Modular Design of Communication Node Prototypes

Sergio D'Angelo (*), Lauro Mantoani (**), Riccardo P. G. Mazzei (***), Stefania Russo (***), Giacomo R. Sechi (*)

(*)Istituto di Fisica Cosmica e Tecnologie Relative - Consiglio Nazionale delle Ricerche
via Bassini 15, 20133 Milan (Italy)
{sergio,giacomo}@ifctr.mi.cnr.it
phone: +39 - 2 - 23699 333/334
fax: +39 - 2 - 2362946

(**) Guest Researcher at IFCTR - CNR

(***) Universita' di Milano - Dip. di Fisica
via Celoria 16, 20133 Milan (Italy)

Abstract

A hardware rapid-prototyping system has been developed in order to experiment different communication networks for Massively Parallel Systems. Processing and communication in such systems should be kept, in our point, well separated both at abstract description and implementation levels. This results in a distinct physical implementation of Processing Elements (PE), relevant to computing tasks, and nodes for internode and node-PE communication. Rationales of this are briefly exposed.

Topic of this paper is the design of a macro library for FPGA devices. A proper combination of library blocks and some user-defined combinatory logic blocks makes it possible to implement the chosen network topology. High communication bandwidth can be achieved by subdividing the node workload among concurrent blocks. As an example, the design of a node for a (4,3)-WK recursive network - issuing wormhole point-to-point and broadcasting - is presented.

1. Separating communication and processing

Massively parallel computing systems gathered the interest of many people from industry, academy and research in the past years. Researchers and engineers paid efforts to set up many computers working together on problems of ever growing complexity to solve them faster than a single computer allows. The objective was that of building a powerful machine by connecting many computers, according to a prefixed topology, to obtain an individual parallel computer.

Our research mainstream in the context of parallel systems is concerned on the design of co-operating adaptive systems for the solutions of scientific problems, and in particular neural networks [1] and broadcast automata [2]. Both systems include a collection of identical processing elements (PE) each one using its own status and that of neighboring PEs to yield a new state: in the former case, PEs (the neurons) transform input values into an output status according a specific neural law, while broadcast automata are modelled as state automata. In this case, the system evolves through three phases: *state observation* involving communication, *computation* and *state transition*. In both cases, a communication network is to allow state exchange among PEs and should be designed as to perform global communication, namely broadcast and multicast.

It seems rather natural to conceive computation and communication as separated at this level. Nevertheless, in traditional parallel systems communication is handled by the same ingredients performing computation task. In other words, *nodes* of a parallel computer network are programmable machines performing both processing and communication tasks by means of adequate software. In our point, this results in a low system efficiency due to resource sharing at the node level; actually, communication is often a computation-intensive process even if techniques to access directly to memory are used.

Differently, the adoption of a communication network composed by nodes only devoted to routing tasks (Figure 1.), allows processing to run independently of (and

1066-1395/97 $10.00

concurrently with) communication, thus resulting in an improved system efficiency. Moreover, a so structured communication network could be, in principle, used with different kinds of PE, just re-programming the node-PE interface. Similarly, different communication network topologies could be implemented basically by re-arranging connections among nodes, thus reducing development costs and time-to-market. All these features can be obtained by an hardware FPGA-based prototype. The necessity of a hardware prototype, rather than a software simulation, must be also recognised in all the cases in which it is difficult, if not impossible, to simulate the operating environment, such as scientific experiments and hard real time applications.

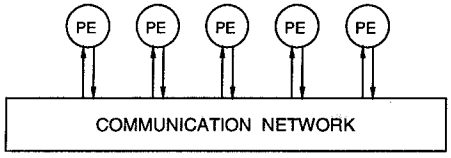

Figure 1. System Outline

Basic concepts on communication networks are summarised in the next section. In section 3 the architecture of a generic node is presented in order to identify the building blocks functionality.

Section 4 presents the building blocks architecture and the communication protocols defined. The design philosophy for the definition of communication nodes is also given. An example is shown in section 5.

2. General remarks on communication networks

Communication networks are made by a set of communication nodes (CN) connected according to a specific topology. Each node has at least two unidirectional channels with one PE. For the sake of simplicity we consider only the case of one PE connected to one node. A single communication can be point-to-point (single CN to single CN), multicast (single CN to some CNs) or broadcast (single CN to all the other CNs). A situation in which all CNs communicate in broadcasting is called gossiping.

For the sake of brevity we will assume all these concepts to be known, and we will only recall the most important keywords.

Several communication mechanisms can be devised; they can be subdivided into three main classes: *packet-switching*, *circuit-switching*, and *cut-through*. A special case of cut-through is *wormhole*.

Virtual channels can also be used, meaning that a physical link is shared among simultaneous messages. Deadlock avoidance techniques can be developed by the use of virtual channels [3]; they are based on link augmentation, so that resulting network has an acyclic dependency graph. With the use of FPGA virtual channels can be physically implemented (even more than 30 serial channels per chip), this yielding more performant communication nodes.

For a given topology, different routing algorithms can exist. Any communication mechanism can be adopted independently of the particular routing chosen, affecting only how messages are moved among and stored into nodes.

3. Communication nodes

Nodes for any kind of communication network are to implement some general features, independently of the particular topology adopted. Internal layout of a generic communication node with four inter-node links plus a link with a PE is sketched in Figure 2.

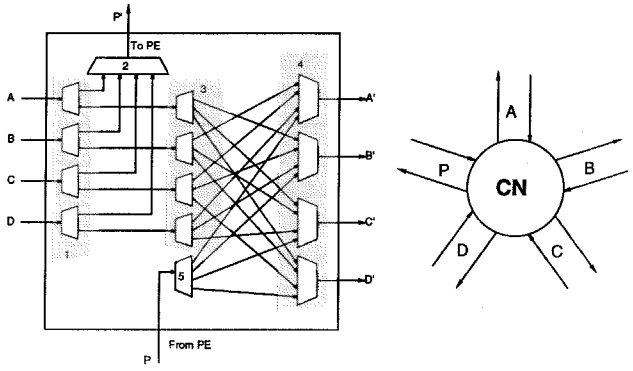

Figure 2. Generic node architecture

Input channels A, B, C, D from other nodes are internally routed through the shaded blocks 1, 3, 4, and blocks 2 and 5. P and P' are input/output channels from/to PE respectively. Note that, for any channel *i*, it does not exist a route to the corresponding *i'*, meaning that it is not allowed a message to be returned back to the sender.

Two basic activities are implemented: *path selection*, performed by some *Arbiter Blocks* (AB), and *path decoding*, which is relevant to *Router Blocks* (RB). In addition, *Interface Blocks* (IB) may be necessary to manage internode communication protocol.

If distributed routing algorithms are developed, RBs are to select one of the possible output channels only on the basis of message header and node address. Thus, in principle, any communication node based on a distributed routing algorithm can be rapidly implemented by endowing each RB of the generic node with an appropriate logic function. This particular implementation would not be the optimal solution for any node architecture, since simplification could be introduced in some cases thus reducing complexity and improving

performances. Anyhow, and this is the substance of the present work, by the use of the mentioned basic blocks the designer should be able to implement any particular node layout. Particular communication mechanisms can be devised by designing single blocks in an appropriate way.

4. The prototyping system

According to the concepts exposed in the previous section, a communication node can be viewed as being constituted by a certain number of basic blocks, such as ABs, RBs and IBs, connected together according to a particular scheme. The designer has therefore to define this interconnection scheme and to assign blocks activities. This can be done quite easily in our prototyping system. Each block is defined according to an architectural scheme, in which a number of sub-blocks is shown. Some of these represent fixed parts that are always used, independently of the node characteristics. Others have a pre-defined behavior and must be dimensioned (number of bits, number of links, etc.) depending on the particular application. A block layout is generally completed by some totally application-specific elements, which are simply combinatorial logic functions. However, most part of the design is developed simply by picking up a set of properly dimensioned sub-blocks from the library, whether at VHDL or schematic level. A macro library for FPGA design synthesis has been developed for the use with Powerview® and Workview® CAE environments [4,5].

Basic ABs, RBs and IBs has been modelled as sequential circuits communicating by a handshaking protocol; they are intended to work concurrently, allowing to extract potential parallelism from independent data paths.

Communication protocols and architectural descriptions of basic blocks are described in the following sections.

4.1 Communication protocols

In order to match different transmission requirements two communication protocols have been developed: an *asynchronous protocol,* that allows, in principle, communicating objects to run at different clock rates, and a *synchronised protocol,* in which the sender handles data transmission timing.

Asynchronous protocol waveforms are given in Figure 3a. The sender drives SEND, DAV (DAta Valid) and DATA lines, while the receiver acknowledges with FREE signal. As soon as the sender has available data, SEND first and then DAV are go high. A low-to-high transition on FREE indicates the receiver availability to acquire data. FREE goes low again when data are acquired; as long as FREE is low, data are forwarded through the receiver. When FREE goes low the sender releases DAV which remains low until new data are ready to be transmitted. The communication terminates when SEND is set to low by the sender. DATA width can be set by the designer, depending on the application.

In synchronised protocol (Fig. 3b), data transmission is made synchronous with the SYNC signal which is driven by the sender and serves as a sort of communication clock. DATA is set by the sender at each SYNC rising edge and is latched by the receiver at each falling edge. EOM (End Of Message) and RTS (Ready To Send) signals have the same meaning of SEND and DAV in the previous case.

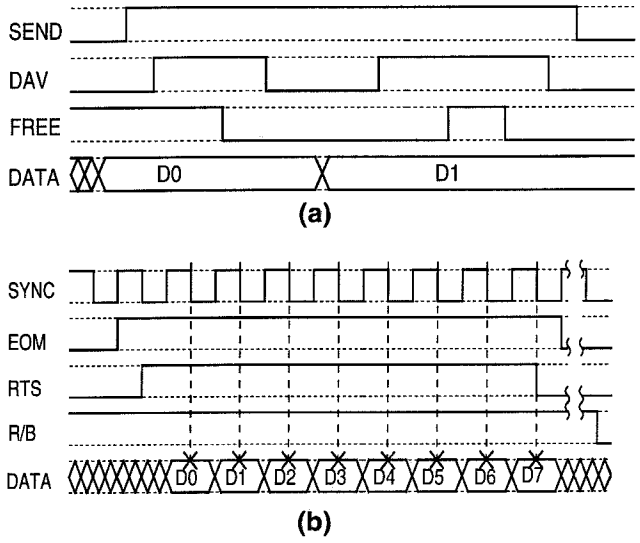

Figure 3. (a) Asynchronous protocol (b) Synchronised protocol

The R/B (Ready/Busy) signal, driven by the receiver, goes high when the receiver is ready to accept data and is held high during the whole data burst transmission. At the end of transmission of a burst, RTS is set to low. This cause the receiver to set R/B low and hold this value until data are routed. EOM allows to send multiple data bursts.

4.2 Communication basic blocks

4.2.1 Interface Blocks

Interface among internal node blocks, as well as node-to-node and node-PE interfaces, are implemented by dedicated interface blocks. Implementing a node by real devices requires, in general, the adoption of a serial internode an node-PE communication, due to relatively limited number of I/O pins with respect to the number of links desired. To face with such a general case, some I/O IBs have been defined to adapt serial data stream to

internal parallel stream which can be easily implemented inside FPGAs thanks to the amount of resources available.

IBs also store message header. They are constituted by an interface manager, devoted to perform protocol conversion, and a memory block, which is driven by the manager according to the particular protocol conversion. In order to implement pipelined communication, such as cut-through, addition of external FIFOs can be envisioned, while internal storage can be dimensioned as to hold either the only message header (wormhole routing) or entire message (store-and-forward).

IB latency l for a single flit is expressed by the relation:

$$l = \left\lceil \frac{n_{out}}{n_{in}} \right\rceil \qquad (+1 \quad for \quad header \quad flit)$$

where n_{out} is output data width and n_{in} is input data width.

4.2.2 Arbiter Blocks

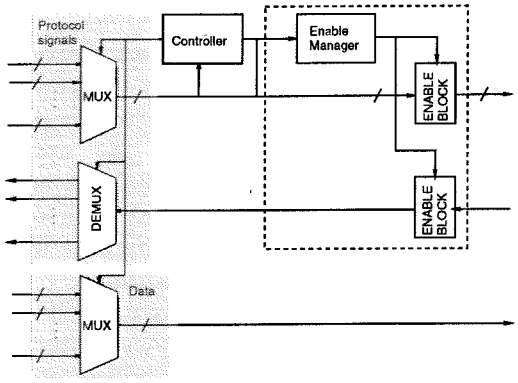

Figure 4. AB internal structure

Arbiter blocks are to generate control signals for both routing blocks and interface blocks, according to communication requests from either PEs or other nodes. The *controller* sub-block (see Figure 4.) polls on the n input request. As soon as a request is detected, the controller terminates his search and keeps the relevant paths selected during the whole communication. Data move along such selected paths towards the subsequent block, whether routing block or interface block depending on the case. The number of polling steps (AB latency), as well as mux/demux size, depends on the number n of inputs.

The part within the dashed lines is used only to connect the AB to an IB. In this case message header propagation through the node is delayed by one more clock step.

4.2.3 Routing Blocks

Routing blocks select the right data paths inside the node, depending on the node address and message header. The main sub-block is a set of application-specific logic functions (*Logic Block* in Figure 5.) that control mux/demux selection. Logic Block operates upon completion of message header acquisition, then selected paths are held for the whole communication, that is, as long as SEND (EOM) is high.

A *Broadcast Manager* and a *Broadcast Mux* (BMux) are used when a message must be forwarded to multiple output lines (typically in the case of broadcasting). In the case of synchronised protocol the R/B signal exiting from BMux must be interpreted as the availability of all lines to receive data. This enables data to be simultaneously sent through the involved channels. In the case of asynchronous protocol, the signal generated by the BMux reports whether all receiving blocks have been served and is used as FREE signal.

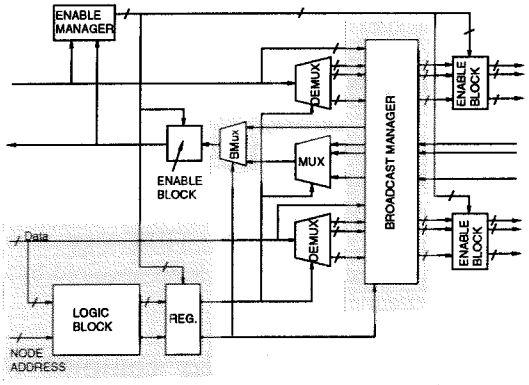

Figure 5. RB internal structure

Multiple replication patterns can be specified simply by adding more Broadcast Managers, each controlled by mutually exclusive broadcast signals.

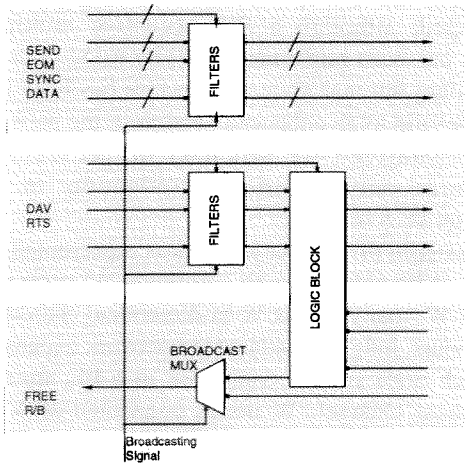

Figure 6. Broadcast manager description

Output lines are enabled, according to adopted protocol, with a clock pulse delay starting from the raising edge of SEND (EOM) signal; this ensures the correct path

selection stability when the communication starts and represents actually the block latency affecting only the header flit.

Broadcast Manager layout is given in Figure 6. *Filters* (Figure 7a and 7b) allow some lines to be disabled or simultaneously activated during broadcasting.

The *Logic Block* generates the FREE (R/B) signal which is handled by the server according to the communication protocol adopted. In broadcasting, FREE (R/B) of all the involved lines are checked at the same time.

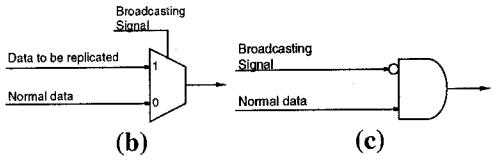

Figure 7. (a) Activating Filter (b) Inhibiting Filter

In the case of synchronised protocol the simultaneous activation of R/B signals tells the sender that receivers are ready to acquire a further data packet; if asynchronous protocol is chosen, simultaneous FREE signals indicate that the current message flit has been acquired by all the receivers.

5. An example: (4,3)-WKr communication node

This communication network model was presented in [6], and it was favourably accepted, thanks to its simplicity and attractive regularity properties. Nevertheless, no remarkable application has been developed, although a WK recursive multiprocessor system has been implemented [7] by using transputers. A convenient internal node architecture has been presented [8], referring to an enhanced version of the network itself. In this paragraph we will refer to this literature and, therefore, only a brief introduction to network characteristics is given.

A sample (4,2)-WKr network is represented in Figure 8. In a WKr network with amplitude W each node is of degree W, that is it has W bi-directional channel with its neighbours, independently of the network expansion level.

Each node is also connected to a PE, through a bi-directional channel. W nodes connected as a complete connected graph form a *virtual node* of first level. Recursively, W virtual nodes of level n form a virtual node of level $n+1$. Each virtual node has W free links which can be connected to other nodes, thus giving to the network full scalability. Node names are assigned in a recursive fashion: chosen an "orientation", for each node a sequence of L W-ary digits $n_{L-1}...n_0$ is used, where n_k shows the position of the virtual node of level k the node belongs to, within the virtual node of level $k+1$. Here virtual nodes of level zero are identified with physical nodes. Identification numbers are also given to links: taken two nodes $a_{L-1}...a_0$ and $b_{L-1}...b_0$, the first digit b_k of the second node name, starting from the most significant position, which differs from the correspondent digit in the first name, is the link name for the node $a_{L-1}...a_0$. The correspondent digit a_k is the link name for the node $b_{L-1}...b_0$.

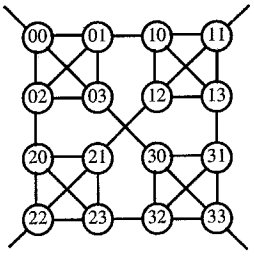

Figure 8. (4,2)-WKr Network

The basic routing of such a network derives simply from previous naming structures and it is done using only local information: if a message coming from a given node must be routed through node $a_{L-1}...a_0$ towards the destination node $d_{L-1}...d_0$, the link chosen for hopping is link d_k, where this digit represents the leftmost digit of the destination node name different from the corresponding digit in the actual node name.

Internal node structure, according to [8] is shown in Figure 9 (bold lines). It allows point-to-point communication according to the described routing. The communication paths shown allow any data path described by the basic routing and, moreover, messages can be forwarded through the node simultaneously in many cases.

Broadcasting is also possible; a distributed broadcast algorithm was presented in [9], but no reference is made to the node internal layout. Our node implementation performs broadcasting according to an equivalent algorithm [10], based on the transmission of a header containing source node address and a broadcasting flag. The routing algorithm is completely distributed, and has been implemented by adding two more RBs and paths (shaded blocks and lines).

Wormhole routing has been adopted, with one byte flits (header length). In order to reduce pinout, the node has synchronised serial interfaces to other nodes and a PE. Since each serial channel consists of 5 lines (SYNC, EOM, RTS, R/B, DATA) including one data bit and interface controls, the 18 channels of the node require a total of 90 pins. Internal paths are 8 bit wide and blocks communicate by an asynchronous protocol, yielding faster internal communication. Parallel In Serial Out (PISO) and Serial In Parallel Out (SIPO) IBs convert external serial

(synchronised) data into internal parallel (asynchronous) streams. Some ABs and RBs have been directly linked (A/RB blocks) as to implement fast crossbar switches, thus reducing memory requirements (number of FPGA flip-flops).

Implemented on a Xilinx 4010 [11] device, a single (4, 3)-WKr node occupied 80% of CLBs and 98 I/OBs (18 serial channels, global clock, master reset and node address). Using a 5 ns speed grade device, messages may run up to 30 Mbit/s on independent channels (single byte flit at no load).

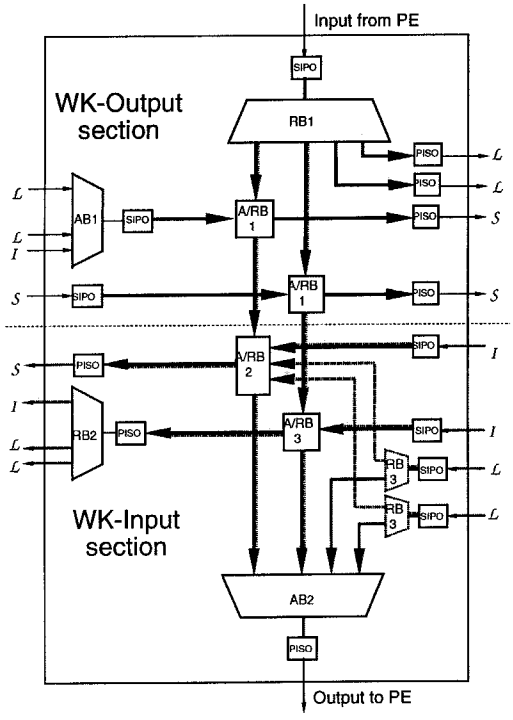

Figure 9. (4, 3)-WKr Internal node architecture

6. Conclusions

This paper was concerned on a particular subject of our research on parallel computational systems. Separating communication from processing came out as the most appropriate way of achieving high computation performances. A rapid prototyping system for communication nodes, making use of Xilinx FPGAs, was developed for the use with Powerview® and Workview® CAE tools. It is based on the subdivision of node design in concurrent blocks performing basic communication activities; the blocks are easily built and adapted to specific designs and needs, by the use of components from a dedicated macro library. By means of the CAE environment it should be also possible to retarget the design to other ASIC technologies, although this might require a further optimisation.

The application developed emphasised a high level of parallelism and a high channel bandwidth thanks to the internal blocks concurrence. The development system allows to exploit intrinsic parallelism due to the presence of independent paths in the node layout.

The future development of the prototyping system will include the definition of macro libraries for other Xilinx families besides X4000. New building blocks will be designed in order to endow communication nodes with dynamic reconfiguration capabilities. Fault tolerance techniques (A*n* codes, spatial redundancy, etc.) will also be included at the level of macro definition.

Applications will be concerned on the design of communication networks for broadcast automata and neural computers.

References

[1] M. Alderighi, S. D'Angelo, F.D'Ovidio, E. Gummati, G.R. Sechi "An Advanced Neural Model for Optimising the SIREN Network Architecture", Proc. of ICA^{3}PP '96, IEEE 2nd International Conference on Algorithms and Architectures for Parallel Processing, Singapore, June 11-13 1996 pp. 194-200

[2] M. Alderighi, R.P.G. Mazzei, G.R. Sechi, F. Tisato "Broadcast Automata: a Parallel Scalable Architecture for Prototypal Embedded Processors for Space Applications", to be published in Proc. of HICSS 30, 7-10 Jan 1997

[3] W.J. Dally and C.L. Sietz "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks", *IEEE Trans. Computers*, Vol. C-36, No. 5, May 1987, pp.547-553

[4] Powerview Reference Manual, VIEW*logic*

[5] Workview Reference Manual, VIEW*logic*

[6] G. Della Vecchia and C. Sanges, "Recursively Scalable Networks for Message Passing Architectures", Proc. Int. Conf. Parallel Processing and Applications (Sept. 23-25), L'Aquila, Italy.

[7] G. Della Vecchia and C. Sanges, "A Recursively Scalable Network VLSI Implementation", Future Generations Computer Systems, Vol. 4, No. 3, October 1988.

[8] A. Iazzetta, C. Sanges, U. Scafuri "A Routing Strategy for WK-Networks" Proc. of MPCS '94, The 1st International Conference on Massively Parallel Computing Systems, May 2-6 1994, Ischia, Italy, pp. 576-582

[9] G. Della Vecchia and C. Sanges, "An Optimised Broadcasting Technique for WK-Recursive Topologies", Future Generations Computer Systems, Vol. 5 No. 4 January, 1990.

[10] R.P.G Mazzei, S. Russo, G.R. Sechi "A Fully Distributed Broadcasting Algorithm for WK-Networks", Technical Report of IFCTR-CNR, No. T-001/96, 1996

[11] "The Programmable Logic Data Book", Xilinx, 1995