Invited Paper

Design Philosophy of a Data-Driven Processor: Q-p

HIROAKI TERADA, HIROAKI NISHIKAWA, KATSUHIKO ASADA*, TOSHIYA OKAMOTO, SOUICHI MIYATA**, HAJIME ASANO, TSUYOSHI TOKURA***, MASAHISA SHIMIZU, SHUJI HARA****, SHINJI KOMORI*****, KENJI SHIMA******

A bread-board prototype of a VLSI-oriented data-driven processor or Q-p (Queue-prototype), which can handle history-sensitive processes under consistency constraints, has been developed jointly by Osaka University and four VLSI manufacturers: Sharp Corporation, Matsushita Electric, Sanyo Electric and Mitsubishi Electric. Since the Q-p is specifically designed to be a one-chip functional element that is easily programmable to form various dedicated processing functions, particular design considerations were taken to realize high flow-rate data-stream processing capabilities. This paper first introduces major objectives of the Q-p development project and then presents the Q-p's basic language structure to demonstrate advantages of diagrammatical representation of programs. After that, this paper reports an execution control based on the data-driven scheme incorporating history-sensitivity that gives rise to a unique "flow-thru processing" concept in which all processing, data-transfer and storage functions are carried out highly-parallel by a packetized data-flow through the elastic pipeline processing stages with completely distributed controls. This paper also reports the hardware structure of the emulated Q-p which allows us to fully utilize the available devices on a VLSI chip by means of a self-timed elastic pipeline scheme.

1. Introduction

Many research projects on data-flow or data-driven processing systems have been carried out by various organizations in the search for highly-parallel processing of the computation intensive problems or large data-base oriented processes [1]-[6]. In contrast to these approaches in which many separate functional subsystems are interconnected by common busses or switching networks to form a complete data-flow/data-driven processor, in the Q-p development project [7], [8], a stand-alone or self-contained realization of the data-driven processing functions on a single VLSI chip has been pursued extensively. One-chip integration was essential in this project because the most likely application of the processor is as a functional component that is easily programmable to form various dedicated functions such as digital filters, communications protocol controllers, and real-time signal processing. With this approach, it is expected that the chip will evolve a large-scale multi-chip system by which number crunching problems or large intelligent-data-base manipulations can be handled efficiently.

In section 2, scope of the project is introduced briefly to give major objectives of the development. In section 3, software aspects of the project are described to demonstrate advantages of a diagrammatical representation of programs. Then, in section 4, execution control schemes in an emulator for a prototype Q-p is discussed. Finally, in section 5, novel hardware implementation principle or a "flow-thru processing" concept that makes one-chip integration possible is presented.

2. The scope and the progress of the development project

The Q-p development project was inaugurated in fiscal year 1981-1982 by a feasibility study on possible application of a data-driven processing concept for realization of a high-performance programmable functional-device on one-chip. Based on the affirmative conclusion of the feasibility study, extensive investigations on the basic language structure followed in fiscal year 1982-1983 since one of the most important objectives of the Q-p development project was to provide the users with a friendly programming environment for a wide range of applications in consumer product fields. The first language being chosen was an intermediate

*Department of Electronic Engineering, Faculty of Engineering Osaka University, 2-1 Yamada-Oka, Suita, Osaka, 565 Japan.
**Integrated Circuit Group, VLSI Development Laboratory Sharp Corporation, Tenri, Nara, 632 Japan.
***Corporate Engineering Division, Systems Research Laboratory Matsushita Electric Ind. Co., Ltd., Moriguchi, Osaka, 570 Japan.
****Research Center, Sanyo Electric Co., Ltd. Hirakata, Osaka, 573 Japan.
*****Dedicated Microprocessor Group, LSI R & D Laboratory Mitsubishi Electric Corporation, Itami, Hyogo, 664 Japan.
******Industrial Electronics Department, Product Development Laboratory Mitsubishi Electric Corporation, Amagasaki, Hyogo, 661 Japan.

language that lies between the data-driven machine and the user oriented ultra-high level specification languages. The intermediate language is named the Diagrammatical Data-Driven Language or D^3L [9]-[11] since it incorporates graphical form of program structure representation on VDT screens and employs data-driven execution rules including consistency constraints. After these provisional investigations, it was decided in 1983 to launch a 3-year development-project in which the prototype data-driven processor Q-p has been developed through two main versions.

The first experimental version [7] was a general purpose multi-microprocessor system consisting of 100 processing elements and was completed in fiscal year 1983-1984. The processing element employed in this system is basically a 16-bit micro-programmable single-board computer (SBC) based on the AMD29xx chips and is provided with improved input/output capabilities to enhance processor to processor communications within the general-purpose multi-processor system by which each of the functional modules for the execution control is simulated by a program on a SBC and the module interconnection is emulated by interconnecting the SBC's in order to have the same topology as the target execution control scheme. The experimental system was expanded to comprise 220 processing elements in fiscal year 1984-1985 and has served as a powerful simulation/emulation tool in investigating possible data-driven processor architectures and hardware configurations.

As a consequence of comprehensive evaluation on various data-driven schemes using the multi-processor system, an experimental bread-board emulator, or E(V-2.0) [7], [8], for a data-driven processor incorporating a novel "flow-thru processing" concept was developed in fiscal year 1984-1985. Since another important target of the development project was the search for the best hardware structure to implement the D^3L functions on one-chip, particular attention was paid to exploit VLSI capabilities extensively. To achieve this goal, we have introduced a new hardware design concept that permit us the best use of available active device count on a VLSI chip with the least deterioration on their intrinsic gain in switching speed.

Finally, in fiscal year 1985-1986, an emulator E(V-2.1) comprising 4 sets of bread-board prototype of the one-chip data-driven processor Q-p [8] was completed successfully with many improvements over the first experimental model or E(V-2.0) developed the previous year. This prototype processor is nicknamed the Q-p or Queue-prototype and is reported in detail in the following sections.

3. Software structure of the Q-p

As mentioned earlier, the Q-p development was started from the study of an intermediate language D^3L [9]-[11] originally developed by Osaka University. The D^3L belongs basically to a class of data-driven languages. However, unlike conventional pure dataflow languages [3], [4], it was expanded to have a capability for describing generalized data-base manipulations and control functions including consistency constraints to assure coherent parallel-processing of history-sensitive processes.

The D^3L is an intermediate language because it is intended to link very high-level specifications and executable physical objects. For very high-level specifications, signal-flow graph and state-transition diagram representations are being investigated. Transformation of such descriptions into the D^3L representation can be carried out automatically since the D^3L is designed to preserve the structures contained in those specifications.

In this section, the D^3L structure and its implementation in the Q-p project are explained briefly. The transformation algorithm from the very high-level specifications to D^3L and its implementation will be reported elsewhere in the near future.

3.1 The basic structure of the D^3L

The D^3L program is represented by an augmented data-flow graph representation in which basic constructs of the graph are limited to three basic shemes shown in Fig. 1 and ordinary concatenation of these three constructs. The nodes in the diagram represent primitive processing functions or processes [9], [10] in which processing functions are defined by the basic scheme. The arcs indicate data dependency from one node to another where a node is either a primitive or a process represented by another data-flow diagram. In the program constructs employed in the D^3L, the firing rules of the node are just the same as those of a pure data-flow scheme as far as the token flow is concerned [9]-[11]. Therefore, as is usual in the data-flow or data-driven systems, it can also be proven by the Petri net approach that a data-driven-program written by any com-

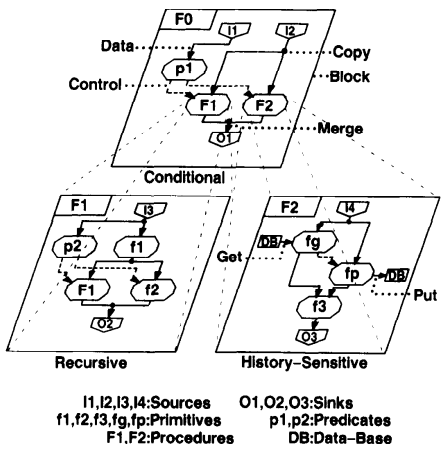

Fig. 1 Hierarchical Structures and Basic Constructs in the D^3L.

bination of these three basic schemes is syntactically valid.

The execution of a D^3L program is entirely done in a ready-to-fire principle or data-driven basis including the Get/Put operations on the data-base and the input arcs controlling the reference/update of the data-bases without introducing inconsistency.

The hierarchical nature and the graphical representation incorporated in D^3L constructs allows one to write a fairly complex program structure in a top-down fashion with a high degree of understandability or readability.

3.2 History-sensitive processing structure in D^3L

In this section, the implementation of data-base operations including the means for assuring the consistency of the data-base is described. By "consistency", it is meant that certain complex operations must be viewed as if they are atomic. That is, once an atomic operation is begun it must either complete its full updating without any updates accessing to its input variables during execution, or it must terminate without making any changes in the target data-base. That is, execution appears as if the atomic operation has exclusive use of the machine once the operation has started and until it has terminated.

The generalized data-base operation function in D^3L is controlled by the reference/update counts and by an additional control arc that indicates the constraint between the reference and the update operations on the data-base or history-sensitive part of the program. The reference/update counts are determined by static analysis of the program during the compilation/transformation phase and keep track of the data-base reference/update or Get/Put operations.

The reference and update counts are dynamically set to a predetermined value each time a program creates a new data-base and they are decremented by 1 if any reference or update to the data-base occurs during execution. The reference/update constraint control arc serves as a dynamic control input to the data-base updating processes to maintain the coherency of the data-base throughout the program execution. Therefore, the arc prevents illegal updating before all the authorized reference to an instance of the data-base finishes or before the reference count is decremented to zero [9], [10].

3.3 UL2: A practical version of the D^3L

In the Q-p development project, the D^3L specification has been expanded to improve its capabilities. Thus, a more practical intermediate language UL2 [7] shown in Fig. 2 was finally established as an intermediate language to be mapped onto an executable form on the emulated Q-p.

In UL2, some conventions such as numerical expressions and a practical set of history-sensitive data types have been introduced. The numerical expression must

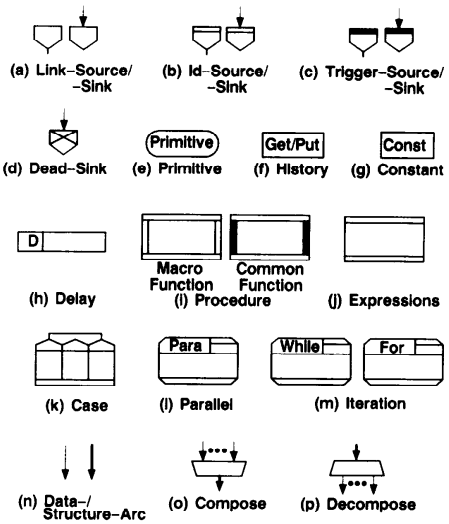

Fig. 2 Diagrammatical Symbols in the UL2.

be written in accordance with single assignment rules. As for the history-sensitive data types, they are either individual words or arrays. The array representation has two subclasses (the memory and the array) in relation to the generation and color control procedure that will be introduced later. Other examples of the convention added to UL2 are the iteration block (While and For), and the common and parallel expansion blocks.

The iteration block is a UL2 representation of a loop in conventional serial programs. Therefore, an existing loop program can be converted into its UL2 counterpart simply by rewriting the original program in accordance with single assignment rules. The common block is introduced to provide a means for easy conversion of existing library programs. The parallel expansion block facilitates parallel execution of existing programs by simply changing original program notations if any implicit parallelism exists in the original program.

The conventions introduced in UL2 have simplified considerably the conversion of many existing serial programs to parallel descriptions by providing a methodology to convert directly from serial form. This greatly facilitated the early stages of programming work on the Q-p. A programming example is shown in Fig. 3. A parallel expansion block containing array data operations is displayed.

4. Execution control functions in the Q-p

A dynamic mode of data-driven execution of the UL2 is introduced in the Q-p to enhance data-stream processing capability of the processor. In this section, the notions of generation and color are introduced briefly to show their roles in the dynamic processing of data-stream.

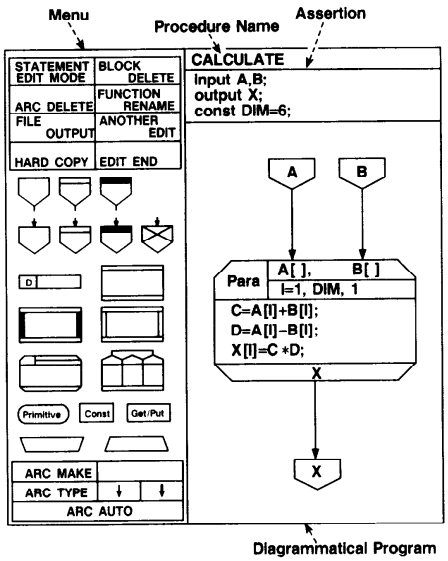

Fig. 3 A Sample UL2 Program.

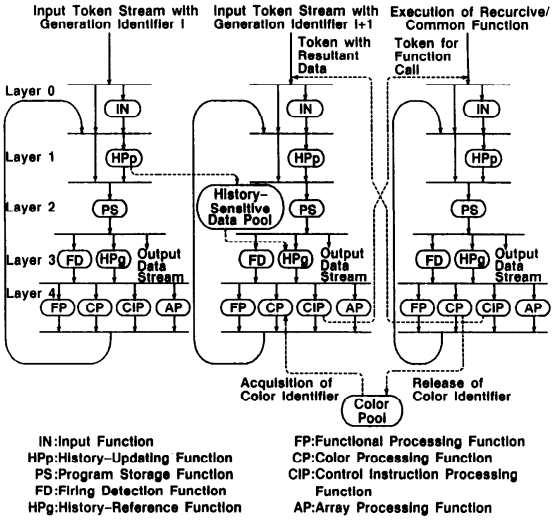

Fig. 4 Dynamic Data-Driven Execution Structure in the Q-p.

4.1 Dynamic execution of the UL2 on the Q-p

There are several alternatives to implement data-driven/data-flow languages on the hardware. Generally, the data-flow or data-driven processors are classified into two broader categories, dynamic and static [4], by the mode of holding the active data within the system. The Q-p adopted a dynamic mode since it is intended to process efficiently a succession of different data streams fed as one continuous stream. Therefore, in the Q-p, the concept of "generation" [9]-[11] is essential to discriminate the different set of sequentially incoming data sets.

The recursive call in D^3L, as shown by a procedure F1 in Fig. 1, was also adopted in UL2. Moreover, a common function call was introduced as one of the basic constructs in UL2 in order to save storage resources in the system. Therefore, it is necessary to distinguish different sets of data calling the same recursive processes or the same common functions simultaneously. These distinctions are done in the Q-p by the concept of "color" [2].

The concepts of generation and color have originated from different requirements. However, the net effect of these two identification tags is almost the same as far as the processing at a particular node is concerned.

4.2 Execution structure on the Q-p

Execution of UL2 on the Q-p is schematically divided into five layers: input (layer 0), history-updating process (layer 1), program fetch (layer 2), firing and history-reference (layer 3) and processing (layer 4) as shown in Fig. 4.

In layer 0, an external input or a resultant datum from previous operations is accepted. The external datum is assigned a generation at this layer when it is first fed into the processor. The datum is passed over to layer 1 if it requests the updating of any history-sensitive data, otherwise the datum is transferred to layer 2. When the datum processed so far arrives at layer 3, it is handled by firing detection function in this layer to produce a pair of data with the data already standing in layer 3 by association of the tags assigned to the data. Layer 3 is also responsible for manipulation of the reference to the history-sensitive data and the output functions. Finally, a set of data is passed to layer 4 where major processing on the data set will take place. The processing includes arithmetic/ logic, control instruction handling, color processing and reference to the array storage operations.

It is also shown conceptually in Fig. 4 how the notions of generation and color are related to the execution layers. As seen in the left-hand half of Fig. 4, the generation identifier keeps a new execution plane as long as it has been assigned to a set of data. The instance of a history-sensitive process is prohibited from being passed to the next execution plane or generation until all the reference/update in the previous plane has finished. The right-hand half of Fig. 4 indicates how the color is assigned and collected. That is, one of the color identifiers kept in a queue in the identifier pool is assigned to a set of data whenever the recursive or common procedure call occurs during execution. It is also shown in the figure that the color is released from the set of data and is reused under the control of the control instruction processing function (CIP).

5. Hardware structure of the Q-p

Since the Q-p is required to be a stand-alone programmable functional device, it is absolutely necessary to ex-

ploit VLSI capabilities. As a consequence of introduction of the "flow-thru processing" concept, a unique hardware structure was established and proved to be effective by experiments. The Q-p hardware system is basically composed of a chain of self-timed elastic data-transfer modules. The nickname of the processor is Q(Queue) and has originated from the extensive use of queues or self-timed elastic lines in the system. The underlying basic architectural and hardware design principles presented here are collectively called the "flow-thru processing" concept.

5.1 Execution structure of the emulated Q-p

As mentioned in the previous section, the data-driven execution of an instruction is represented by a token-flow through the layers as shown in Fig. 4. It is easily understood that the average data-flow rate through the execution loop dominates the processor performance. In a data-driven processor, the amount of tokens flowing through the loop is always fluctuating around an average flow-demand since it is influenced both by static factors like concurrency embedded in a program and dynamic factors like conditionals determined only at run time. Therefore, it can also be seen that the inter-layer buffering function and the distributed autonomous controls are key factors in maintaining a high average data-flow rate.

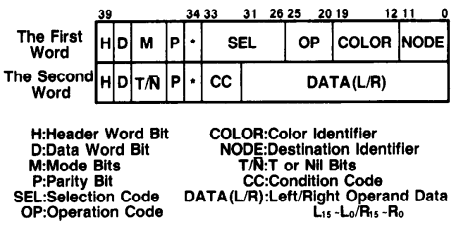

Fig. 5 Packet Format for the Q-p.

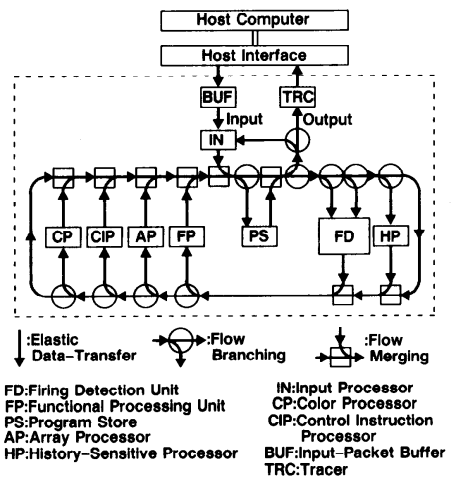

Fig. 6 An Example System Structure for the Q-p.

In order to realize a completely distributed control, all the tokens in the Q-p are represented by a packet as shown in Fig. 5. The routing along the execution loop in Fig. 4 is performed in each of the functional unit by modification of a selection-code in the packet header so as to direct a resultant packet to successor module(s). This means that the packet manipulations in the functional modules are organized as a completely self-contained process which is controlled solely by the contents of the packets it receives.

The structure of the emulated Q-p is designed as shown in Fig. 6 where a ring-shaped elastic data-transfer line connects all the functional modules. These processing and memory modules in Fig. 6 have a one-to-one correspondence with the functional representation in the execution control scheme shown in Fig. 4. Each of these basic processing and memory modules as well as basic data-transfer/branching/merging units are organized as an elastic pipeline structure as will be shown later.

This structure is particularly suitable for the experiments for analyzing packet behavior, because any overflow packets from the modules can be observed by monitoring the left- and right-most parts of the data-transfer ring. It is also advantageous in this organization that there are paths to all modules. Note that the ring has a tremendous total data-transfer capacity since the ring is shared among the processing modules in space as well as in time. That is, the data-transfer on the ring is time- and space-multiplexed at the same time.

5.2 Elementary data-transfer mechanism

The most elementary hardware of the Q-p structure is a self-timed elastic data-transfer mechanism as shown in Fig. 7. In this structure, the data-transfer through a chain of temporary registers (data latches) is controlled by a self-timed transfer control circuit (STC) in which the two-stage cascade connection of speed-independent coincidence memory circuits generates the transfer signal to the latches. With this simple control scheme, an elastic mode of data-transfer can be realized with completely distributed control without any system tim-

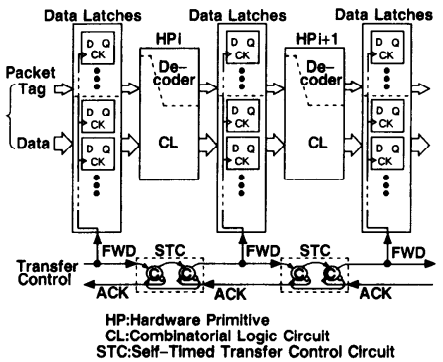

Fig. 7 Self-Timed Elastic Data-Transfer Mechanism.

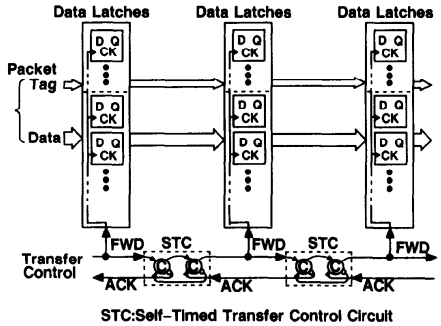

Fig. 8 Pipelined Processing Mechanism Based on the "Flow-Thru Processing" Concept.

ing clocks.

In the Q-p, a ring-shaped packet collecting and distributing mechanism is formed with the aid of selective branching and priority merging functions working between two independent data-transfer lines.

5.3 Elementary data-processing mechanism

A particularly important feature of the Q-p is that all the data-processing mechanisms are also realized by an elastically pipelined scheme using the same elementary self-timed elastic data-transfer mechanism.

As shown in Fig. 8, all of the data-processing units in the Q-p are implemented by inserting combinatorial logic circuits (CL) between adjacent data latches to form an elastic pipelined-processing function.

Since these two functions are implemented by using the same elastic data-transfer mode, the data-buffering function is located distributively throughout the pipeline stages in the processing units and the buffering connections in between the processing units.

5.4 Elementary hardware structure of the firing detection mechanism

Undoubtedly, the most important hardware function in a data-driven process is the mechanism to detect firing conditions efficiently. A novel associative mechanism for the firing detection function is introduced into the Q-p as shown in Fig. 9. The firing detection mechanism consists of a pair of elastic data-transfer rings that move the data packets in opposite directions to each other. The comparators are distributed between the two rings. A pair of packet streams is injected into the rings via merging junction on each ring. If there exists a matching between the two packets on each of the rings at a particular comparison stage, the data travelling through a line is diverted to the other line to form a data-pair. The diversion control is also greatly simplified by the elastic nature of the self-timed elastic data-transfer mechanism since the data-flow in the line can be controlled locally without knowing the overall status of the lines. The data-pair thus generated will be removed from the ring by a selective branching function inserted in the outer ring.

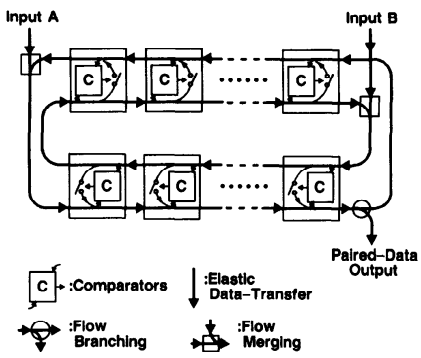

Fig. 9 Schematic of Firing Detection Mechanism.

The features of the firing detection mechanism can be summarized as follows:

1) highly-concurrent associative operations along the lines,

2) very high data flow-rate through the pairing processor,

3) automatic garbage collection facilities inherent in the elastic transfer lines, and

4) very high and constant pair-generation rate independent of the capacity of the lines.

5.5 Elementary organization of the memory module

It is obvious that the Q-p hardware structure is very efficient in storing active data since the active instructions as well as the data associated with the living tokens being processed are stored in the buffer distributed through the data-transfer and processing functions as well. However, a sizable amount of storage capacity is required in the processor for the program, array, constant and history-sensitive data. It is apparent that inactive data can be placed in more area-effective and power-saving storage such as linear-array memory structure.

Whereas, in the Q-p, it is also a requisite for the memories (PS, AP and HP) to function in accordance with the elastic mode of system operation. Therefore, the static memory of the Q-p is so designed to exhibit quasi-elastic behavior that it is able to function in conformity to the basic hardware-implementation concept, that is, conventional linear-array memory is combined with a self-timed elastic control mechanism by which an input packet is moved through the memory module in the same manner as the basic self-timed data-processing mechanism.

5.6 Significance of the data-driven processing principle in a pipeline environment

As emphasized above, one of the most salient features of the Q-p emulator hardware is that the "flow-thru processing" concept is extensively employed throughout the system to form an integrated elastic

pipeline-processing system.

Although the pipeline-processing scheme is popular in many data-processing systems, it has a particular significance in a data-driven processing environment since a pipeline stage can show its ultimate effectiveness with extremely simple control. This is due to basic nature of the data-driven processing where all the fired processes can be executed independent of any other processes being fired concurrently. This means that every processing function can be organized just to do simple unidirectional or straight-forward pipeline-processing without any interlocking or flushing with other processes in progress concurrently.

A unified elastic-pipeline implementation of data-transfer and processing scheme adopted in the Q-p emulator is expected to exhibit substantial speed gain when it is integrated on a chip because the scheme is realized with almost minimal wiring length to connect regular and orderly subdivided control logic. Note also that the scheme is highly efficient in chip area utilization since there is no space consumed by metallic passive-buses spanning across the chip.

6. Conclusion

The Q-p project has carried out the implementation of data-driven principles represented by a diagrammatical program as a one-chip stand-alone programmable functional device. The project also has developed the "flow-thru processing" concept which appears to be an excellent basis for the VLSI implementation of the data-driven processor. The major advantages of the design concept can be summarized as follows:

1) Very high degree of repeatability in the elementary hardware structure,
2) Very high usability of the available active device count on the chip,
3) Minimum extrinsic deterioration on the intrinsic operating speed of the device on the chip,
4) Ease in the logic design, and,
5) High testability.

Acknowledgements

Although it is impossible to give credit individually to all of those who organized and supported the Q-p development project, the authors gratefully acknowledge all of their colleagues on the project. The project would not have been so fruitful without their devotion.

The authors are very grateful to Dr. H. S. Stone of T. J. Watson Research Center, IBM for his efficient review of a draft for this paper and valuable comments.

The work described in this paper is supported in part by the Ministry of International Trade and Industry under Contracts 58KO-GI-SO-1008, 59KO-GI-SO-1560 and 60KO-GI-SO-1537.

References

1. DENNIS, J. B. The varieties of data flow computers, *Proc. of 1st Int'l Conf. on Distributed Computing Systems* (Oct. 1979), pp. 430–439.
2. ARVIND and KATHAIL, V. A multiple processor data flow machine that supports generalized procedures, *Proc. of 8th Symp. on Computer Architecture* (May 1981).
3. VEGDAHL, S. R. A survey of proposed architectures for the execution of functional languages, *IEEE Trans. Computers, C-33*, 12, (Dec. 1984), pp. 1050–1071.
4. SRINI, V. P. An architectural comparison of dataflow systems, *IEEE Computer*, **19**, (Mar. 1986), pp. 68–87.
5. AMAMIYA, M., TAKESUE, M., HASEGAWA, R. and MIKAMI, H. Implementation and evaluation of a list-processing-oriented data flow machine, *Proc. of 13th Ann. Int'l Symp. on Computer Architecture*, (1986), pp. 10–19.
6. HIRAKI, K., SEKIGUCHI, S. and SHIMADA, T. Maintenance architecture and its LSI implementation of a dataflow computer with a large number of processors, *Proc. of Int'l Conf. on Parallel Processing*, (1986), pp. 584–591.
7. NISHIKAWA, H., TERADA H KOMATSU, K., YOSHIDA, S., OKAMOTO, T., TSUJI, Y., TAKAKURA, S., TOKURA, T., NISHIKAWA, Y., HARA, S. and MEICHI, M. Architecture of a one-chip data-driven processor: Q-p, *Proc. of 16th Int'l Conf. on Parallel Processing* (Aug. 1987).
8. ASADA, K., TERADA, H., MATSUMOTO, S., MIYATA, S., ASANO, H., MIURA, H., SHIMIZU, M., KOMORI, S., FUKUHARA, T. and SHIMA, K. Hardware structure of a one-chip data-driven processor: Q-p, *Proc. of 16th Int'l Conf. on Parallel Processing* (Aug. 1987).
9. NISHIKAWA, H., ASADA, K. and TERADA, H. A decentralized controlled multi-processor system based on the data-driven scheme, *Proc. of 3rd Int'l Conf. on Distributed Computing Systems* (Oct. 1982).
10. NISHIKAWA, H., TERADA, H. and ASADA, K. A data-driven schema incorporating history sensitivity, *Trans. of IECEJ*, J66-D, 10, (Oct. 1983), pp. 1169–1176, in Japanese.
11. NISHIKAWA, H., ASADA, K. and TERADA, H. A data-driven execution control scheme and its experimental study, *Trans. of IECEJ*, J67-D, 5, (May 1984), pp. 607–614, in Japanese.

(Received November 4, 1987)