Microprocessing and Microprogramming 34 (1992) 59–62
North-Holland

On the design of a dynamic reconfigurable network switch

Gerard J.M. Smit, Paul J.M. Havinga, Pierre G. Jansen

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

Keywords: Kautz graphs, programmable architecture, network switch, Field Programmable Gate Array (FPGA).

1. Introduction

In this paper we will present a reconfigurable network switch 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]. As a communication network for our system we use a Kautz network [Kautz 68].
Each node consists of three autonomous sub-systems: a Computation Processor (CP) with local memory, a Router (R) and a Network Switch (NS). The Computation Processor is a standard off-the-shelf type of processor that executes application programs. The Router provides the interface to other nodes and implements the communication protocol. The routes in the network are generated by the Route Generator (RG) [Smit 91b] which is part of the Router. The Network Switch and Router are implemented with Field Programmable Gate Array (FPGA) technology [Xilinx 91]. In this technology the gate arrays can be re-programmed an unlimited number of times. Essential in our approach is that FPGAs are used as dynamic programmable units, which function can be changed on-the-fly under program control. Therefore they can be used in designs where hardware is changed dynamically, or when hardware must be adapted to different user applications.

In order to provide full connectivity in a network of computers, routing mechanisms must be used. These mechanisms must satisfy a number of requirements such as: free of deadlocks, no starvation, low latency and high throughput. Well-known routing mechanisms are store-and-forward, wormhole routing, virtual cut-through. When a message in the network is unable to proceed because some resource it needs is held by other messages (collision) some action has to be taken. Possible options are: blocking the message, buffering the message prior to the node where the collision occurs, dropping and retransmission of the message, or misrouting the message to a free link. It is known that some of these solutions have a potential danger of deadlock. There are a number of mechanisms for avoiding communication deadlocks in networks. Most techniques such as virtual networks and class climbing, are based on breaking loops in the dependency graph [Dally 87]. Dropping and retransmission of messages is inherently free of deadlocks. This last method, also called the method of the nosy worms [Whobrey 88], is used in the configuration presented in this paper. The method of misrouting as used in the Connection-Machine is not applicable, because misrouting is not deadlock free and in case of a misroute a new route has to be computed.

The main function of the *Network Switch* is to route messages in the communication network. The NS does not compute nor change the contents of the messages. It only uses the information in the route field of a message to control the destination of that message. The route itself is computed by the Router and the Route Generator. The NS communicates with other switches and the Router via links. The NS manages incoming messages autonomously: establishes the route and passes the data of the messages through or returns a message if all links are busy. Because the switches are implemented in FPGA technology, the precise message format is not fixed by design. For instance the decision for fixed or variable length messages can be taken at a later stage in the design process.

The *Router* assembles outgoing messages, sends these messages to the NS and handles incoming messages. The data of the messages generally comes from and goes to the local memory. The Router interacts with the memory without intervention from the CP. The *Route Generator*, a sub-unit of the Router, is a logic unit that generates the d node disjoint routes of a Kautz graph, given a source and a destination node (see section 2). If the NS of the source reports that a message did not reach the destination (due to congestion or link/node failures), the Router reads a new route from the Route Generator and assembles a new message. If all node disjoint paths have been tried the CP is informed. The CP can decide to try again later after a random delay [Whobrey 88].

2. Kautz networks

We use Kautz networks in our project because these networks have interesting properties [Bermond 89]. Particularly, they interconnect considerably more nodes than the usual topologies, and they have a small diameter, and a small and fixed degree. Furthermore they are highly fault tolerant, admit selfrouting and can embed standard computation graphs. Imase [Imase 86] showed the existence of d node disjoint paths between any pair of nodes in a Kautz graph of $N = d^k + d^{k-1}$ nodes. These properties makes Kautz graphs suitable as an interconnection network for large scale parallel computer systems.

Definition of Kautz graphs [Kautz 68].

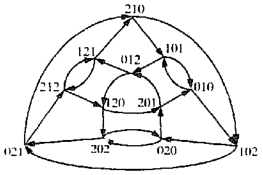

Fig. 1: Example of a Kautz graph (K(2,3)).

The Kautz digraph K(d,k) with in-degree and out-degree d and diameter k is the digraph whose vertices are labelled with words $(x_1,...,x_k)$ of length k from an alphabet of d+1 letters by removing those words in which there are two consecutive identical letters ($x_i \neq x_{i+1}$, for $1 \leq i \leq k-1$). There is 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.
A straightforward generic route of length k can be found by simple concatenation of source and destination word. However there may be routes with length < k [Smit 91a].

Example 1 (see fig. 1)

In the graph we find the route Rg = < 120201 > from (120) to (201) via node (202) and (020). This route has length 3 (= k).
The shortest route is Rs = < 1201 > of length 1.

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.
The Router Generator as described in [Smit 91b] generates the d node disjoint routes with increasing length . The routes are as short as possible and free of loops.

3. The Network Switch

A designer of communication systems for multi-computers is faced with conflicting demands, due to varying application requirements, fast changing technology, etc. Experiences with existing parallel machines and simulations have shown for instance that not one single routing mechanism is optimal for all kinds of applications. In the case of low or medium communication traffic, circuit switching or worm-hole routing seem to have advantages over store-and-forwarding. But in case of intensive communication as for instance due to frequent broadcasting, the store-and-forward mechanism is more adequate [Seidel 89]. If a communication network is designed with full-custom VLSI components, the designer is forced to make crucial decisions early in the design process.

In our design the Network Switch and Router are implemented with Field Programmable Gate Array (FPGA) technology. This technology allows the gate arrays to be reprogrammed for an unlimited number of times. Therefore they are suited for designs in which the functions of the hardware needs adaptations in order to meet changing application requirements. This has a number of advantages, such as:

- The selection of the most suitable communication mechanism can be postponed to a later stage of the design.
- The system designer or application programmer can 'esign application specific communication primitives and mechanisms. Knowledge of the communication structure of the computation can be used to tune the network to the requirements of the computation on-the-fly.
- Due to its flexibility the system can be used in a wide variety of applications, ranging from high speed computations to dedicated real-time applications.
- The design cycle of FPGAs is very short. Minor design changes can be made instantaneously.
- The cost of a prototype is very low. FPGAs are standard components, no design costs for full-custom components.

In this paragraph we present a possible Network Switch configuration. The most important design decisions are:

- *Uni-directional* links of 12 bits wide.
- We have chosen a *fixed network topology* and a fixed number of wires for each link because we cannot change the physical wiring of the system dynamically.
- We use *worm-hole routing* [Dally 87]. This type of routing suits well to routing in Kautz networks and gives a low latency. As soon as a Network Switch reads the header of a message, containing the route information, it

selects the next link on the route and forwards the remaining part of the message down that link. Each NS consumes one byte of the route.

- To avoid deadlock we use the *nosy worms protocol* [Whobrey 88]. If a message is blocked it is recoiled to the source. Because a Kautz network has d node-disjoint routes we expect that the probability of a collision is acceptable.
- As the amount of memory in a FPGA is relatively small the store-and-forward routing mechanism seems less obvious. Although the local memory could be used as a bufferspace for store-and-forward routing. Store-and-forward has a latency that is proportional to the product of packet length and number of hops. So worm-hole routing or virtual cut-through are more appropriate for our system.
- All Network Switches synchronize with each other via two synchronization signals per link.

A message consists of: a route field, a variable length data field and an End Of Data marker. The route field of the packet defines precisely the route the message takes from source to destination.
As soon as a NS reads the header of a message, containing the route information, it selects the next link on the route, "consumes" the used route information and forwards the remaining part of the message down that link. However, if the output link is occupied it returns a negative acknowledge (NACK). Each NS between source and destination consumes one byte of the route until the message arrives at the NS of the destination. Now the route field of the message is empty and the message is delivered at the Router of the destination. The end of the data is indicated with an End Of Data (EOD) mark.
We assume that the Router of the destination never refuses an incoming message forever.

If the Router of the source receives a NACK (due to congestion or link/node failures), the Router assembles a new message with another node disjoint route. If all node disjoint paths have been tried, the CP is notified.

4. Implementation

Fig. 2 shows the internal structure of the data-path of the Network Switch. The switch has 3 input and 3 output links. One of the input links and one of the output links is connected to the Router. With this switch several physical networks with in-degree and out-degree 2 can be built, such as: torus, mesh, deBruijn networks and Kautz networks. A link

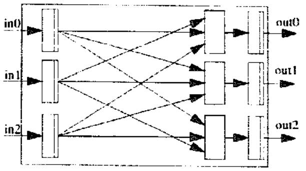

Fig. 2: Internal structure of the Network Switch.

consists of the following 12 uni-directional signals: 8 data bits, 1 type bit, a NACK signal and 2 synchronization signals (cl_i and cla_i). The type bit is used to indicate the start of a message and the end of the message (EOD). The NACK signal goes in the direction opposite to the data.

In the presented system we use an externally asynchronous and internally clocked design methodology.

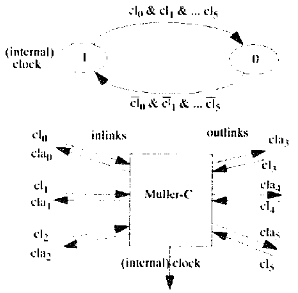

Fig. 3: State diagram and block diagram of a Muller-C element.

The design process of synchronous circuits is much easier compared to the design of asynchronous circuits. Moreover, the structure of FPGAs suits well to synchronous designs. However in large scale multi-computers there is a difficult and often underestimated problem of clock distribution. In our design we found a compromise that invokes the advantages of both the synchronous and asynchronous design methodologies, but without many of their disadvantages. From the perspective of the external world such a system operates asynchronously but its internal clock is driven following a specific handshake protocol [Tinder 91]. In this way the problem of clock skew can be avoided. The internal clock is generated by a Muller-C element, also called "Rendezvous Module". All internal registers are clocked on the rising edge of the *internal* clock. The synchronization works as follows: the Muller-C element of a switch receives the clock signals (cl_i) of all neighbor switches (see fig. 3). When all clock signals are asserted it negates the internal clock. This signal is also sent as an acknowledge (cla_i) to all its neighbors. If a C element has received all the acknowledges (all cl_i signals negated) it asserts the internal clock and so the cla_i signals again. To assure the correct synchronization of the switches all switches must respond to the assertion and negation of the cl_i signals of the neighbors even if they have nothing to send.

5. Realization

The above mentioned network switch configuration has been realized with a XC3042 Field Programmable Gate Array of Xilinx [Xilinx 91]. It uses a decentralized control, so six bytes can be handled simultaneously by a switch in one "clock cycle". Table 2 gives some results of the realization. The clock speed is derived from the output of the hardware simulator of Xilinx. The implementation was first described in VHDL and then simulated. After that the switch was automatically synthesized by the VHDL synthesizer from Viewlogic [Viewlogic 90]. Due to the high level specification in VHDL and the powerful synthesizer, the whole design process took only 3 weeks. We expect that the speed of the switch can be improved significantly by a careful (manual) redesign.

Number of CLB[1]'s used: 126	available: 144
Number of IO pins used: 73	available: 96
Number of logic levels used: 4	
Maximum "clock" speed: 10 Mhz.	
Transfer rate per link: 80 Mbit/sec.	

Table 2: Results of a realization with a XC3042 FPGA.

6. Conclusion

In this paper we have presented the design of a Network Switch for a multi-computer. It can efficiently support different styles of communication, such as worm-hole routing, store-and-forward routing and virtual cut-through. Each node of the multi-computer consists of three autonomous subsystems: the Computation Processor, the Router, and the Network Switch. The Router and the Network Switch are implemented with FPGA technology. This implies that the system designer can alter the communication mechanism, even in between the execution of two application programs. The communication network is based on a Kautz topology. Kautz graphs form a class of interconnection networks with interesting properties such as: small diameter, large number of nodes ($N = d^k + d^{k-1}$), the degree is independent of the network size, the network is fault-tolerant, it can embed standard computation graphs and has a simple routing algorithm.

The presented network switch uses worm-hole routing, and a simple deadlock avoidance protocol. Worm-hole routing suits well to routing in Kautz networks and gives a low latency. All Network Switches synchronize with each other, such that a global clock is not required.
Processors communicate with non neighboring nodes directly without involving the Computation Processor and Routers at the intervening nodes.

1. A Configurable Logic Block (CLB) contains programmable combinatorial logic and two storage registers.

References

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

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

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

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

[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 91a] Smit G.J.M., Havinga P.J.M., Jansen P.G.: "An algorithm for generating node disjoint routes in Kautz digraphs", Proceedings Fifth International Parallel Processing Symposium, pp. 102-107, May 1991.

[Smit 91b] Smit G.J.M., Havinga P.J.M., Jansen P.G., de Boer F., Molenkamp E.: "On hardware for generating routes in Kautz graphs", proceedings Euromicro91, 1991.

[Tinder 91] Tinder R.F.: "Digital Engineering Design, A Modern Approach", pp. 638-646, Prentice Hall, 1991.

[VHDL 87] "VHDL Language Reference Manual", IEEE-STD-1076-1987, IEEE Computer Society.

[VIEWlogic 90] "VHDL-Designer User's Guide", VIEWlogic Systems, Inc. April 1990.

[Whobrey 88] Whobrey D.: "A communications chip for multiprocessors", Proc. CONPAR 88 pp 464-473, 1988.

[Xilinx 91] "The Programmable Gate Array Data Book", Xilinx Inc., 1991.