A Functional Level Simulation Engine of MAN-YO
A Special Purpose Parallel Machine for Logic Design Automation

Toshiyuki Nakata, Nobuhiko Koike

C&C Systems Laboratories, NEC Corporation
4-1-1 Miyazaki Miyamae-ku Kawasaki 213 , JAPAN

ABSTRACT

The architecture of a proto-type functional level simulator element of a massively parallel machine (MAN-YO) designed for logic design automation is presented. At functional level, hardware systems are described in a hardware description language, FDL. The FDL description is compiled into stack oriented intermediate language instructions. Communicating with other gate level/block level/ functional level processors, each functional simulator interprets the compiled instructions and simulates various circuits using 4-value logic. In order to realize high speed processing of 4-value logic/arithmetic operations, the functional simulator utilizes low-level parallelism realized by 3 ALUs which are controlled by the different fields of a long horizontal type microinstruction.

By utilizing low-level parallelism at processor level, as well as processor level parallelism, high speed execution of mixed level simulation becomes possible. The system also provides further performance enhancement by compiling often used FDL macros into microcode.

This paper describes an outline of the MAN-YO (Japanese for ten thousand leaf-nodes in the processor tree),a brief description of FDL, and the architecture of the functional level simulator element (called FDLPE). A rough performance based on the current design is also described.

1 INTRODUCTION

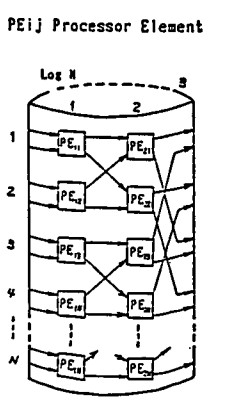
The rapid increase in VLSI complexity has made various impacts on design methodologies and CAD tools. Conventional simulators implemented by software have become inadequate, as far as processing speed is concerned. One answer to this problem is the development of special purpose-hardware logic simulation engines. Several engines, such as HAL[1][2][3], YSE[4] and Zycad Logic Evaluators, have been developed. These accelerators have been successful in enhancing simulation speed by several orders of magnitude. However, their application fields are limited to gate level or block level simulations and they alone cannot cope with another trend in VLSI design. Namely, in order to meet the increase in VLSI complexity, there is a tendency to shift the logic design step to a higher (functional) level. To cope with this tendency, there has been an increasing demand for an efficient mix-level simulator, which supports functional level simulation while maintaining the high-speed processing capability of dedicated hardware simulators.

MAN-YO [5] is a parallel processor array, designed to provide significant performance improvement in executions of logic automation tasks. In the MAN-YO system, a large array (on the order of a thousand) of processor modules are connected by a loop-structured inter-connection (See Fig. 1.1). Each processor element in the system contains a logic simulation element, a functional level simulation element, a microprocessor and a router cell. The logic simulation element performs gate and block level simulation. (A block is a collection of 10 to 100 gates.) The microprocessor, which manages symbolic data, performs symbolic simulation and knowledge based processings. The router cell manages packet transmission among PEs.

This paper describes the architecture of the functional level simulation element, called FDLPE (FDL Processor Element). At the functional level, hardware systems are described in a hardware description language called FDL. The FDL description is compiled into stack oriented intermediate language instructions. The FDLPE interprets these instructions and simulates various circuits, using 4-value logic. In order to realize high speed processing of 4-value logic/arithmetic operations, the FDLPE uses low-level parallelism.

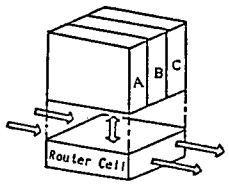
Low-level parallelism is defined as a fine grained parallelism at the register-

0884-7495/86/0000/0202$01.00 © 1986 IEEE



(A) Man-Yo System Organization

A : Logic Simulation Engine
B : Functional Simulation Engine
C : General Purpose Micro-Processor



(B) Processor Element blockdiagram

Figure 1 Man-Yo System blockdiagram

transfer level, where several ALUs, memory control units and sequencers operate in parallel under the control of a long word instruction (or a horizontal type microinstruction.) Several experimental or commercial processors with low-level parallelism have been built, such as, QA-2[6][7], AP-120B and MENTOR Graphic Corporation's Compute Engine[8]. Until recently, these processors had been large-scaled processors. However, with the availability of hardware building blocks, such as AMD29300 series and CMOS gate arrays, it has become possible to utilize processors with low-level parallelism as an element in a multi-processor system.

In the FDLPE, three ALUs work in parallel, sharing a 2-port register file and a hardware stack, under the control of a horizontal type microinstruction. In order to reduce the overhead involved in memory access, two kinds of cache (instruction cache and a read-only data cache) were implemented.

By combining low-level parallelism at functional processor element level with processor-level parallelism due to parallelism in the simulation algorithm, large-scaled mix simulator can be realized which operate at several orders of magnitude faster than conventional software simulators.

2 FDL: A STRUCTURAL BEHAVIOR DESCRIPTION LANGUAGE

2.1 Overview of the FDL

This section briefly describes the hardware description language used in the authors' system called FDL. As this paper is not meant to describe FDL, the authors only touch on the characteristics of the FDL and how FDL is processed in the system. For more details on FDL, see [9].

In an FDL program, a digital circuit is described as a set of interconnected elements, called nodes. A node is a section of a circuit, such as a combinatorial circuit, register, memory or a combination of the three. Usually, a node corresponds to a block in a hardware block diagram. FDL is a structural behavior language(or structural/functional language[10]). In FDL, each node is described within one statement. A hardware element may appear only once in the left hand side of an FDL statement. A set of FDL statements describes one hardware module, such as a small gate, LSI, CPU or a logic library.

There are seven kinds of statements in FDL. INPUT, OUTPUT and INOUT statements describe boundary (interface) signals. MODULE statement describes other FDL module invocations and interfacing signals which go into / out of these modules. REGISTER, TERMINAL and MEMORY statements describe internal signals or node definitions. INPUT,OUTPUT, INOUT and MODULE statements can be considered as declaration statements and the REGISTER,TERMINAL and MEMORY statements can be considered as execution statements.

The right hand side of the latter three statements are expressions, which are recursively composed of CASE clauses, IF clauses and operations on sub-expressions. No rigorous definition of these clauses or operations will be given, but roughly speaking, the CASE clause corresponds to a switch expression (that is, a switch statement which returns a value), an IF clause corresponds to an if expression and various operations correspond to operations in the C language. Meanings of various operations are shown in Table 1, and an example circuit and its FDL description are shown in Fig. 2.1.

Table 1 Operators in the FDL

CLASS	OPERATOR	NOTATION
LOGICAL OPERATOR	NOT	.NOT. or '
	OR	.OR. or +
	AND	.AND. or *
	EXCLUSIVE OR	.XOR. OR @
	COINCIDENCE	.COIN.
	GATING	.IF.
	REDUCTION OR	.ROR.
	REDUCTION AND	.RAND.
	REDUCTION EXCLUSIVE OR	.RXOR.
FUNCTIONAL OPERATOR	SHIFT LEFT	.SHL.
	SHIFT RIGHT	.SHR.
	ROTATE LEFT	.ROTL.
	ROTATE RIGHT	.ROTR.
	CONCATENATE	.COMB. or -
	ADJUST SIZE	.SIZE.
	COPY PATTERN	.COPY.
	LEADING0	.LD0.
	LEADING1	.LD1.
	TRUE AND COMPLEMENT	.TC.
ARITHMETIC OPERATOR	ADD	.ADD.
	SUBTRACT	.SUB.
	MULTIPLY	.MUL.
	DIVIDE	.DIV.
	CARRY	.CRY.
RELATION OPERATOR	EQUAL	.EQ.
	NOT EQUAL	.NE.
	GREATER THAN	.GT.
	GREATER OR EQUAL	.GE.
	LESS THAN	.LT.
	LESS OR EQUAL	.LE.
TIMING OPERATOR	GO UP	.UP.
	GO DOWN	.DN.

2.2 FDL PROCESSING SYSTEM

The FDL processing system consists of compiler, linker, and mix level-simulator.

The compiler compiles the FDL program into a stack-oriented intermediate code. It also derives the linking information from the INPUT, OUPUT, INOUT and MODULE statements, and puts them out in a tabular form for the linker. From these tables, the linker creates a node network which consists of fan-in-lists, node values, and fan-out lists.

The intermediate codes for FDL can be classified into the following groups.

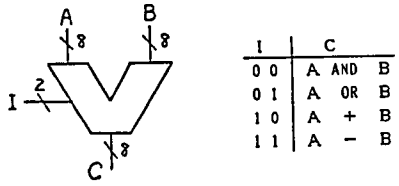
a) Get codes: These operations read the value from the fan-in list and push them on the stack.

b) Operator codes: These codes correspond to operations in FDL. They pop-up the operand(s) from the stack, perform logical/arithmetic operations and push the result on the stack.

c) Branches: Used for IF clauses, Case clauses and exits.

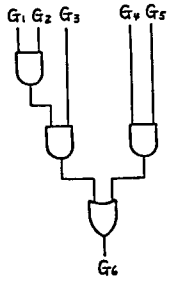
d)Assignment codes: These codes put the value on the stack to the appropriate node. If the new value differs from the old value, then an event is registered on the event wheel.

The compiled code and the node network is loaded on to the FDL processor at runtime, and the simulation is started.



I	C
0 0	A AND B
0 1	A OR B
1 0	A + B
1 1	A - B

```
INPUT A(0:8), B(0:8), I(0:2);
OUTPUT C(10:8);
C(0:8) = CASE I OF
          /0  0/  A .AND. B,
          /0  1/  A .OR. B,
          /1  0/  A .ADD. B,
          /1  1/  A .SUB. B;
```




```
TERMINAL G6<4,5,6> = (G1*G2*G3)+(G4*G5);
```

Figure 2.1 Example Circuits and their FDL Descriptions

2.3 Mix Level Simulator

The mix level simulator of MAN-YO is an event driven simulator. Summarized below is the logic simulation process for the prototype FDL processor.

1)Event set : When events are sent from other FDL processors, gate level processors, or block level processors, they are registered in an input event queue. The FDLPE extracts the events and sets them into input event node lists which are connected by a fan-in list. It then registers the functional node corresponding to the hardware which receives the event in an input event list.

2) Functional node evaluation:Each function node has a pointer which points to the intermediate code corresponding to the FDL description of the node. The FDLPE simulates each node by interpreting the intermediate codes and registers the output data with the delay count in an event wheel.

3) Delay Operation: Each time the time

wheel is advanced by one, the FDLPE decrements the delay count by one and extracts values whose delay count is zero. If the new value differs from the old value the FDLPE searches for the addresses of the hardware element (logic gate node / block node/ or FDL node) connected to the changed output. If the element resides within this PE, the output data is transferred to the input queue. Otherwise the FDLPE creates an event packet and sends it to the router for delivery.

2.4 Micro-compilation

The FDLPE was designed to efficiently simulate the hardware blocks by interpreting the FDL-intermediate codes. However there is another way to gain performance, namely by microcompiling. Microcompiling has been implemented on systems such as the QA-2[11] and the LAMBDA machine[12], and have proved to be useful if core functions which are used often are compiled. By compiling FDL codes of often used macros into microcode, an additional 200-300% performance gain can be expected.

3 HARDWARE ORGANIZATION OF THE FDLPE

3.1 Over-all Organization

The over-all configuration of the FDLPE is shown in Fig.3.1. It consists of the MIU (Macro-Instruction Unit), the FEU (Function Evaluation Unit), and the SCU(Sequence Control Unit). These units are controlled by the separate fields of a horizontal type microinstruction. The MIU is mainly concerned with fetching the next intermediate FDL-code, and determines the entry point for the microroutine corresponding to the next FDL-code. The FEU fetches the logic values from the fan-in lists, and executes logical/arithmetic operations, specified by the intermediate FDL code on data stored on a stack . These two units are connected by a 2-port register file. So, for FDL-codes, such as conditional / multiway branches, these units work together , communicating through the register file. Also, when executing complex operations, the MIU can work as a part of the FEU. These units also work together in event set and delay operation phases. The SCU fetches the next microinstruction of the FEU. The microinstruction fetch and the execution phases are overlapped to provide maximum efficiency.

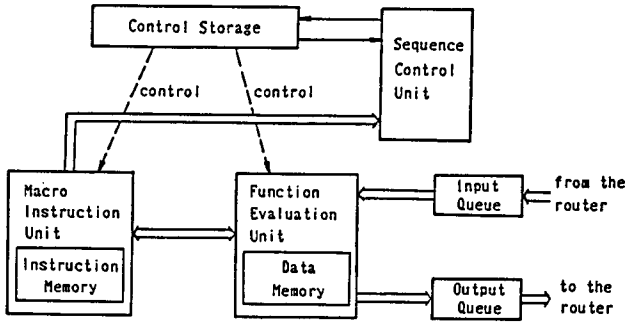


Figure 3.1 Over-all Configuration of the FDLPE

3.2 The Organization of the MIU

The hardware organization of the MIU is shown in Fig. 3.2. It consists of the instruction memory, instruction cache, microinstruction address table and an ALU with several registers plus a communication register to the FEU. Design considerations, which led to the implementation of the instruction cache are summarized below.

1) As has been stated before, the intermediate language for the FDL is stack oriented, and has many codes which are only one byte in size. Thus, the instruction memory must have byte access capability and must contain bus exchange logic to bring arbitrary byte fields to the 32-bit ALU.

2) FDL programs become very large for complex systems. So in order to meet the capacity demands, it would be necessary to use dynamic RAMS. (with the capacity of 64-kw*4bits or more).

3) On the other hand, if the instruction memory is too slow, the efforts spent in speeding up the ALU operations will be wasted due to the overhead involved in

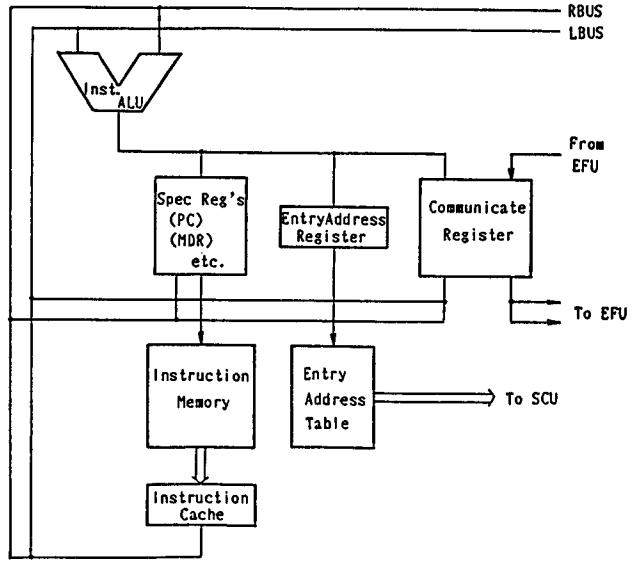


Figure 3.2 Hardware Organization of the Macro Instruction Unit

instruction fetch and decode phases [11]. For the memory to keep pace with the speed of the FEU (100 nsec or so,), some kind of cache is necessary.

4) It is desired to build the whole system using many FDLPEs, so the entire implementation will have to be cheap.

In order to satisfy these conditions, the authors designed a cheap instruction cache taking advantage of the high-bandwidth offered by the dual ported video RAMs.

By combining the VRAMS with shift registers, as shown in Fig. 3.3, the instruction memory also acts as an instruction buffer The shifters implement the bus exchange logic and the instruction cache. By this configuration, overhead due to slow memory speed is reduced to zero for most cases. The only exceptions are various branch operations, which are taken care of by microprograms. (Even in this case, overhead is as little as 60nsec for short branches and at maximum, 300nsec for long branches.)

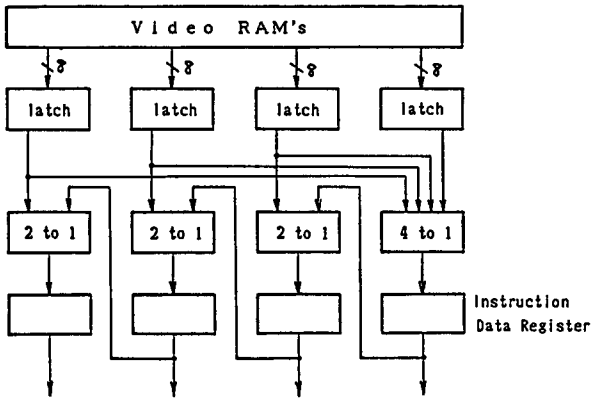


Figure 3.3 Organization of the Instruction Cache

While the FEU is executing the appropriate logical/arithmetic operations, the MIU fetches the next opcode of the FDL intermediate code, and derives the entry of the microroutine by referring to the microinstruction address table. The entry address is used by the SCU at the end of the microroutine, to get the next microinstruction address.

3.3 Hardware Organization of the FEU (Function Evaluation Unit)

The FEU is composed of the evaluation unit (comprised of two ALUs , a 2-port register file, a hardware stack) , a data collector, and the data memory (See Fig. 3.4). Features of each part are briefly outlined.

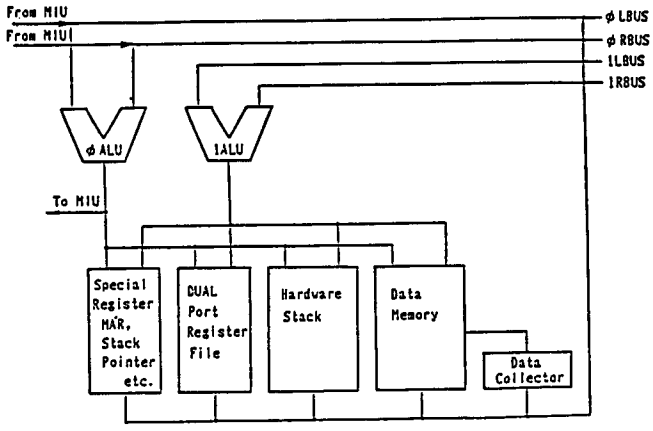


Figure 3.4 Hardware Organization of the Function Evaluation Unit

3.3.1 Data Format

As mentioned before, the FDLPE simulates functional blocks using 4-value(0,1,z(high impedance) and x(unknown))logic. There are three data categories; short (up to 32-bits), double (up to 64-bits) and triple (up to 96-bits). In each category, the x-z bits and 0-1 bits are grouped as shown in Fig. 3.5. By treating the 0-1 bits and x-z bits as one would treat the real part and the imaginary part of a complex number, the overhead in logic/arithmetic operations due to treating 4-value logic can be kept to a minimum.

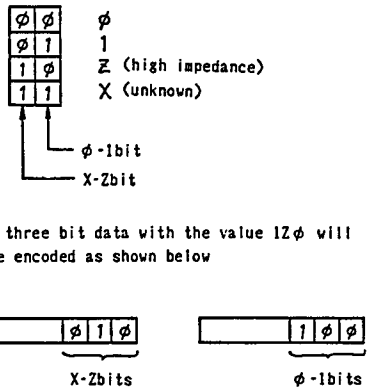


Figure 3.5 Bit encoding scheme for the 4-bit values

3.3.2 Evaluation Unit

The hardware configuration of the evaluation unit is shown in Fig 3.4.

The ALUs and the register file are 32-bits wide, while the stack is 64-bits wide. The ALUs operate in parallel on data in the registers, and the stack. The microinstructions, which control the ALUs and the registers are of the 3 operand

type to provide maximum flexibility. The operations supported are the usual arithmetic/logical operations. Status bits include zero, minus and parity. The last is used in exclusive-or operations.

The register file is a 64-word by 32-bit dual port register file, using either the AMD 29323 or 4 copies of high-speed RAMs. The hardware stack (1-kw x 64-bits) is a FIFO, of which only the top 64-bit data can be accessed in one microinstruction. The 64-bit data is composed of the 32-bit x-z bits and the 32-bit 0-1 bits. The upper 32-bits and the lower 32-bits can be independently accessed by all the ALUs (including the ALU of the MIU) in one microinstruction. When stack overflow/underflow occurs, an interrupt at microprogram level will be invoked.

3.3.3 Data memory and the data collector

Due to the structure of the fan-in lists, Sometimes the data in the FDLPE is stored in the manner shown in Fig.3.6. So, in order to efficiently transform the data in the main memory to the data format used by the evaluation unit, a data collector was designed for collecting the x-z bits and the 0-1 bits from multiple words , using the VRAMs and shift registers as in the case for the instruction cache, as shown in Fig 3.7. Barring refresh cycles, this collector can collect an n-bit data (n<=32) in 120+40n nsecs.

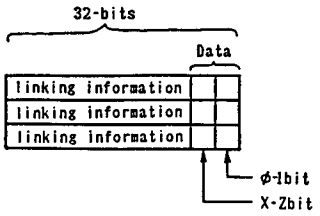


Figure 3.6 Data Format in the Data Memory

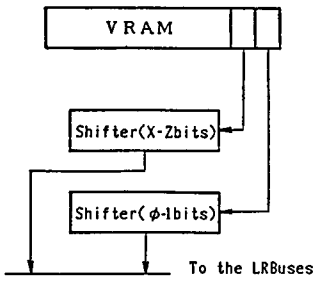


Figure 3.7 Configurationof the Data Collector

3.4 Hardware Organization of the SCU

The SCU determines the address of the next microinstruction and fetches the microinstruction from the control storage. The overall configuration of the SCU is shown in Fig 3.8. The capacity of the control storage is 2-KWs. The SCU supports unconditional branches, microsubroutine calls/returns (up to 32 levels), indirect branch using the value from the data sent from the MIU , conditional branches of the type if <condition> then goto label else NEXT, and a multiway jump using two status bits. For the conditional branch, one status bit, chosen from various status bits sent from the 3 ALUs is used. As the microinstruction fetch is overlapped with the execution phase, the status used by the SCU will have to be determined by the microinstruction before the execution of the conditional and multiway branches. When a micro-interrupt (such as hardware stack overflow/underflow or interrupt from the master processor in the MAN-YO), occurs, it is treated just like a microsubroutine call, except that the address of the current (not the next) address is saved on the stack. It is intended to implement the SCU either using the AM29331 or a custom CMOS gate array.

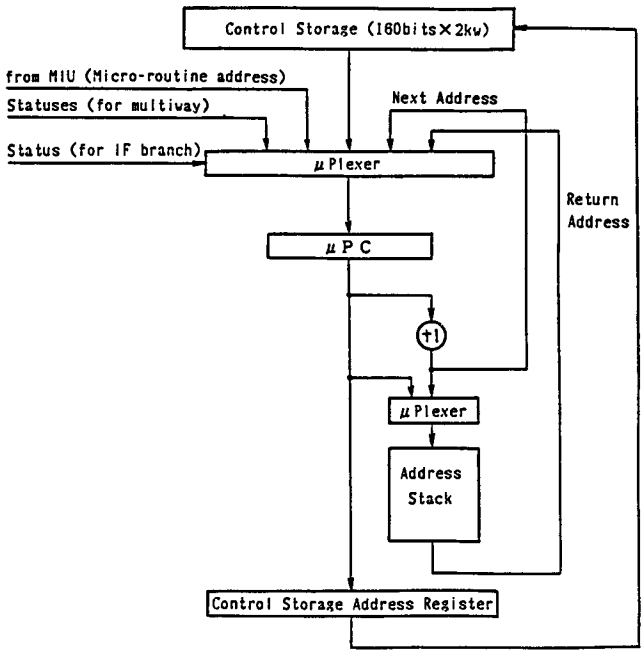


Figure 3.8 Hardware Organization of the Sequence Control Unit

4 PERFORMANCE ESTIMATION

As hardware design is still at the early stage, it is difficult to estimate the performance of the FDLPE let alone the total throughput for the system. However, in order to determine the effectiveness of the implementation decisions , we calculated the the number of EFU micro-cycles for the FDL statement shown in Fig. 4.1. The number of micro-instructions executed is shown in Table 4.1. In this simple case, even with the data collector, over head due to data memory access is influential. As the FDL expression becomes more complex, this overhead is expected to decrease. However, even for these cases, assuming a 120 nsec microcycle time, (which we believe is quite reasonable,) the execution time of one FDLPE would be 10 times that of a VAX-780 and would be comparable to 10 MIPS mainframe computers at much less hardware cost. As the granularity level of functional level simulation is much larger than the granularity level of gate-level processors, by employing a network of FDLPEs, with gate-level PEs and block-level PEs, mix-level simulators will become possible, which are on the order of several orders of magnitude faster than conventional simulators .

Table 4.1 Performance Estimation

Estimated cycle for simulating
C(0:8) = CASE I OF

/0 0/	A.AND. B
/0 1/	A .OR. B
/1 0/	A.ADD. B
/1 1/	A.SUB. B

in 4 value logic

Number of microcycles	30~32
Cycle loss due to memory access	8

5 CONCLUSION

The architecture of a functional simulator element of a massively parallel processor for logic design automation was presented. By making use of low-level (fine grained) parallelism and pipelining , as well as processor level (coarse grained) parallelism, mix-level simulations will become possible, which are on the order of several orders of magnitude faster than conventional simulators .

ACKNOWLEDGEMENTS

The authors wish to thank Dr. Katsuya Hakozaki and Dr. Masahiro Yamamoto for their kind interest in this work. They also thank Mr. Tohru Sasaki, Tsuneo Kurobe, Nobuyoshi Nomizu and Yoshitada Fujinami for many helpful discussions, and Mr. Shinichi Habata for discussions which led to the implementation of the MIU.

REFERENCES

[1] N.Koike, K.Ohmori, H.Kondo and T.Sasaki, "A High Speed Logic Simulation Machine",digest of papers, Spring COMPCON, pp446-451, Feb.,1983

[2] T.Sasaki, N.Koike, K.Ohmori, and K.Tomita "HAL; A Block Level Hardware Logic Simulator", proc. 20th Design Automation Conf., pp150-156, June, 1983

[3]N.Nomizu, T.Sasaki, H.Tanaka, N.Koike and K.Ohmori, "Block Level Hardware Logic Simulator -Its Application and Results-",proc. ICCAD 1984 pp254-256, Nov. 1984

[4] G.F.Pfister, "The YORKTOWN Simulation Engine: Introduction", proc. 19th Design Automation Conf., pp-51-54, June, 1983

[5] N.Koike and K.Ohmori "MAN-YO : A Special Purpose Parallel Machine for Logic Design Automation", proc. 1985 ICPP, pp.583-590, Aug.1985

[6] S.Tomita, K.Shibayama, K.Kitamura, T.Nakata and H.Hagiwara,"A User Microprogrammable Computer with Low Level Parallelism", Proc. 10th Annual Int. Symposium on Computer Architecture, pp153-159, June 1983.

[7] H.Hagiwara, S.Tomita, S.Oyanagi and K.Shibayama, "A Dynamically Microprogrammable Computer with Low-Level Parallelism", IEEE Trans. on Computers, Vol.C-29, No.7, pp557-595, July 1980

[8] M.R. Butts, "A General-Purpose Accelerator", VLSI Systems Design, pp.85-86, Oct. 1985

[9] S. Kato , and T. Sasaki, "FDL: A Structural Behavior Description Language" IFIP-1983, (Computer Hardware Description Languages and Their Applications Uehara T., and Barbacci M. (ed) , North-Holland Publishing Company) pp.137-152(1983)

[10] S.M.German and K.J.Lieberherr, "Zeus: A Language for Expressing Algorithms in Hardware", IEEE Computer, Vol.18, No.2, pp55-65, Feb. 1985

[11] S.Tomita, K.Shibayama, T.Nakata, S.Yuasa and H.Hagiwara, "A Computer with Low-Level Parallelism QA-2
- Its Applications to 3-D Graphics and Prolog/Lisp Machines -", Submitted
to 13th Int'l Symp. on Computer Architecture

[12] LMI, "The Microcompiler"