ARCHITECTURE DESIGN OF A FULLY ASYNCHRONOUS VLSI CHIP FOR DSP CUSTOM APPLICATIONS

Xingcha Fan and Neil Bergmann

CSIRO/Flinders Joint Research Center in Information Technology
School of Information Science and Technology
Flinders University
GPO Box 2100, Adelaide 5001, Australia

ABSTRACT

A fully asynchronous, distributed VLSI architecture is introduced for dedicated real-time Digital Signal Processing (DSP) applications. The architecture is based on a data-driven computing model to allow maximum exploitation of the fine-grained concurrency. An asynchronous, self-timed signaling protocol is used in the architecture to naturally match data-driven computing and circumvent the clock skew problem. After a brief description of the architecture, key issues of the architecture, such as the interconnection network, data identification, and operand matching are discussed. Finally, advantages of the architecture and future work are outlined.

1. INTRODUCTION

Advances in modern signal processing technology depend critically on the device and architecture innovations of computing hardware. Due to the severe system control overhead, general-purpose computers cannot offer satisfactory processing speed for most real-time digital signal processing(DSP) applications, and they are often an overkill in price and programmability for specific DSP applications. Therefore, application specific integrated circuits(ASICs) or dedicated VLSI implementations of DSP algorithms have been considered as the most appealing alternative when design costs allow.

The architectures of the most commonly used ASICs or dedicated DSP processors are either systolic array architectures [1], or centrally controlled multi-functional unit architectures [2], [3]. The locally interconnected systolic arrays maximize the strength of VLSI in terms of intensive and pipelined computing and yet circumvent its main limitation on communication. Their massive concurrency derived from the pipeline processing and/or parallel processing, makes systolic arrays very successful for those DSP applications with highly regular algorithms, but the strict regularity requirement for the algorithms limits their applications.

The centrally controlled multi-functional unit architecture, on the other hand, has been used to cover a much wider range of DSP applications with medium to high performance requirements. The common feature of this architecture is the utilization of multiple functional units to exploit concurrency by supporting pipeline processing and/or parallel processing. The interconnections between functional units are through buses and multiplexing. Resources are shared among operations, hence leading to an efficient hardware utilization. All the operations and data transfers are controlled in a predefined schedule by a central controller, which is usually a microcode sequencer. The control-flow computing model of this architecture limits the exploitation of the fine-grained parallelism in the algorithms.

Conventionally, these architectures both use the synchronous design methodology, i.e. the system is synchronized by a global clock signal. This becomes a limiting factor for VLSI system performance with the continuous growth of chip size and scaling down of the IC process technology, because of the difficulty of clock signal distribution and clock skew [4]. The synchronous design methodology also requires a large effort for the chip level layout design and timing simulation, and can limit the board level extensibility of the system because of the global timing constraint.

To circumvent these disadvantages, we introduce a fully asynchronous, self-timed distributed VLSI architecture for dedicated real-time DSP applications, especially those with irregular(e.g. data dependent) algorithms.

Asynchronous, self-timed techniques have been attracting a lot of attention in VLSI design in recent years. Several publications [4], [5], [6] about the design theory and automatic synthesis of self-timed logic circuits have appeared. Due to the handshaking character of the self-timed signaling [4], self-timed circuits are inherently slower than their synchronous counterparts at the gate level, and are usually more complex because of the extra circuitry needed for the self-timed signaling protocol. But self-timed signaling has the advantage of its event-driven character. Therefore, we believe our efforts to apply self-timed techniques to VLSI system level design will allow us to take full advantage of its potential, and make it a more competitive technique for VLSI design.

Asynchronous, self-timed signaling has been introduced into VLSI system level design by several researchers [7], [8], [9]. Sutherland's "Micropipeline" [8] concept is committed to those applications with data flow and control flow which are simple and have no feedback paths. On the other hand, Martin's asynchronous microprocessor [9] and the asynchronous DSP processor in [7] are both based on the conventional centrally controlled control-flow computing model. In contrast to previous research, the architecture we introduce here is a data-driven, distributed controlled multi-functional unit architecture.

The purpose of the this paper is to introduce such an architecture, and briefly discuss some key issues of the architecture and their implementations. The synthesis of VLSI chips based on this architecture from high-level descriptions is outside the scope of this paper. The paper is organized as follows. First, we present a general description of the architecture. Then we discuss the implementation of the interconnection network by using a multi-bus structure. Following a discussion of the data identification scheme, the matching block and I/O blocks will be briefly discussed. Finally, some conclusions and future work will be outlined.

2. ARCHITECTURE DESCRIPTION

The architecture is based on the data-driven computing model, instead of the conventional control-flow computing model. As a special case of dataflow computing, data-

0-7803-0593-0/92 $3.00 1992 IEEE

driven computing allows the maximum exploitation of the fine-grained concurrency inherent in an algorithm. A four-cycle self-timed signaling protocol [4] is enforced throughout the system to naturally match the data-driven computing. The DSP algorithms are represented by data flow graphs(DFG's) [1], in which nodes model computation(or logical operations), and arcs model communication.

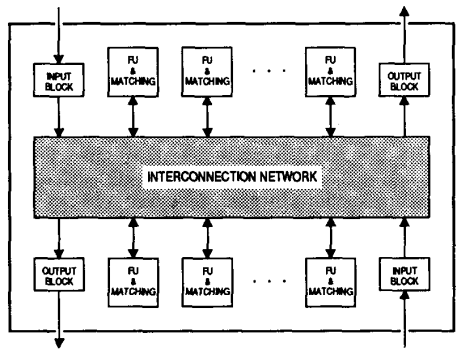

Figure 1: An asynchronous, distributed VLSI architecture

The architecture is shown in Fig.1. It is composed of multiple self-timed functional units(FU's), such as adders, multipliers, ALU's, and even some complex computation blocks. For example, a complex multiplication block can be designed and included as a functional unit for a FFT chip. These functional units are interconnected through an interconnection network which allows the concurrent transfer of multiple data items, or tokens. External communications are through the Input and Output blocks which are also connected to the interconnection network.

A functional unit can be shared by logical operations with the same function, which trades off the system throughput with chip size.

The execution of functional units, and the communication among them are controlled and coordinated by a decentralized, or distributed control scheme. A functional unit is fired in a data-driven manner, i.e. it will start the execution of a logical operation mapped to it as soon as all of the inputs for that operation are ready and the FU is free. The result of the execution together with a tag which identifies the data is sent out as a token. This token is self-routed through the interconnection network to the functional unit(s) that will consume it. The tag is generated and tagged locally by each functional unit. Therefore, the central controller, or control-path of conventional VLSI processor architecture is eliminated.

Because of functional unit sharing, the operands of logical operations shared on a functional unit may overtake each other and arrive at the two input ports of the functional unit in a different sequence due to the irregularity of the algorithm, and the data-dependent computation delay of self-timed functional units. Therefore, all the operand pairs are matched before being forwarded to an FU for execution through a local matching block. A matching block also produces tags for each logical operation result.

3. INTERCONNECTION NETWORK: MULTI-BUS STRUCTURE

VLSI systems are wire limited. With the continuous scaling down of IC process technology, the area occupied by interconnections becomes more significant relative to that of functional units, and the communication delays become a limiting factor of the system performance. Therefore, the structure and the implementation of the interconnection network in our architecture are very important to the performance and the size of the synthesised chips. Two performance metrics of an interconnection network are network latency and network throughput.

In recent years, regular, direct communication networks such as the hypercube [11], or dynamic multistage switching networks [10] have been commonly used to interconnect highly concurrent computers. The advantages of these communication networks are higher throughput, which allows many packets to travel concurrently through the network, and their extensibility, which allows the system interconnected by these networks to be extended arbitrarily.

On the other hand, the bus structure has often been criticized for its data transfer ability since it is limited to a single transaction at once. More importantly, it is limited in extensibility because of the large capacitance of long interconnection wires, especially off-chip interconnection wires. Nonetheless, the bus has the advantages of being simple and wire-efficient.

At this stage, our architecture is aimed for dedicated DSP applications which can be implemented with less than or around ten functional units on a single chip. In this special architecture, we believe the advantages of hypercubes or multistage switching networks cannot be fully exploited. Although the simulation data have yet to be obtained, we believe the data transfer latency of such networks would be larger than that of a bus, because with relatively low numbers of functional units, the delay of on-chip buses is not so significant, but significant switching delay must be counted for self-routing communication networks. Such a network occupies larger chip area than the buses due to the wires and the switching circuitry utilized. Therefore, for the current version of the architecture, we choose the multi-bus structure for on-chip interconnection. With the future upgrade of the architecture on which large number of functional units are implemented, pipelined buses, or high-dimension communication networks discussed above will be considered.

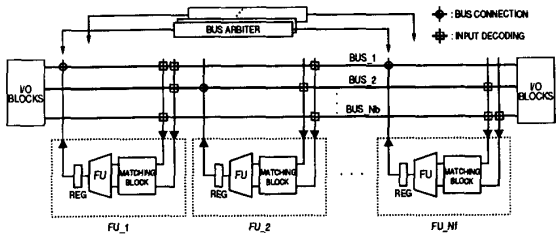

Figure 2: The architecture with multi-bus interconnection

With the multi-bus interconnection structure, the current version of our architecture is as shown in Fig.2. Each bus transfers the tokens in parallel. A token is delivered to its destination(s) through proper decoding of its tag. The number of buses used in the system is decided by the cost constraints and the performance requirements of the specific applications. The output of a functional unit is connected to only one bus, but the inputs of the functional unit can be obtained from different buses for different logical operations through proper connection and decoding. Buses are shared by functional units to reduce the number of buses required.

Because of the distributed character of the architecture, the functional units operate independently of each other. Operations and bus transfers are not assigned to any specific times, as in a synchronous architecture. Therefore, an

arbiter is necessary for a bus whenever more than one FU will possibly request the bus concurrently. The function of the bus arbiter is to ensure at most one bus request can be served at any time, i.e. only one data can be loaded to the bus at any time. Although a distributed bus arbiter is desired to match the distributed architecture, a fair arbiter constructed on a simple interlock circuit [4] is currently implemented for its simplicity and speed advantages [13].

The self-timed buses in an asynchronous architecture can be utilized more efficiently than their counterparts in synchronous architecture. In a synchronous architecture, bus transfers are usually carried out only in a specific phase of the clock signal [2], [12], thus if a bus request is delayed due to bus congestion, it must be delayed to the next clock cycle. This will significantly affect the system performance, especially when such a request is a critical one and a long clock cycle is used. For the self-timed buses in an asynchronous architecture, if a bus request is delayed because of congestion, it will be served as soon as the bus finishes the current bus service.

The buses in this architecture do not need to be all global. Each bus can be made as short as necessary to cover the functional units it connects, reducing its capacitance and improving its speed.

Two communication styles can be supported by self-timed bus structure. In one style, a functional unit sends out its output token through a bus without taking care of the state of the functional unit(s) which will receive the token. In another style, a functional unit sends out its output token only when the functional unit(s) which will receive the token is ready to do so. In our architecture, the former is used because it is faster and simpler. A potential problem of this communication style is that it may cause deadlock of the system. For example, a token which is waiting to be accepted on a bus may block other bus transfers, and hence may put the system into a deadlock state. To avoid deadlock, it is desirable that each token on a bus will always be accepted instantly by its destination(s) by offering enough storage space to accommodate it. This will be discussed later in this paper.

The authors have designed a self-timed bus with a four-cycle self-timed, bundled data communication scheme. This bus is a so called ***indirect-transfer*** bus, on which a bus transfer is completed by two separate sef-timed communications, i.e. a handshaking communication from the sending FU to the bus, and a handshaking communication from the bus to the receiving FU('s). A functional unit that gets the grant from the bus arbiter puts a token onto the bus. The FU is then released after the bus has received the token. Then the bus requests the receiving functional unit(s) through the decoding of the bus lines which carry the tag. After all the receiving functional units have received the token, the bus is released and precharged. By dividing the bus transfer into two separate procedures, more overlapping between the operations of sending and receiving functional units is allowed, hence hardware utilization and system throughput are improved.

This indirect-transfer self-timed bus is estimated to have the same drive capability as a typical synchronous precharged bus [12]. The details of our self-timed bus are to be discussed in a forthcoming paper [13].

4. DATA IDENTIFICATION AND TOKEN STRUCTURE

The synchronization of logical operations in a data-driven computing system is a difficult task. Because of functional unit sharing and communication path sharing, it is necessary to identify the logical operations shared on the same functional unit, and identify data in the system, so that a data item can be sent to the correct functional unit, and matched with another operand for the correct execution of a specific logical operation. This identification is represented as a tag which is bound to the data.

There are two approaches to naming data items in the DFG. One is by identifying the parent of the data, i.e. by the logical operation which produces the data, while the other is by identifying the child of the data, i.e. by the logical operation(s) which will consume the data. If an algorithm contains N_{op} logical operations, and we assume each logical operation has two inputs, then the ***parent identification*** scheme has N_{op} different data to identify, but the ***child identification*** scheme has at least $2N_{op}$ different items to identify.

In our architecture, it is desired that an output of a functional unit is transferred only once, no matter how many children it has, i.e. one-to-multiple destination bus transfer may be needed. With the parent identification scheme, this is easily done by a proper decoding scheme, but this is difficult with the child identification scheme. Therefore, the parent identification is chosen to identify the output data of every functional unit in the current version of our architecture.

Because the logical operations shared on a functional unit are specified in the design, a tag which includes the parent identification of a data is sufficient for uniquely defining that data, its destination in bus transfer and the logical operation(s) it belongs to. Therefore, a token can be composed of a W_d bits data and W_t bits of tag. The width of the data is not fixed in this architecture description. It can vary between different applications, with a practical choice from 12 bits to 16 bits for most real-time DSP applications. W_t equals $\lceil \log N_{op} \rceil$.

5. LOCAL MATCHING BLOCK

Operand matching to synchronize logical operations is an inherent problem of dataflow computing. As we mentioned above, because of functional unit sharing and the irregularity of the algorithms, it is possible for the operands of logical operations to arrive at the two input ports of the functional unit they share in different sequences. Operand matching is carried out for every functional unit to ensure that the functional unit takes the correct or matched pairs of operands to execute. The local matching block also generates the tag for the output token. A fast and simple matching block is strongly desired for our architecture to suit the real-time applications.

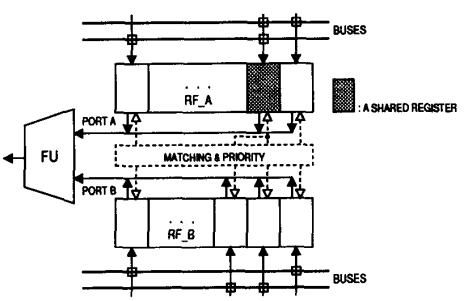

Figure 3: The matching block

A matching block is shown in Fig.3. Each input port of a functional unit has a register file, i.e. RF_A for port A and RF_B for port B. To ensure that every token on the bus can be accepted instantly to avoid the potential of system deadlock, a register is offered for operands of each logical operation in both register files. To reduce the size

of register files, two registers in the same register file can be merged if the two data they accommodate have direct or indirect dependent relation, i.e. one of them cannot be exist without the consumption of the other.

To allow the concurrent loading of the registers in a register file, each register is connected to an appropriate bus, or buses for the shared register, through a simple decoder. The decoder decides which tokens on the bus should be loaded into the register based on the tag currently on the bus.

The decoder also decodes the tag of the incoming token to give the identification of the logical operation which will consume it, to allow the correct matching of the operand pair. This is especially important for the matching of data in a shared register. After the operands have been matched and forwarded for execution, this identification can be used directly as the tag for the output token.

In Fig.3, a token in one register file only needs to match its partner in a specific register, or several specific registers in the opposite register file, hence fast matching can be expected. The matching block in Fig.3 is also convenient for implementing the priority scheme between operations.

6. INPUT AND OUTPUT BLOCK

The function of an input block is to buffer and tag the input data. An input FIFO buffer is usually needed to smooth out density fluctuations in the data flow, even though a real-time DSP chip is usually designed to process a continuous input data stream at a rate equal to or faster than the rate the data are fed in. If blocks of data are transferred between chips, then buffer memories may be needed to store the data block.

In addition to the buffering function, an input block also tags the input data with appropriate identification. We call this the ***input operation***. If an input port is shared for several inputs, then data are tagged based on the sequence of data entering the input block. The tagged input data tokens can be sent to different buses. The number of input operations should be counted into N_{op}.

Another important function of input blocks is to control entry of the data in the buffer into the system in a pipelined implementation of the architecture, so that only the desired number of continuous input data sets can enter and concurrently stay in the system, to make the system work most efficiently. This is done by a token control scheme.

The output block is much simpler than the input block. It just de-tags the tokens to be sent to off-chip and puts them into the output FIFO buffer in an appropriate sequence.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a fully asynchronous distributed VLSI architecture based on a data-driven computing model and a self-timed signaling protocol. This architecture has the following advantages:

- The asynchronous, self-timed signaling protocol circumvents the difficult global distribution of clock signal, and overcomes the potential clock skew problem in a large VLSI system design. This advantage will be more significant with the continuous scaling down of IC process technology and growth of chip size in the near future.
- The data-driven computing and the distributed control allows the maximum exploitation of the fine-grained concurrency. The out-of-order execution of logical operations gives the architecture more efficient hardware utilization and higher throughput than the conventional control-flow synchronous VLSI processor architecture.
- This architecture covers a wider range of applications than the systolic array architecture, from applications with very regular algorithms to those with irregular algorithms.
- The self-timed design methodology allows functional units in the system work on an average computation delay, instead of the worst-case computation delay in a synchronous design. Thus, a single functional unit may potentially have a better average performance than its synchronous counterpart.
- The design effort of an asynchronous chip can be reduced, because each functional unit and its matching block can be designed and tested locally, without undue attention to the global timing constraints.
- The asynchronous chip can be used more flexibly in a board level design because of the very loose inter-chip communication constraint. The system constructed on asynchronous chips is more extendable.

The problems of the current version of the architecture include the inefficient utilization of local registers. At this stage, because register sharing is only considered within the register files, the chance of the register sharing is limited. The direct connection of registers in a register file to the bus may increase the loading of the bus. These problems are to be considered in the near future.

The future work will be focused on the high-level synthesis of the fully asynchronous VLSI chip from a data flow graph(DFG) or a high level behavior description. Special synthesis algorithms are to be developed for functional unit sharing, logical operation assignment, especially for the design of pipelined systems.

As far as we know, this is the first fully asynchronous, distributed VLSI architecture which supports data-driven computing, for ASICs or dedicated DSP applications. We anticipate that our research will lead to the wider adoption of asynchronous, self-timed techniques, with their inherent advantages, in such applications.

8. REFERENCES

[1] S. Y. Kung, *VLSI Array Processors*. Reading, Prentice Hall, 1988.

[2] B. S. Haroun and M. I. Elmasry, "Architecture synthesis for DSP silicon compilers," *IEEE Trans. Computer-Aided-Design*, vol. CAD-8, pp. 431 - 447, April, 1989.

[3] J. Rabaey, H. De Man, J. Vanhoof, G. Goosens and F. Catthoor, "CATHEDRAL-II: A synthesis system for multiprocessor DSP systems," in *Silicon Compilation*, D. Gajsk, Ed. Reading, MA: Addison-Wesley, pp. 311 - 360, 1988.

[4] C. L. Seitz, "System timing." Chap. 7 in: C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA:Addison-Wesley, 1980.

[5] A. J. Martin, "Programming in VLSI: From communicating processes to delay-insensitive circuits," in *Developments in Concurrency and Communication*. C. A. R. Hoare, Eds. pp.1 - 64, Addison-Wesley, 1990.

[6] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messerschmitt, "Automatic synthesis of asynchronous circuits from high-level specifications," *IEEE Trans. Computer-Aided-Design*, vol. CAD-8, pp. 1185 - 1205, Nov. 1989.

[7] G. M. Jacobs and R. W. Brodersen, "A fully asynchronous digital signal processor using self-timed circuits," *IEEE J. Solid-State Circuits*, vol. 25, pp. 1526 - 1537, Dec. 1990.

[8] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, pp. 720 - 738, June, 1989.

[9] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic and P. J. Hazewindus, "The design of an asynchronous microprocessor," *Decennial Caltech Conference on VLSI*, C. L. Seitz, ed., pp 351 - 373, MIT Press, 1989.

[10] X. C. Fan and N. W. Bergmann, "Design of elements for a self-timed fast packet switch," in *Proc. 1991 IEEE International Symposium on Circuits and Systems*, pp. 1025 - 1028, 1991.

[11] C. L. Seitz, "Let's route packets instead of wires," in *Advanced Research in VLSI, Proc. 6th MIT Conf.*, pp. 133 - 138, 1990.

[12] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA:Addison-Wesley, 1980.

[13] X. C. Fan and N. W. Bergmann, "Design of a self-timed bus for a fully asynchronous VLSI DSP chip," *In preparation*.