A Reconfigurable Processor Array with Routing LSIs and General Purpose DSPs

Jacob Levison, Ichiro Kuroda and Takao Nishitani
C&C Systems Research Laboratories, NEC Corporation
1-1, Miyazaki 4-chome, Miyamae-ku, Kawasaki 216, JAPAN

Abstract— A building block for a scalable signal processor array is developed with a general-purpose DSP and a message routing LSI. Each DSP can be connected by multiple routing LSIs forming a point-to-point message-passing network with data packet communication. Low network latency is obtained by cut-through routing technique with sufficient communication bandwidth. The employment of an on-chip routing table allows regular as well as irregular topologies with complex routing techniques such as broadcast/multi-casting and dynamic routing. The combination of DSPs (μPD77240), a flexible message-passing network and an optional application-specific I/O interface makes the processor array suitable for a wide range of high-speed signal processing applications such as adaptive array processing and 3-D vision processing.

1 Introduction

Traditionally, highly parallel real-time signal processing is dominated by dedicated parallel architectures such as the Systolic type of processing. Typical processing systems are dedicated to a specific application with a homogeneous data flow following a strict static scheduling. However, in order to reduce developing costs and time, an adaptable processor array is needed. Many processor arrays have been proposed. A processor array of DSPs is presented [1] and an already well-known DSP has been implemented in a version for parallel processing [2]. The Inmos transputer [3] is a compact building block targeting general purpose parallel computation. These three examples are all based on local communication between the processing elements rather than a global communication scheme covering the whole processor array. The iWarp system [4] and the Symult S2010 multicomputer [6] are examples of processor arrays where the communication between the processing elements are employing global routing techniques. iWarp is using a technique called "street sign" routing, while the mesh-routing chip (MRC) in S2010 is based on relative X-Y routing. However, processor arrays covering a wide range of applications require a flexible routing mechanism with broadcast/multi-cast capabilities. Depending on the application, different topologies should be employed which also require different routing algorithms for efficient communication. Although, routing algorithms are available for most regular topologies, there has not been much consideration for irregular topologies which requires routing tables [5]. Furthermore, integration of larger signal processing tasks in the same processor array requires handling of irregular and dynamic data flow.

A processor array with a reconfigurable architecture and a flexible routing mechanism has
been designed. Each node is composed of a DSP and a routing chip, and can also be provided with an application specific I/O interface. The number of processing nodes can vary depending on the application, therefore, giving a high degree of scalability.

Good cost-performance is obtained by employing 32 bit floating point digital signal processors (DSPs) μPD77240, as processing elements. The routing chip, also called Gate Chip, that interconnects the DSPs has been implemented in an LSI. The Gate Chip provides the processor array with a low-latency message-passing network. An on-chip routing table allows regular as well as irregular topologies. In addition, complex routing techniques such as broadcasting and dynamic routing is possible to realize.

This paper will describe the routing chip and its function in a processor array with emphasis on the communication mechanism of the message-passing network. Sections 2 and 3 give a general introduction to the processor array and the Gate Chip respectively. Hereafter, Section 4 describes the basic switching technique and derives the latency for the message-passing network. Then Section 5 explains the communication pattern in the processor array. Section 6 and 7 is two examples of applications, illustrating the performance of the processor array.

2 The Processor Array

Figure 1 shows an example of a 4 × 4 processor grid. The message-passing network in this processor array employs a router with four links currently implemented in an LSI. The LSI will be described in Section 3.

2.1 Node Processor and Their Interconnection

The processor array can be split into nodes, where each node can be divided into a permanent core and additional modules. The permanent core consists of a DSP μPD77240 as the processing element, a Gate Chip representing the message-passing network and finally instruction memory. Figure 2 shows a single node where the permanent modules are drawn with solid lines, while the additional modules are dashed. External data memory, a host-bus interface
and an application specific I/O interface are considered as additional modules since the size and capabilities depend heavily on the application. The host interface is used for down-load of programs, process control and monitoring of the processor array. The application specific I/O interface can be mounted on the node processor which enables easy re-targeting of the node processor.

While internal communication in each node between the DSP, the Gate Chip and the I/O interface is based on a synchronous clock, internode communication is asynchronous. Asynchronous communication for internode communication is chosen because of the difficulties of distributing the clock in larger processor arrays [7]. Asynchronous communication requires additional handshaking signals and gives an extra time delay, but problems with clock distribution are reduced. Another advantage of using asynchronous internode communication is the high degree of modularity. This gives a scalable system that can realize various multiprocessor topologies like one or two dimensional arrays or even irregular structures. Figure 3(a) shows a linear topology, where each node is interconnected with neighbour nodes via two buses which allows simultaneous data flow in both directions without collision. Figure 3(b) shows a two dimensional torus topology, which has a low latency due to the bus connections wrapped around the edges. Figure 3(c) gives an untraditional example of an irregular topology dedicated to a specific application. Furthermore, it is possible to realize three dimensional structures by regarding two DSP-nodes as a one 6-branch node as shown in Figure 3(d) & (e).

Since the processor array is highly flexible, the number of nodes, as well as the configuration of the message-passing network can easily be adapted from massively parallel processing to application specific processing. Various applications can be realized by the processor array such as real-time processing of multidimensional signals eg. multidimensional filtering, Fast Fourier Transform (FFT) and adaptive array processing.

2.2 Massively Parallel Processing

The maximum number of DSPs in one processing cluster is limited to 256, due to the size of the routing table in the Gate Chip, giving a peak computational power of 5.6 Gflops (floating point operations). Since each node in the processor array has individual I/O capabilities a high I/O bandwidth for the whole system can be obtained. For example high-speed input data can be demultiplexed and distributed to the nodes. After processing, the data from each node can be multiplexed to a high output data rate. The maximum peak I/O bandwidth, using 256 processors, is 1.4 Gwords or 5.6 Gbytes per second. If larger processing power is needed, it
Figure 3. Examples of multiprocessor topologies which are possible to implement using the Gate Chip as message-passing network. (a) One dimensional linear topology. (b) Two dimensional torus topology. (c) Irregular topology. (d) Six branch node realized by two DSP nodes. (e) Three dimensional structure.

is possible to connect several array clusters, thus the number of nodes exceeds more than 256 DSPs. Communication among the clusters will be accomplished through nodes operating like gateways.

2.3 Application Specific Processing

The specifications for the I/O interface, the external data memory and the host-interface varies from application to application. In cases where a relative small I/O bandwidth is required, not all nodes need to be equipped with an I/O interface. Data can be distributed from one node via the message passing network to other nodes which does not have an I/O interface. The amount of external data memory can also be adjusted, if only a small amount of data requires intensive computation, the external data memory can be omitted, since the DSP already has on-chip data memory. Furthermore, if the demand for communication with the host is small,
3 The Gate Chip

Figure 4 shows a block diagram of the Gate Chip. It consists of 6 main blocks, namely Central Control Unit and Bus System, DSP Port Element (DPE) and four Gate Chip Port Elements (GCPE), one for each of the directions North, South, East and West. The DPE is connected to the DSP via a 32 bit data bus, while the four GCPEs can be connected to the other Gate Chips via an 8 bit asynchronous data bus and 5 handshaking signals. The DPE serves as an interface to the DSP and contains two 64 words First-In First-Out (FIFO) buffers (32 bits), one for input and the other for output. The DSP is connected to the Gate Chip via two buses, a 32 bit wide bidirectional data bus and a 16 bit wide unidirectional address bus. Three additional signals, used for read, write and interrupt control link the Gate Chip and DSP together. The Gate Chip also contains a programmable interrupt controller which accomplishes an efficient interface between the DSP and the Gate Chip. The interrupt controller can be programmed to give interrupt in various situations, such as data collision, buffers full or empty. Furthermore, the DSP can retrieve status information about the Bus System and the Port Elements.

Asynchronous communication and handshaking with the neighboring Gate Chip are provided through the GCPEs which contains a 12 words FIFO buffer (8 bit). As the name indicates, the Central Control Unit & Bus System plays a significant role in Gate Chip by opening and closing the connections between the Port Elements. While the Bus System contains bus switches and logics for keeping status on which connections are active, the Central Control Unit is serving requests from the Port Elements and is routing the incoming messages. The Gate Chip has a lookup table, also called a routing table, that determines which direction(s) a message is going to be sent. The advantages of using a routing table is that it allows irregular topologies and multicast capabilities. Routing algorithms are already known for regular topologies such as the torus [5].
All the five bidirectional ports can be connected in an arbitrary pattern and communication paths in orthogonal directions can be established independent of each other. Figure 5 shows two examples of how the five ports in the Gate Chip can be connected, (a) one source transmits to four other receivers and (b) two simultaneous communication paths through the Gate Chip. The bandwidth of each port is 22.2 Mbytes/s which correspond to the I/O bandwidth for DSP which is 5.56 Mwords/s where one word is 32 bits. The Gate Chip is implemented in a 176 pins and 60 ktrans LSI using 1 μm CMOS technology. The maximum power consumption is 1.9 W.

4 Message Passing

The message-passing network serves as a communication link between the nodes in the processor array. Each processing element, in this case a DSP, is connected to a Gate Chip which serves as the router and takes care of the low level communication protocol such as packet generation, routing and handshaking with the neighbour Gate Chips.

4.1 Data Packet

Data which is sent between the nodes in the processor array is organized in packets as illustrated in Figure 6, where a packet consists of two parts, a header and a body. While the header has a fixed size of one 32 bit word, containing information about the destination of the packet, the type of packet, the message class and the return address (also called the ID of the sending node). The body contains the actual data and the size of the body can vary from one word to any number depending on the amount of data that is going to be sent. The packet type refers to the three different types of addresses, Single Ended, Symbolic and Neighbour which will be described in Section 5. The message class is an additional number which can be applied by the user to classify the data packets. The user can for example define different message classes: command, data, synchronization and error. By reading the class number in the header, the receiving DSP can determine how to interpret data in the received packet. The message class number can also be used for describing the size of the data packet.

4.2 Routing Technique

When a packet traverses through a processor array, the path which the packet will follow is determined by the information given in the Routing Table of each Gate Chip that the packet
The messages send between the Gate Chips is organized in packets. The latency for a given path between node \( X \) to \( Y \) is the delay for a packet to propagate from node \( X \) to \( Y \). The time for sending a packet of length \( L \) between two nodes is denoted by \( L/B \), where the channel bandwidth is \( B \) given in \( \text{word/s} \). By using cut-through routing, where the packet is forwarded direct through the node, the header must advance through each Gate Chip taking \( T_p \) seconds before the data can follow. Assuming that the packet is non local and therefore is passing through \( D \) nodes. The required transmission time for cut-through routing is:

\[
T_{ct} = T_p D + L/B
\]

Where \( D > 1 \). For the Gate Chip, \( T_p = 315 \text{ ns} \) and \( B = 5.56 \text{ Mwords/s} \) which is given from the hardware specifications. Calculation of the latency is essential for scheduling of algorithms for the processor array.

5 Communication Patterns

This section will explain how to send packets between nodes in a message-passing network using the Gate Chip as a routing chip. The three most basic addressing techniques are described in Section 5.1, Section 5.2 and Section 5.3. Section 5.4 explains how to define multiple paths and the use of dynamic routing.

5.1 Single Ended communication

In a processor array, where the Gate Chip is used for interprocessor communication, all nodes (Gate Chips) have an individual address (node ID), which must be given during the initialization of the system. When data is going to be sent from one DSP to another DSP, the sending DSP simply writes data to the address of the receiving DSP in the same way as writing to an address in the memory. The Gate Chip will then generate the header and send the packet in the direction given by the routing table. This type of communication will be called Single Ended Communication and is the most basic and simple addressing technique. Up to 256 single DSPs are directly addressable, but a higher number of DSPs can be achieved by using gateway
Examples of addressing, using the three different methods (a) Single ended, (b) Group and (c) Neighbour addressing.

5.2 Group Broadcasting

Besides the Single Ended communication the Gate Chip has two different broadcast techniques. The first one is Group Broadcasting where an arbitrary number of DSPs (less than 255) can be selected to be members of a group which can be reached by the same address. In general it is possible to define up to 128 different groups by using 128 symbolic addresses. But if the processor array consists of less than 256 nodes, the DSP addresses can also be applied for group broadcasting. The different groups can be overlapping which means that one DSP can be a member of up to 128 different groups. Figure 7(b) illustrates an example of Group Broadcasting, node8 sends to the group with members node0, -1, -3, and -4. The data packet is routed through node5 before it reaches the group members.

5.3 Neighbour Broadcasting

The second broadcasting technique is Neighbour Broadcasting which makes it possible to broadcast data to the surrounding neighbours. This gives 15 possible combinations, for example sending to north, north and south, north and east or north and west etc. Each combination has a specific address. Neighbour Broadcasting is a popular communication technique applied in for example Systolic implementation of algorithms. Figure 7(c) shows an example of Neighbour Broadcasting where node4 sends to node5 and -7.

5.4 Multiple paths and Dynamic Routing

As already mentioned in Section 4, it is possible to define more than one communication path between two nodes, but since each defined path requires an address in the routing table the number of paths is limited by the size of the routing table which is 384 in one cluster. Having more than one path between two nodes does not increase the bandwidth since the I/O bandwidth of the DSP is the same as the bandwidth of the path between two nodes. But multiple path makes it possible to realize dynamic routing without changing the content of the routing table during program execution.

The processor array shown in Figure 8 consists of 64 nodes. This means that $256 - 64 = 192$ addresses are vacant and can be used for alternative paths. Besides the 192 unused Single Ended
addresses the 128 Symbolic addresses can also be used for alternative path definitions. It is now assumed that each of the 64 nodes has four different communication paths to all the other 63 nodes. This means that each DSP besides its basic address has three so called alternative addresses, which also lead the packets to the DSP via a different path than the path defined by the basic address. This can be realized by using all 256 single ended addresses in the routing table. The example in Figure 8 shows the communication routes from node 3 to 28. The four communication paths correspond to four addresses. Address A goes via the nodes 11, 19 and 27. Address B goes via the nodes 4, 12 and 20. C goes via the nodes 4, 5, 13, 21 and 29. Finally, D goes via the nodes 2, 10, 18, 26 and 27.

One way to realize dynamic routing can be accomplished by defining multiple path and then let each DSP use different addresses depending on the traffic. Another way to realize dynamic routing is to change the content of the routing tables during actual program execution. This requires a careful scheduling and timing in order to control the location of the packets. The Gate Chip is also prepared for the last type of dynamic routing, since it allows access to the routing table even when the CCU is active. Collision is avoided by using a type of hardware semaphore.

6 Example 1; Matrix Multiplication

This section illustrates the performance of the processor array for a matrix multiplication which is a regular algorithm. It is demonstrated that the efficiency of the message-passing network allows maximum performance from the DSPs in the processor array.

6.1 Load Balancing

Many signal processing applications are based on regular types of algorithms such as matrix multiplication and QR decomposition. When mapping an algorithm to a processor array it is important to consider what is the maximum practical performance that can be achieved. This number depends on the performance of the processor, the algorithm and the number of processors. The DSP μPD77240 can execute two flops in one instruction cycle while one I/O cycle requires two instruction cycles. Thus, the ratio between number of flops and I/O operations is $\frac{\text{flos}}{\text{I/O}} = 4$. 

![Figure 8. Four paths between node 3 and node 28 are defined.](image)
For the algorithm, in this case a matrix multiplication, a ratio between required number of flops and I/O operations can also be derived. Assume that two regular matrices A and B, each with the dimension $n \times n$, is multiplied resulting in matrix C. The total number of flops will approximately be $n^2(2n)$ (Each element in C requires $2n - 1 \equiv 2n$ flops (multiply or add)). By partitioning the computation to $M$ processors, where $n/M$ is an integer number, the number of flops per processor is now reduced by a factor $M$ giving:

$$N_{flops, matrix}(n, M) = \frac{2n^3}{M}$$  \hspace{1cm} (2)

Assuming that each processor contains $n/M$ columns of matrix B and the matrix A is broadcasted to all processors, the results from each processor are $n^2/M$ elements of matrix C. The number of I/O operations per processor then becomes:

$$N_{I/O, matrix}(n, M) = n^2 + \frac{n^2}{M}$$  \hspace{1cm} (3)

The ratio between flops and I/O operations per processor is:

$$FIO_{matrix}(n, M) = \frac{N_{flops, matrix}}{N_{I/O, matrix}} = \frac{2n}{M + 1}$$  \hspace{1cm} (4)

In order to achieve maximum utilization of the DSP it is required that most time is spend on floating point operations, rather than I/O operations. Therefore, the $FIO_{matrix, p}$ should be greater or equal to the $FIO_{DSP}$. Under the condition that the DSP can receive and send data when needed. Similar optimizations can be made for other algorithms by finding the $FIO_{algorithm}$ ratio and fulfill the condition, $FIO_{algorithm} \geq FIO_{DSP}$.

### 6.2 Linear Array Implementation

Figure 9 shows an example of a matrix multiplication in a one-dimensional linear array with one unidirectional communication channel. The size of the matrix is $n = 16$ and number of processors are $M = 4$. Matrix B is partitioned into four sub-matrices each with $n/M = 4$ columns and $n = 16$ rows. One sub matrix is located in each node e.g. $B_1$ is located in node 1, $B_2$ in node 2 etc. Matrix A is divided into $n = 16$ messages where each message consist of a row in A. The group broadcasting scheme is used for sending each row of A to all four processing nodes. First is row1 sent, then row2 etc. The result matrix C will leave the processor array in rows. Each row is divided into four messages but sent consecutively as shown in Figure 10.

Figure 11 shows the scheduling of the process, where the status of each Port Element and the DSP is shown for the four nodes. When a row is broadcasted, it will due to the low latency arrive almost at the same time at the four nodes. Remark that the DSPs are not involved in the broadcasting, as it was local interprocessor communication. The DSP Port Element (DPE), East Port Element (E.PE) and West Port Element (W.PE) in each node receives one row in the period $T_0$. The DSP will start processing immediately when the first word arrives. In the shown example, the results will be stored in the internal memory until processing of the following row. This is done in order to multiplex the results. The numbers in the output boxes, shown in Figure 11, correspond to the node number and the row number, e.g. When node 1 reads and processes row 1, it will also send the results for the previous processing, row 16 (1,16).
Figure 10. Each row in matrix A is sent as a message. Matrix B is divided into four sub matrices $B_1$, $B_2$, $B_3$ and $B_4$, $B_1$ is located in node 1, $B_2$ in node 2 etc. Each row in the result matrix C is divided into four messages, one from each node. The numbers in matrix C refers to message number in Figure 11.

Figure 11. Scheduling for a matrix multiplication in a linear processor array with $M = 4$ nodes. The dimension of the matrix is $n = 16$.

In this case, where only one channel is employed for input and output data, it is important to remark that the bandwidth of the path that goes through the four processing nodes also constrains the partitioning of the algorithm. In order to obtain maximum performance for each DSP, following condition must be fulfilled.

$$T_{\text{proc+io}} > T_{\text{in}} + M T_{\text{out}}$$  \hspace{1cm} (5)

Where $T_{\text{proc+io}}$ is the total processing and communication time for one row in the matrix, $T_{\text{in}}$ is the time for receiving one row input data. $T_{\text{out}}$ is the time for sending the results of one row-column multiplication. If the condition can not be fulfilled, an overhead caused by the channel bandwidth limitation will reduce the maximum performance of the DSPs. In this case maximum performance is obtained for all four DSPs, which means a total performance of $4 \times 22.2 \text{ MFlops} = 88.8 \text{ MFlops}$. The example shows that due to an efficient message-passing
network given by the routing LSI maximum performance from the DSP can be obtained.

7 Example: Acoustic Beamforming

Acoustic beamforming is an example of a system application which can be implemented by using the processor array. The example will be based on the frequency-domain approach which enables simultaneous computation of multiple beams. The partition of the application will be illustrated, mainly for a 2-D FFT which is the essential part of the beamforming. This section will be focusing on the computational requirements and the mapping rather than the design of the beamformer.

7.1 The Algorithm

Fundamental equations has been developed for the frequency domain beamformer [8] and will now briefly be explained. For a linear sensor array with $M$ sensors equally spaced with a distance $d$, the sensor signals are defined as $r(n, m)$ where $m$ is the space index $0 \leq m < M - 1$ and $n$ is the discrete time index $t = nT_s$. $T_s$ is the sampling frequency. The frequency domain beamformer output is written as:

$$B(k, s) = \frac{1}{M} \sum_{m=0}^{M-1} w(m) R(k, m) \exp\left(j \frac{2\pi km d \sin \theta_s}{NT_s c}\right)$$

(6)

$R(k, m)$ is the $N$-point DFT (Discrete Fourier Transform) of the sensor signals $r(n, m)$, $k$ is the frequency index where $f = k/(NT_s)$. $s$ is the steering direction index where $\theta_s$ is the beam steering direction. Finally, $c = 340 \text{ m/s}$ is the propagation speed for sound waves.

For $N = 2^p$, where $p$ is a positive integer, $R(k, m)$ can efficiently be evaluated for $N$ values of $k$ by using the FFT (Fast Fourier Transform) algorithm over the time period $NT_s$ (for each sensor). Equation (6) can be rearranged to the form:

$$B(k, q) = \frac{1}{M} \sum_{m=0}^{M-1} w(m) R(k, m) \exp\left(j \frac{2\pi}{M} m q\right)$$

(7)

By substitution of:

$$\frac{2\pi}{M} m q = \frac{2\pi}{NT_s c} km d \sin \theta_s$$

(8)

Where $q$ is called the wavenumber index. Using FFT, Equation (7) can now be evaluated for $M$ values of $q$ (for each frequency index $k$). Thus, the beamformer output in equation (6) can be regarded as a 2-D FFT which transforms the time-space signal $r(n, m)$ into the frequency-wavenumber signal $B(k, q)$. Finally the time-domain beamformer output can be computed by taking a $N$-point IDFT (Inverse Discrete Fourier Transform) of equation (7). The whole frequency-domain beamformer is shown in Figure 12.

7.2 Computational Requirements

A complex $N$-point FFT needs $N \log_2 N$ complex multiply-and-add operations and the total number of operation per second is therefore:

$$P_{MA} = \frac{2MN \log_2 N + N M \log_2 M}{NT_s}$$

(9)
In order to keep the example simple, Equation (9) does not include overlapping FFTs. In a practical implementation this must be considered.

Assuming that $N = M = 32$ and the sampling frequency $f_s = 25$ kHz. The necessary computational power can be calculated from equation (9) giving $P_{MA} = 12$ Mega complex multiply-and-add operations per second. A complex multiply-and-add is equivalent to 8 flops giving a total of 96 $M$ flops.

7.3 Mapping and Scheduling

The $μP.D77240$ calculates one 32-point complex FFT in $T_{FFT,32} = 131 \mu s$ (including bit reverse) and the requirements needs 3 FFTs to be calculated per sample. In the theory, at least 10 DSPs must be employed in order to fulfill this demand.

The algorithm is mapped on a 2-D torus processor array with 16 nodes as depicted in Figure 3(b). In order to sample and distribute data from the 32 sensors, one extra node, called I/O node, is inserted between node0 and node3. Since the I/O bandwidth in this application is relatively low, only the I/O node is equipped with an I/O interface. The I/O node does not participate in the computations.

Each node will receive two rows of data from the I/O node. Node1 will for example receive $\{r(0,2), r(0,3), \ldots, r(N,2), r(N,3)\}$. When all nodes have received two rows of data, computation of the FFT of each row can start (time to frequency). After computation, all nodes must swap data (corner turning) before the next FFT can be evaluated (space to wavenumber). Hereafter, all nodes must again swap data before the final FFT (IDFT) transforms the data into time-domain beamformer output. Figure 13 shows the scheduling of the frequency-domain beamformer algorithm for node0-15 and the required amount of processing and communication time for the different time intervals will now be calculated.

First, sensor data must be distributed from the I/O node to the 16 processing nodes. The data consists of 32 packets each containing 32 real samples.
\[
T_{\text{distribute}} = N^2 T_{\text{DSP}} \text{IO} = 184 \text{ \mu s}
\]

(10)

Where \( T_{\text{DSP}} \text{IO} = 180 \text{ ns} \) is one DSP I/O cycle. The computation time for the two FFTs that each node must compute is:

\[
T_{\text{FFT}} = \frac{N}{L} T_{\text{FFT,32}} = 262 \text{ \mu s}
\]

(11)

Due to the low latency torus topology and the advanced routing technique in the Gate Chip, swapping of data among the 16 processing nodes can be done almost simultaneously. This means all nodes will either receive or send data in the time period:

\[
T_{\text{swap}} = 2LST_{\text{DSP}} \text{IO} = 46 \text{ \mu s}
\]

(12)

Where \( L = 16 \) is number of nodes in processor array (The I/O node is not included since it does not take any part in the computations). \( S = 2(N/L)^2 = 8 \) is the packet size. The same data swapping would have taken \( T_{\text{swap, bus}} = L(L - 1)ST_{\text{DSP}} \text{IO} = 346 \text{ \mu s} \) in a shared bus type multiprocessor.

Finally, the I/O node collects beamformer output data from all nodes:

\[
T_{\text{out}} = N^2 T_{\text{DSP}} \text{IO} = 184 \text{ \mu s}
\]

(13)

The total time can now be calculated:

\[
T_{\text{total}} = T_{\text{distribute}} + 3T_{\text{FFT}} + 2T_{\text{swap}} + T_{\text{out}} = 1246 \text{ \mu s}
\]

(14)

Which gives a slight overhead since the available time is given as:

\[
T_{\text{avail}} = NT = 1280 \text{ \mu s}
\]

(15)

It has now been explained how an application, frequency-domain beamforming, can be implemented by building a processor array which fulfill the demands for computational power as well as I/O capabilities. The example shows that the combination of the general purpose DSP and the efficient message passing network, provided by the Gate Chip, creates a flexible processor array for various applications.

8 Conclusion

A flexible processing element for a scalable processor array has been developed. Good cost-performance is obtained by using general-purpose DSPs, \( \mu P D 77240 \) with routing chips. The routing chip, which is being implemented in an LSI, realizes an efficient message-passing network for a processor array with various topologies. By using the cut-through routing technique together with a low delay for the messages routing, low latency is achieved.
The on-chip routing table and asynchronous communication between the Gate Chips results in a scalable processor array which can form regular as well as irregular topologies. Moreover, advance routing techniques can be realized such as Single Ended Communication, Group Broadcasting, Neighbour Broadcasting, and Multipath Routing.

Example 1 shows that due to the low latency message-passing network and broadcast capability, efficient processing can be obtained. Example 2 illustrates how an application can be mapped on the processor array. The processing power and I/O capabilities of the processor array are matched to the requirements.

Future work will be concentrated on scheduling and mapping of applications for processor arrays implemented with the described building block. Implementation of a $4 \times 4$ mesh processor array is expected to be completed in the autumn 1992.

REFERENCES


