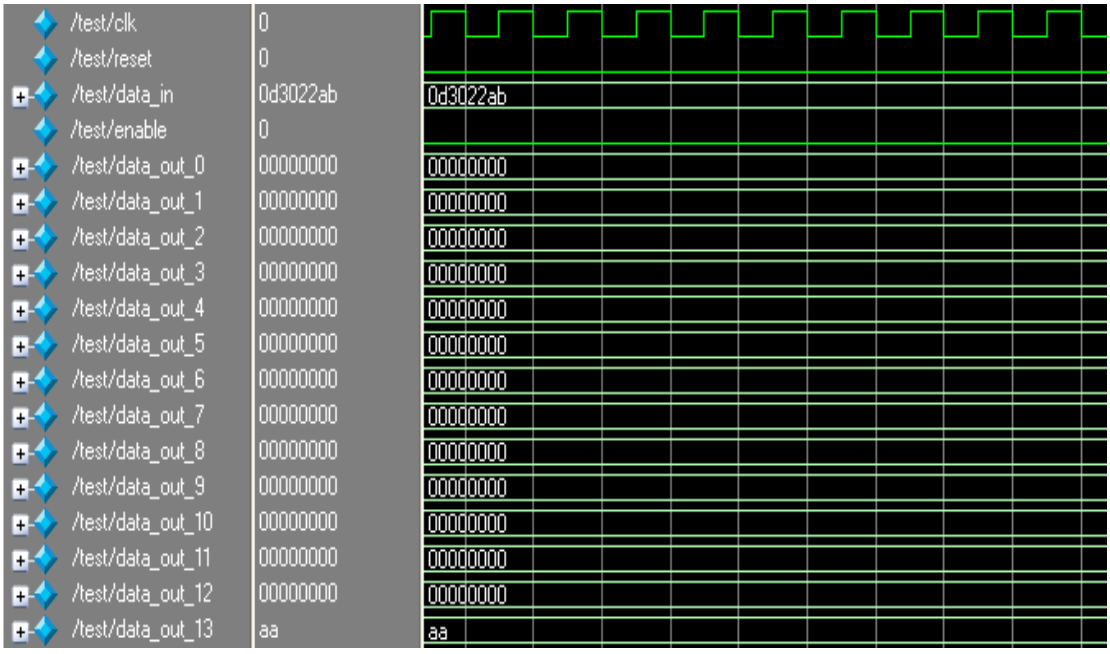

Fig.3.6 Data movement from node 0 to 13

5 CONCLUSION

This project investigates the use of generalized binary deBruijn graph as the network topology in a large network-on chip. A deadlock-free reliable routing algorithm will be presented considering channel faults.

REFERENCE

[1] L. Benini and G. D. Micheli, "Networks on chips: A new SoC paradigm," *IEEE Computer*, vol. 35, no.1, pp. -70–78, Jan. 2002.

[2] Deepthi chamkur .V and Vijayakumar"Reliable Routing & Deadlock free massive NoC Design with Fault Tolerance based on combinatorial application".International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-1, Issue-5, June 2012 .

[3] H. P. Hofstee, "Future microprocessors and off-chip SOP interconnect,"*IEEE Trans. Adv. Packag.*, vol. 27, no. 2, pp. 301–303, Feb. 2000.

[4] D.Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep./Oct. 2007.

[5] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. IEEE Int. Conf. Des. Autom.*, Jun. 2001, pp. 684–689.

[6] S. R. Vangal et al., "An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS," IEEE J. Solid-State Circuits, vol. 43, no. 1, pp. 29–41, Jan. 2008

[7] S. M. Reddy, D. K. Pradhan, and J. G. Kuhl, "Direct graphs with minimumdiameter and maximal connectivity," Sch. Eng., Oakland, Univ. Techn. Rep., Jul. 1980.

[8] H. Moussa, A. Baghdadi, and Jézéquel, "Binary de Bruijn on-chip network
for a flexible multiprocessor LDPC decoder," in *Proc. 45th Annu. Des. Autom. Conf. (DAC)*, 2008, pp. 429–434.

IJERT