Microprocessing and Microprogramming 32 (1991) 593-600
North-Holland

On hardware for generating routes in Kautz digraphs

Gerard J.M. Smit, Paul J.M. Havinga, Pierre G. Jansen, Fokke de Boer, Bert Molenkamp

University of Twente, Dept. of Computer Science
P.O. Box 217, 7500 AE Enschede
the Netherlands
e-mail smit@cs.utwente.nl

Abstract

In this paper we present a hardware implementation of an algorithm for generating node disjoint routes in a Kautz network. Kautz networks are based on a family of digraphs described by W.H. Kautz [Kautz 68]. A Kautz network with in-degree and out-degree d has $N = d^k + d^{k-1}$ nodes (for any cardinals d, k> 0). The diameter is at most k, the degree is fixed and independent of the network size. Moreover, it is fault-tolerant, the connectivity is d and the mapping of standard computation graphs such as a linear array, a ring and a tree on a Kautz network is straightforward.
The network has a simple routing mechanism, even when nodes or links are faulty. Imase et al. [Imase 86] showed the existence of d node disjoint paths between any pair of vertices. In Smit et al. [Smit 91] an algorithm is described that generates d node disjoint routes between two arbitrary nodes in the network. In this paper we present a simple and fast hardware implementation of this algorithm. It can be realized with standard components (Field Programmable Gate Arrays).

Keywords: interconnection networks, Kautz graphs, routing algorithms, routing hardware, VHDL.

1. Introduction.

COMPAS (COMmunication in PArallel Systems) is a research project carried out at the University of Twente. The goals of this project are the design, realization and evaluation of interconnection networks for parallel systems.
In this paper we consider regular interconnection networks for multi-computer systems. A multi-computer system is defined as a collection of linked node computers (abbreviated as nodes), in which the nodes communicate via message passing [Dally 87]. We assume that each node consists of a communication processor and one or more node processors with local memory (see fig. 1).

The choice of a suitable interconnection network topology is an important issue of parallel systems. As the number of processors increases, the demands imposed on the network become more strict. There are several, sometimes conflicting, demands that have to be considered, such as:

- *Diameter*: The network should have a small diameter and thus a low average distance.
- *Degree*: The degree of a node should be independent of the actual size of the network.).

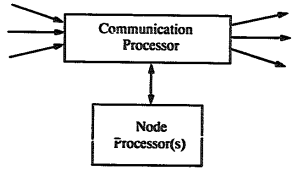

Fig.1: Architecture of a node.

- *Fault tolerance*: The network should be fault-tolerant. In a system consisting of a large number of nodes, the probability of a node or a link failure, cannot be neglected. Connectivity and diameter are important parameters for the construction of fault-tolerant networks [Bermond 89].
- *Embedding computation graphs*: The network should allow for efficient and simple mappings of standard computation graphs like lines, trees and

meshes [Hilbers 89].
- *Communication delays*: In order to achieve a low latency and a high throughput, communication delays between nodes must be short. Note that these factors are also influenced by topology related factors such as network diameter and routing strategy.
- *Routing:* The network should have a simple routing mechanism.

In this paper we focus on Kautz networks [Kautz 68]. These networks have drawn some attention recently [Bermond 89, Fiol 84, Imase 83].
We finish this introduction with some definitions. The standard definitions from graph theory, such as *degree, diameter, distance, connectivity,* etc. can be found in [Berge 73]. Let $N(x)$ denote the set of neighbours of a vertex x. A graph is called *degree-regular* if all vertices have the same degree. Let IN(x) be the set of neighbours that have an arc pointing to x (*in-neighbours*). The *in-degree* $d^-(x)$ is the cardinality of IN(x). The *out-degree* $d^+(x)$ and the set of *out-neighbours* ON(x) are defined in a similar way.

2. Kautz graphs.

2.1 Definition.

In this section a definition of Kautz digraphs is given. This family of graphs was first described by W.H. Kautz [Kautz 68] and has been rediscovered by Imase et al. [Imase 83] and Fiol et al. [Fiol 84]. Kautz graphs have a minimal diameter and maximal connectivity for a given number of nodes and a given degree. Fiol et al. showed that for a given out-degree d and diameter k the number of vertices in a Kautz graph is larger than $(d^2 - 1)/d^2$ times the non-attainable Moore bound [Bridge 80][1]. This property makes a Kautz graph suitable as an interconnection network for large scale parallel computer systems.

In this paper we will only consider directed Kautz graphs (Kautz digraphs). The undirected Kautz graphs can be obtained from the associated digraphs by removing the orientation, and the parallel edges.

Definition [Kautz 68]:
The Kautz digraph K(d,k) with in-degree and out-degree d and diameter k is the digraph from which the vertices are labeled with words of length k from an alphabet of d+1 letters by removing those words in which there are two consecutive identical letters. The vertices are labeled as words $(x_1,...,x_k)$, where x_i belongs to an alphabet of d+1 letters, and $x_i \neq x_{i+1}$, for $1 \leq i \leq k-1$. There exists an arc from a vertex x to a vertex y if and only if the last k-1 letters of x are the same as the first k-1 letters of y. So the vertex $(x_1,...,x_k)$ is neighbour to d vertices $(x_2,...,x_k,z)$, where z is any letter different from x_k.

1. For a given degree d and diameter k there is a theoretical upperbound on the number of nodes N in the graph. This upperbound is known as the Moore bound. $N \leq 1 + d + d^2 + d^3 + .. d^k$

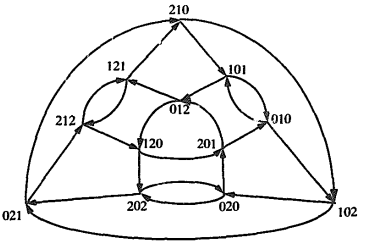

Fig. 2: Example of a Kautz graph K(2,3).

2.2 Properties of Kautz digraphs.

Some properties of Kautz digraphs are:

- The number of vertices $N = d^k + d^{k-1}$ (see Table 1).
- The degree of the graph is *fixed* and independent of N. Networks of arbitrary large size can be built using (VLSI) components as nodes with a *fixed* number of connections per node. Whereas other networks, such as the binary hypercube, require the number of connections per node to *increase* with the number of nodes, Kautz networks have a fixed degree.
- The *diameter* of the network is small, at most k (< ^{d}log N).
- A Kautz network is *fault tolerant*. The connectivity of K(d,k) equals d [Imase 86]. The diameter of the network in case of faulty nodes has also been studied by Imase et al. They showed the existence of d vertex disjoint paths between any pair of vertices in K(d,k), one of a length of at most k, d-3 of a length of at most k+1 and two of a length of at most k+2. This means that the performance degradation due to increased routing distances resulting from faults is fairly low.
- Another interesting property of the network is the fact that it allows for *self routing* of messages, both when the network is fault free, as well as when

some nodes or links are faulty. Self routing refers to the ability to route messages from node to node using only the address of source and destination (see section 3).

- A Kautz digraph can *emulate standard computation graphs*. Fiol [Fiol 84] showed that Kautz digraphs are line digraphs of Eulerian digraphs, so a Kautz digraph contains a Hamilton cycle. This implies that a linear array and a ring can be mapped on it. Furthermore a tree can be mapped on a K(d,k) digraph. Because the diameter of a K(d,k) is at most k ($<^d\log N$) a mesh can be embedded in O($^d\log N$).

2.3 Comparison with other graphs.

Table 1 compares Kautz digraphs with 'de Bruijn' digraphs [de Bruijn 46] and the binary hypercube [Hillis 85], [Seitz 85]. The de Bruijn digraph has been selected because its definition is closely related to Kautz digraphs.
The difference between a de Bruijn digraph B(d,k) and a Kautz digraph K(d,k) is that in a de Bruijn digraph two consecutive letters in the word representing a particular vertex may be equal. As a consequence, this digraph contains self loops.

	d=k=4	d=k=6	d=k=8	number of nodes
hypercube	16	64	256	$N = 2^k$
de Bruijn	16	729	65536	$N = d^k$
Kautz	24	972	81920	$N = d^k + d^{k-1}$

Table 1.: Number of nodes of some graphs.

Note that for the de Bruijn and Kautz digraphs the out-degree and in-degree are half the degree mentioned in the table. Thus a Kautz digraph with in-degree and out-degree 4 and a diameter of 8 connects 81920 nodes, which is significantly more than the 256 nodes in a hypercube.

3. Routing in a Kautz digraph.

3.1 Generic routing.

One of the interesting properties of a Kautz digraph is its straightforward way of routing. This follows immediately from the definition and is known as the self-routing property.
To find a route R(x,y) from $x = (a_1, .., a_k)$ to $y = (b_1, .., b_k)$, we first select one particular out-neighbour vertex $(a_2, .., a_k, b_1)$ of x (provided $b_1 \neq a_k$). In other words, we find the first intermediate node $(a_2, .., a_k, b_1)$ of the route R(x,y) from x to y, by applying a '*shift*' operation on the letters of the source x, where a_1 is shifted out and b_1 is shifted in. We continue this process of shifting out source letters and shifting in destination letters until the source word is completely replaced by the destination word (see Table 2). In this way we can (by concatenation of source and destination words) generate a straightforward generic route $R(x,y) = <a_1, .., a_k, b_1, .., b_k>$ from source to destination with a length of k. (The case $b_1 = a_k$ is covered in the next section.)

$(a_1, a_2, a_3, ... , a_{k-1}, a_k)$	(source node)
$(a_2, a_3, ..., a_{k-1}, a_k, b_1)$	(first intermediate node: $\in \Gamma(x)$)
$(a_3, ..., a_{k-1}, a_k, b_1, b_2)$	(second intermediate node)
. .	
. .	
$(a_k, b_1, b_2, ..., b_{k-1})$	(last intermediate node: $\in \Gamma^{-1}(y)$)
$(b_1, b_2, ..., b_{k-1}, b_k)$	(destination node)

Table 2: Routing in a Kautz digraph (assume $a_k \neq b_1$).

The above algorithm gives a route of length k. However, in general there may be routes of length less than k. If there exists a route of length m (m<k), then the last k-m letters of the source are equal to the first k-m letters of the destination.

3.2 Generating node disjoint routes.

As we mentioned in section 2, Imase et al. [Imase 86] showed the existence of d vertex disjoint routes between any pair of vertices in K(d,k). For a Kautz graph K(d,k) their algorithm returns one route of length ≤ k, d-3 routes of length k+1 and two routes of length k+2. Such a route might contain loops. Our algorithm is an improved version of the algorithm proposed by Imase. We allow more than one route of length ≤ k and at most that many plus one of length k+2. The remaining routes are of length k+1. The routes are generated with increasing length, are node disjoint and are free of loops. See Smit et al. [Smit 90] for an extensive proof of the algorithm.

Example:

Using the graph of figure 2 we find the following two routes from (120) to (201) with the Imase approach:
$R_1 = < 1201 >$ and $R_2 = < 12020201 >$.
{R_2 contains a loop!}
Our algorithm finds the following two routes:
$R_3 = < 1201 >$ and $R_4 = < 120201 >$.
{R_3 and R_4 both have a length ≤ k}

3.3 Informal description.

Our algorithm is based on the property that two generated routes that depart from the source node via two different out-nodes, *and* arrive at the destination node via two different in-nodes, are node disjoint.

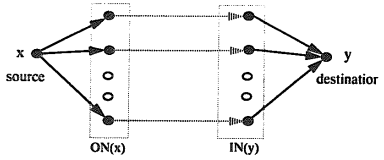

Fig. 3.: Routing strategy.

The algorithm finds all routes in three phases.

Phase A:	Routes with length ≤ k (R_I).
Phase B:	Routes with length k+1 (R_{II}).
Phase C:	Routes with length k+2 (R_{III}).

We start with generating the routes R_I, after that the routes R_{II} and finally the routes R_{III} are generated. Let ON(x) denote the set of out-neighbour nodes (out-nodes) of x and IN(x) the set of in-neighbour nodes (in-nodes). Let $ON_u(x)$ denote the set of out-nodes of x that have been used in a previous path and $IN_u(x)$ the set of used in-nodes. Initially $ON_u(x)$ and $IN_u(x)$ are empty.

Phase A: Routes R_I.

There is a unique, ordered set of routes R_I with length ≤ k. Only when there is an overlap of the source and destination words, then there is a possible candidate R_c for set R_I. For each route R_c we test whether the out-neighbour of the route $\notin ON_u(x)$ and the in-neighbour of the route $\notin IN_u(y)$ or not. If one of the neighbour nodes was used before, the route is rejected, because it is not node disjoint. Otherwise a new route is found and the neighbour nodes are added to $ON_u(x)$ and $IN_u(y)$ respectively. It is straightforward that searching starts with an overlap for k-1 positions.
The set of routes R_I is uniquely defined by:

$$\begin{aligned} R_I = \{\ & v : v \in \{0 .. k-1\} \\ & \wedge (a_2, .. a_k, b_{v+1}) \in ON(x) \\ & \wedge (a_{k-v}, b_1, .. b_{k-1}) \in IN(y) \\ & \wedge (a_2, .. a_k, b_{v+1}) \notin ON_u(x) \\ & \wedge (a_{k-v}, b_1, .. b_{k-1}) \notin IN_u(y) :: \\ & :: = <a_1, .., a_k, b_{v+1}, .., b_k> = <a_1, .., a_{k-v}, b_1, .. b_k> \\ & \} \end{aligned}$$

Phase B: Routes R_{II}.

If all routes with length ≤ k have been generated, the routes R_{II} can be found by inspecting the sets ON(x) and IN(y). If the set ON(x) contains nodes $c_{ON(x)}$ such that the last letter of $c_{ON(x)}$ equals the first letter of a node $c_{IN(y)}$ of IN(y) *and* for both nodes $c_{ON(x)} \notin ON_u(x)$ and $c_{IN(y)} \notin IN_u(y)$, then we have found a new route of length k+1 with $c_{ON(x)}$ as the out-neighbour node and $c_{IN(y)}$ as the in-neighbour node.

The set of routes R_{II} is defined as a set:

$$\begin{aligned} R_{II} = \{\ & (a_2, .., a_k, c_1) \in ON(x) \\ & \wedge (c_1, b_1, .., b_{k-1}) \in IN(y) \\ & \wedge (a_2, .., a_k, c_1) \notin ON_u(x) \\ & \wedge (c_1, b_1, .., b_{k-1}) \notin IN_u(y) :: \\ & :: < a_1, .., a_k, c_1, b_1, .., b_k> \}. \end{aligned}$$

Phase C: Routes R_{III}.

If all routes with length ≤ k+1 have been generated and $|ON_u(x)|^{1)} \neq d$, then there are routes with length equal to k+2. These routes can be found by arbitrary choosing one node u of set ON(x) with $u \notin ON_u(x)$ as first intermediate node and choosing one node v from IN(y) with $v \notin IN_u(y)$ as the last intermediate node. This completely defines a set of routes of length k+2:

$$\begin{aligned} R_{III} = \{\ & (a_2, .., a_k, c_2) \in ON(x) \\ & \wedge (c_3, b_1, .., b_{k-1}) \in IN(y) \\ & \wedge c_2 \neq c_3 \wedge (a_2, .., a_k, c_2) \notin ON_u(x) \\ & \wedge (c_3, b_1, .., b_{k-1}) \notin IN_u(y) :: \\ & :: < a_1, .., a_k, c_2, c_3, b_1, .., b_k> \}. \end{aligned}$$

Note that if there are more than one routes of length k+2 then these routes are not uniquely defined.

Example:

In figure 2 Re = < 021201 > is a route from (021) to (201) via node (212) and (120).
on_{Re} = (212); in_{Re} = (120); ON(021) = {(212), (210)}; IN(201) = {(120), (020)}.

Appendix A shows a condensed VHDL [VHDL 87] description of the specification.

4. Implementation.

The hardware implementation of the routing algorithm is also described in VHDL. A simplified schematic representation is given in figure 4. The datapath consists of a *match block*, two *shift/increment* parts and a *shift-block*. The *used in-nodes* and *used out-nodes* parts take care of the registration for the used in-nodes and out-nodes. All blocks are supervised by the *control unit*. There are external signals for initializing, for generat-

1. |S| denotes the cardinality of set S.

ing a new route and for status information (such as: a route has been found or there are no more routes). Actually the last signal is not necessary, as it follows from the degree of the network.

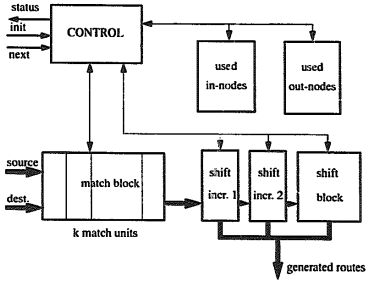

Figure 4.: Block diagram of the implementation.

The route generator operates as follows:

1. In the initialization phase the source and destination are shifted into the match block and the control registers (used in-node and used out-node) are set to their initial values.
2. The destination is shifted via the shift/increment parts to the shift block until an overlap between source and destination is found (i.e. all match units indicate a match). Only if an overlap is found *and* the corresponding in-nodes and out-nodes have not been used before (which is checked by the used in-node and used out-node units), a new route is found. This is signalled to the environment by the control unit. When the control unit receives the signal to generate the next route, the above process continues.
3. For routes with length k+1 the first shift/increment part generates all possible letters that can be inserted. If a letter is not used before, which is checked in the modules used in-nodes and used out-nodes, a new route is found.
4. For routes with length k+2 both shift/increment parts generate these letters. When all out-nodes have been used, no more routes can be generated. This is also signalled to the environment.

The VHDL description is designed hierarchically. The matchblock for instance consists of k blocks (match units) that each compare one letter of a node word. Figure 5 shows such a match unit.

It consist of the registers *src_part* and *dst_part* that

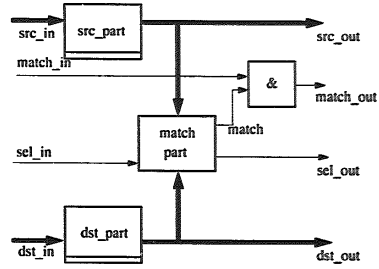

Figure 5.: Block diagram of a match unit.

contain a letter of the source node and the destination node respectively, and a comparator (*match part*). The comparator can be switched off by the control signal sel_in (match output always high). The src_out, match_out, sel_out and dst_out signals are connected to its neighbour's src_in, match_in, sel_in and dst_in signals.

Initially all sel_out signals (and so all but one sel_in signals) are cleared and the left most sel_in signal is set to true. At each 'tick of the clock' the destination and the sel_in signal is shifted from left to right, disabling one more match unit. Eventually, after k clock ticks all match units are disabled and the control unit is signalled to enter the phase for generating routes with length k+1.

										$ON_u(x)IN_u(y)$		Route
0	1	2	3	4								
	2	3	4	3	0					3	1	(0123430)
		2	3	4	3	0						
			2	3	4	3	0					
				2	3	4	3	0				
					2	3	4	3	0	2	4	(0123423430)
					0	2	3	4	3	0	0	0 (01234023430)
					1	2	3	4	3	0		
					2	2	3	4	3	0		
					3	2	3	4	3	0		
					4	2	3	4	3	0		
					0	0	2	3	4	3	0	
					1	1	2	3	4	3	0	
					1	2	2	3	4	3	0	
					1	3	2	3	4	3	0	1 3 (012341323430)

Table 3.: Routes from (01234) to (23430) in K(4,5).

5. Conclusion.

Kautz graphs form a class of interconnection networks with nice properties such as: diameter at most k (for N = $d^k + d^{k-1}$), the degree is independent of the network size, the network is fault-tolerant, it can embed standard computation graphs and has a simple routing algo-

rithm. Kautz networks have the advantage of connecting considerably more processors for a given degree and diameter than other networks. Therefore they are an important candidate for interconnection networks in a new generation of massively parallel computer systems.
We have presented an algorithm that finds d node disjoint paths between arbitrary nodes in the network. We have shown that a hardware implementation of this algorithm is simple and straightforward. The algorithm is described in VHDL. The hardware can be realized with standard components such as Field Programmable Gate Arrays. We aim at realizing the route generator, together with a dedicated communication processor, in one single FPGA.

References.

[Berge 73] Berge C.: "Graphs and hypergraphs"; North-Holland Publishing Co. 1973.

[Bermond 89] Bermond J.C., Homobono N., Peyrat C.: "Large Fault-Tolerant Interconnection Networks", Graphs and Combinatorics, 1989.

[Bridge 80] W.G. Bridge and S. Toueg: On the impossibility of directed Moore graphs. J. Combin. Theory, series B29, pp. 339-341, 1980.

[de Bruijn 46] de Bruijn N.G.: "A combinatorial problem"; Koninklijke Nederlandse Academie van Wetenschappen Proc. A49, pp 758-764; 1946.

[Dally 87] Dally W.J.: "A VLSI Architecture for Concurrent Data Structures", Ph.D. thesis, Computer Science, California Institute of Technology, 1987.

[Fiol 84] Fiol M.A., Yebra J.L.A., Alegre I.: "Line digraph iterations and the (d,k) digraph problem"; IEEE Trans. on Computers C-33, pp 400-403; 1984.

[Hillis 85] Hillis W.D.: "The connection machine.", The MIT press, 1985.

[Hilbers 89] Hilbers P.A.J. "Mapping of algorithms on processor networks"; Ph.D. thesis, University of Groningen; 1989.

[Imase 83] Imase M. and Itoh M.: "A design for directed graphs with minimum diameter.", IEEE Trans. on Comp., vol c-32, pp 782-784, 1983.

[Imase 86] Imase M., Soneoka T., Okada K.: "A fault-tolerant processor interconnection network" (original in Japanese); translated in Systems and Computers in Japan, Vol 17, no 8 pp 21-30, 1986.

[Kautz 68] Kautz W.H.: "Bounds on directed (d,k) graphs. Theory of cellular logic networks and machines", AFCRL-68-0668 Final report, pp 20-28, 1968.

[Seitz 85] Seitz C.L.: "The cosmic cube"; Comm. ACM, Vol 28, no 1, jan. 1985.

[Smit 90] Smit G.J.M., Havinga P.J.M., Jansen P.G.: "An algorithm for generating node disjoint routes in Kautz digraphs", INF/90-43, memorandum University of Twente, dept. Computer Science, october 1990.

[Smit 91] Smit G.J.M., Havinga P.J.M., Jansen P.G.: "An algorithm for generating node disjoint routes in Kautz digraphs", to appear in Proceedings Fifth International Parallel Processing Symposium, Anaheim California, May 1991.

[VHDL 87] IEEE Standard VHDL, Language Reference Manual, (IEEE Std. 1076-1987), IEEE, New York, 1987.

Appendix A: Condensed specification of the route generator in VHDL

```
ENTITY route_generator IS
  GENERIC(d : INTEGER := 4;       -- degree
          k : INTEGER := 5);      -- diameter
  PORT (SIGNAL src, dst       : IN node_type; --source,destination
        SIGNAL route          : OUT route_type;
        SIGNAL init, next     : IN vlbit;
        SIGNAL ready, done    : OUT vlbit );
END route_generator;

ARCHITECTURE behaviour OF route_generator IS

generate_route: PROCESS
  ... -- variable declaration part
BEGIN
  WAIT UNTIL (init ='0') OR (next = '0');
  IF init = '0' THEN
    -- initializations (a.o.: overlap := k)
  END IF;
  route_found := FALSE;
  WHILE (NOT route_found) AND (NOT routing_done) LOOP
    CASE state IS
      WHEN phase_A => -- routes with length <= diameter  (k)
          match_found := FALSE;
          WHILE overlap > 0 AND NOT match_found LOOP
            overlap := overlap-1;
            match_found := match( src, dst, overlap );
          END LOOP;
          IF match_found THEN
            IF outnode_free( dst(k-overlap) ) AND innode_free( src(overlap+1) ) THEN
              route <= dst(k-overlap DOWNTO 1) & extend( dst(1), overlap+2 );
              set_in_out_node( src(overlap+1), dst(k-overlap) );
              route_found := TRUE;
            END IF;
          ELSE
            state := phase_B;
          END IF;
      WHEN phase_B => -- routes with length diameter+1  (k+1)
          route_found := outnode_free( r1_cntr ) AND innode_free( r1_cntr );
          WHILE r1_cntr < d AND NOT path_found LOOP
            r1_cntr := r1_cntr+1;
            route_found := outnode_free( r1_cntr ) AND innode_free( r1_cntr);
          END LOOP;
          IF route_found THEN
            route <= r1_cntr & dst & dst(1);
            set_in_out_node( r1_cntr, r1_cntr );
          ELSE
            state := phase_C;
          END IF;
      WHEN phase_C => -- routes with length diameter+2 (k+2)
          route_found := outnode_free( r2_outnode_cntr );
          WHILE r2_outnode_cntr < d AND NOT route_found LOOP
            r2_outnode_cntr := r2_outnode_cntr+1;
            route_found := outnode_free( r2_outnode_cntr );
          END LOOP;
          IF route_found THEN
            WHILE NOT innode_free( r2_innode_cntr ) LOOP
              r2_innode_cntr := r2_innode_cntr+1;
            END LOOP;
            route <= r2_innode_cntr & r2_outnode_cntr & dst;
            set_in_out_node( r2_innode_cntr, r2_outnode_cntr );
          ELSE
            route_done := TRUE;
          END IF;
    END CASE;
  END LOOP;
END PROCESS generate_route;
END behaviour;
```