A Study in the Use of PLA-Based Macros

PETER W. COOK, MEMBER, IEEE, CHUNG W. HO, MEMBER, IEEE, AND STANLEY E. SCHUSTER, MEMBER, IEEE

Abstract—**This paper describes a study in which a PLA-based macro design of a small processer is carried out in the same technology as the original "random" logic design of the same processor. The objectives of the study were to determine gains or losses in "technology utilization" when a PLA-based approach is used to replace the more conventional "random" logic approach. The results in this case are a design of equal performance and density, with only one-third the power of the original design.**

INTRODUCTION

IN RECENT years, the PLA (programmable logic array) [1] has received considerable attention as a possible replacement for conventional random logic and/or ROS (read-only storage) in digital system design. This attention is largely due to growing difficulties in the areas of physical design, timing and logic verification, and testing, which lead to untenable design times and costs with conventional methods. Moreover, technological progress has reached the point where large arrays capable of implementing significant logical functions are physically realizable.

Comparisons of PLA's and conventional "random" logic implementations of systems that can pinpoint any advantages or disadvantages to the PLA approach are not common; at the time of this work, they were not existent. Such comparisons as existed [2] included not only a change in design methodology (the use of PLA's) but also included significant technological differences (LSI versus SSI), and the results attributable to PLA's alone were difficult to determine. The work of [2] also focused on what can be termed "fixed-image" PLA's: in such PLA's, the size of the array is fixed and the designer can only specify bit patterns in that fixed array image. This leads to the search for a universal image and, in specific designs, can lead to a forced use of the PLA.

In the context of LSI or VLSI, such a fixed-image approach seems unwarranted. As technology advances, it is unlikely that any logic chip will be simply one "thing" such as a PLA (present microprocessors that combine in a single chip logic, RAM and ROS are examples of this). An approach termed the "PLA-based macro" approach in which the chip designer has access to a limited number of flexible parts including PLA's, registers of several kinds and buses to provide the required combinatoric logic, storage, and communications functions has been chosen. All of these "parts" are flexible in their size, so that a register can be from one to many bits, and a PLA can range from several hundred bits to several thousand. This flexibility prevents any artificial forcing of functions into a fixed image of any kind.

Manuscript received March 1979; revised June 8, 1979.

The authors are with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

Because there was no "common technology" comparison of PLA-based designs and more conventional "random logic" designs, the present study was begun. It objective was simple: to determine, in a common technology base, the possible advantages or disadvantages in using the PLA-based design approach. The method adopted was to design, in a known technology using the PLA parts described briefly above, a small low-end microprocessor. The final design was to be compared with a random logic design of the same processor in terms of power, speed, and density. The particular microprocessor (an 8-bit machine designed elsewhere in IBM as a 1200 circuit MOSFET chip) was chosen for this study because it was simple enough to be accommodated by a small group, already was designed in "random" logic, and was well enough defined to provide a fixed target. The design resulting from that study is the subject of this paper.

The basic approach taken in this study was to provide the designer with a set of three very flexible parts. The parts include PLA's for combinatoric and control functions, registers of several types, and appropriate means for interpart communication.

CONCLUSIONS

The results of this study show that the PLA-based macro approach to machine design can, with appropriate circuitry, result in a machine exhibiting equal or superior characteristics relative to the same machine designed in conventional random logic. The equality or superiority is measured in terms of power, performance, and density of a large function, such as an entire microprocessor, and is observed when both designs are implemented in an identical technology. The major area of superiority observed is in the power per function and is a result of the dynamic logic circuits employed essentially throughout. This form of dynamic logic is in turn made practical by the organization of the system into large macros, which simplifies clock distribution, and by the specific circuitry employed, which allows the use of simple drivers at the output of every logic stage.

This observed result is based upon a careful competitive design effort in a low-end machine; the extension of the result to higher performance environments may require further study.

VEHICLE FOR STUDY

The microprocessor used in this study is a single-chip processor implemented in a MOSFET technology. The processor, considered equal to about 1200 circuits, fills a single 235 mil X

0018-9200/79/1000-0833$00.75 © 1979 IEEE

235 mil chip. Worst case power is 400 mW. The basic measure of processor speed is taken as the instruction rate of the processor. The processor interfaces a 64K byte memory with a 3.2 μs cycle time. A single byte is read or written in a cycle. The instruction set of the machine consists of some 45 instructions, most of which employ single byte operands, though some are double byte operations. The (unweighted) average instruction time is 2.71 memory cycles, or 8.67 μs; this gives a processor speed of 115 000 instructions/s. The machine has general registers when viewed logically; the general registers are in fact implemented as the low address space of main memory, so that register accesses are actually memory accesses.

Macro Set

The set of macros used in this study has three members, all parameterized so as to achieve extensive use. Members of the set and their basic characteristics follow.

1) PLA: used for combinatoric elements, including next-state computation in sequential controls. As a macro, the PLA is parameterized in both size and function by its bit pattern. Additional parameters provide 2-bit partitioning options, output driver types, and electrical characteristics.

2) Registers: used for all on-chip storage, but not main memory. Parameters control latch type, data width, and electrical details.

3) Buses: used for all data transfers. The bus macro is also capable of dot-oring several inputs. Parameters include width and electrical parameters. The bus is an unusual macro in that it is not confined to a single area of a chip but rather is distributed; however, bus support circuits such as precharge paths, load devices, and clamp circuits are localized and parameterized.

With these macros in mind, a data flow for the PLA-based processor was created. A diagram of the data flow appears in Fig. 1. Several factors went into the partitioning of the machine as shown. First, most instructions are single byte operations. This, together with the complexity of a 16-bit ALU led to the use of an 8-bit ALU for all data path operations; 16-bit instructions were to be accommodated in two ALU passes. This was felt to represent no performance penalty since only a single byte path from memory was provided and fetching arguments for 16-bit operations would require two memory cycles in any case. On the other hand, all addressing in the processor is 16 bits. Address incrementing could have been done in the main ALU; however, it was believed to be more efficient to provide a separate 16-bit incrementing ALU (IALU) to provide the increment function. This function is, of course, much simpler than a general ALU, and so several other forms of address modification were subsequently incorporated into the IALU. Two major buses were provided in the system: the LOBUS and the HIBUS. The LOBUS is used for all operands and ALU operations; the HIBUS and LOBUS together are used for addressing paths. The shared use of the LOBUS together with the use of several registers that can communicate to both buses simplifies saving of instruction addresses during an interrupt or branch-and-link operation.

Key to the data flow is the ability of the clocked PLA to feed its output directly back into its own input, with no race problems. Details will be presented subsequently; basically, this results from the specific circuitry used in the PLA which

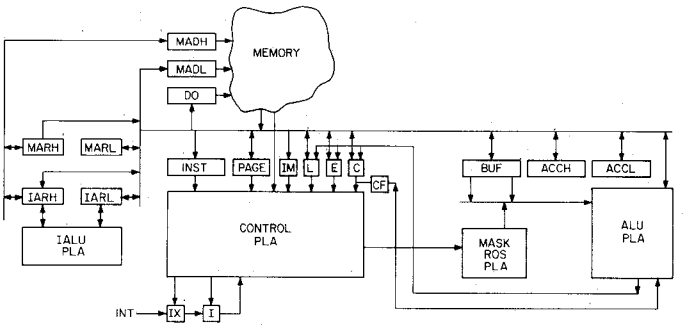


Fig. 1. PLA-based processor data flow. The PLA macros are designated in the diagram; the remaining blocks are registers of from 1 to 8 bits width, using several latch types.

behaves as if the input were provided with a sample-and-hold circuit. Thus, the single connection of the ALU PLA to the LOBUS serves as both input and output, the functions being separated in time. Likewise, most registers which tie to the LOBUS do so by a bidirectional path. In the data flow, a single PLA cycle can allow for data to be brought from a register to ALU, operated on, and returned to the same or another register.

At this point it became necessary to define in detail the various parts of the machine. This entailed the electrical design of the various macros and the verification of their operability under appropriate worst case circumstances. Second, it was necessary to define in rather careful detail the contents of the control PLA. In PLA's size not only implies silicon cost, it also implies delay. It was therefore desirable to implement the control function in as small a PLA as possible. While one can obtain rather general insights regarding the size of a PLA for certain combinatoric functions as might be found in ALU's, control functions are less structured, and detail was here considered important.

Circuit Family

The PLA-based macros used in this study were all based on dynamic circuits. This resulted from a need to drive some circuits under conditions of high (exceeding 100) fan-out, and the ability to provide low-power bootstrap drivers in simple dynamic circuits.

Dynamic logic, per se, is not new, and has been used in random logic arrangements, and the question naturally arises, why not use dynamic random logic. In an earlier study of various FET logic circuit alternatives [4], dynamic random logic was considered; it was found to offer no significant advantages over static random logic. This arises from problems associated with clock distribution, from the need to restrict the number of phases lying between two circuits that are connected to each other (a "phase 1" circuit cannot drive a "phase 1" circuit), and from the circuitry itself, with stacked devices, limited levels, and possibly higher node capacitance. The specific circuitry used in the PLA's in this study avoided these problems, and offered significant power reductions.

Timing

All circuitry in the PLA-based processor operated on a single four-phase clock. Each phase consisted of an on time followed

by an identical interphase gap time. Thus, if T is the on time of a given phase pulse, the entire four-phase cycle will take $8T$. At the outset of the physical design process, the following timing assumptions were made.

Φ1: Loads register from bus; (precharge AND array in PLA).
Φ2: (Drive 2-bit partitioning outputs to AND array; precharge OR array.)
Φ3: (Precharge buses; refresh registers; drive AND array outputs into OR array.)
Φ4: Drive OR array output or register content to bus.

The actions described above are such as to have their effects remain in force until explicity altered. Thus, when a Φ4 signal drives a register content to a bus, the bus retains that register content until the subsequent Φ3 time, when buses are all unconditionally precharged.

In the list above, the items in parentheses are unconditional and ensure the proper operation of the macros themselves. Items not in parentheses are conditional, and must be controlled in such a way that the resulting machine performs the correct operations. Therefore, pulsed drivers, identical in structure to those used internally in the PLA's were designed as possible PLA output drivers. These are so arranged that when connected to a PLA output, they may provide conditional pulses at times Φ4 or Φ1. These are the times during which all conditional operations take place, and the drivers allow the use of an otherwise conventional PLA as a control for the machine.

The original processor was designed to interface a 3.2 μs memory. A preliminary examination of several of the instructions in the data flow of Fig. 1 indicated that four PLA cycles per memory cycle would ensure equivalent performance of the "random" and PLA-based processor machines. This translates to an 800 ns (worst case) PLA cycle, and a nominal 100 ns on-time for any phase.

Logic Levels

The PLA is normally thought of as an AND-OR device. In its simplest form it performs this function only if a ground level is taken as logical 1. While inverting drivers make such strict binding unnecessary, it was decided to adopt such a definition of logic levels for all data paths. This served to facilitate the construction of buses that could dot-or data from several sources.

For control pulses, the opposite approach was taken, with a high level taken as logical 1 for all control pulses. This is because it is far easier to initiate an action with such a signal than with a low level. Again, the choice proved useful when it became evident that simple bootstrap drivers used for such control pulses could also dot-or control pulses from several sources if such pulses were defined with a high logical 1.

Interface and Clocking

In the original processor, clocks control the interface to the memory: when the memory is busy, some processor clocks are stopped, thus placing the processor in a hold state. For the PLA-based processor, it was assumed that the basic four-phase clock was continuously running. A line (*DR*) was assumed available to the processor to signal when the memory had completed the last action requested by the processor. This line is monitored by the control PLA of the PLA-based processor, which places that processor in a hold state whenever a requested memory action has not been serviced. The original processor did not directly drive the memory address bus, but drove that bus through an interface chip. In the PLA-based processor, it was assumed that memory address bus contention problems would be accommodated in that chip, just as the original processor accommodated such problems through the clock chip. In retrospect, it appears possible to accommodate the memory address contention problem in a manner more consistent with the original design by eliminating the *DR* signal, and providing a memory available (MA) signal. (This *DR* signal represents the only "architectural" difference between the original processor and the PLA-based design.)

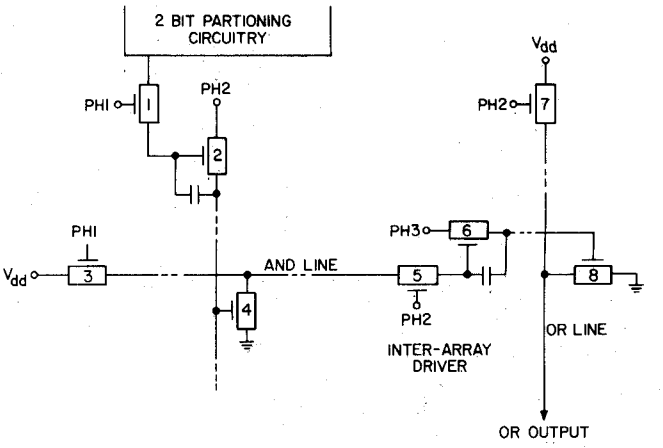


Fig. 2. Conceptual circuitry used in PLA's.

Physical Design

The basic element in the design was the PLA itself, including peripherals for 2-bit partitioning and bus driving. This circuitry went through a number of modifications. Initially and conceptually, a simple circuit such as that of Fig. 2 was envisioned. Both AND and OR arrays are shown. The circuitry is similar to that in [5] in its extensive use of dynamic circuits and bootstrap drivers. Two-device bootstrap drivers (device 1,2 and 5,6) actually drive the two arrays. During Φ1, the 2-bit partitioning circuit is gated onto the bootstrap driver gate (device 2) while the AND array is precharged through device 3. When Φ1 falls and before Φ2 rises, the 2-bit partitioning circuitry operates in a manner to be described to provide the proper level at the gate of device 2. During Φ2, the driver operates and conditionally drives the output of the AND array circuits to their final value. This final value is gated into the interarray driver (device 6) and the OR array precharged during Φ2 also. During Φ3, the interarray driver operates and the OR array outputs assume their final values. At the same time, buses are unconditionally precharged. During Φ4, drivers similar to the interarray driver operate to place the OR array output on the bus. Alternatively, pulse drivers can deliver control pulses from the OR array during either Φ4 or Φ1 of the subsequent cycle.

The drivers of Fig. 2 may not operate properly in a technology with significant gate to drain overlap, where substantial gate-to-drain capacitance may cause a false turn-on of the

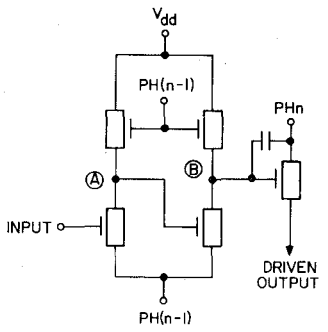


Fig. 3. Actual clamped bootstrap driver used throughout the design. The bootstrap capacitor is implemented by an MOS capacitor to minimize clamp requirements and yet provide high capacitance when a high output is to be driven.

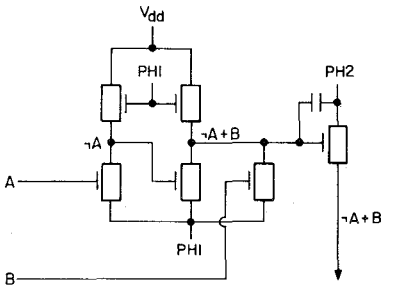


Fig. 4. Extension of basic driver into a 2-bit partitioning circuit for the PLA. Logic functions shown assume that a low (ground) level signifies logical 1, and are valid only at appropriate phase times (e.g., the driver output has the indicated value only during Φ2).

bootstrap device by coupling the phase pulse of the driver into the gate of the driver. To accommodate such a technology, the driver was redesigned, resulting in the circuit shown in Fig. 3. Here, two dynamic inverters precede the actual bootstrap driver device; the bootstrap capacitor of the driver is now connected from gate to drain, rather than from gate to source. Most of the bootstrap capacitance is formed between a metal electrode and an inversion layer under that electrode; the inversion layer is contiguous with the bootstrap device drain, and thus electrically connected to it. When the gate of the bootstrap device is at a high potential, the inversion layer is formed, and a high capacitance exists from gate to drain. The phase pulse on the drain of the bootstrap device couples into the gate and provides a very fast rising waveform at the driver output. Reasonable fall time is also achieved.

Conversely, when the gate of the bootstrap device is at a low potential, the inversion layer is not formed and coupling between gate and drain is much reduced. The double inverters provide added protection against this coupling. When the input to the first of the inverters is at a low level, both nodes A and B will precharge to a high level during $\Phi(n-1)$, but node A will remain high after $\Phi(n-1)$, while node B drops. Thus, the active device of the second inverter is on during Φ_n and serves to clamp the bootstrap device gate to ground. If the driver input is high, then as $\Phi(n-1)$ falls both node A and node B begin to fall. Node B generally is the more heavily loaded node, and it falls more slowly than node A. Use of proper width-to-length ratios of the devices and occasional inclusion of a padding capacitor at node B will ensure a high level at node B. In this case, at Φ_n, the bootstrap device gate is high and unclamped, and a pulse will appear at the driver output. The only modification that this driver makes in basic timing concepts is that the driver input must remain valid until after the output pulse has been completed. This restriction does not alter the basic timing described earlier.

The driver is quite flexible and is the basic circuit in the PLA's. The clamp action of the second inverter stage works properly provided its input does not go high during the output time of the driver. Thus, the first inverter may be omitted and the driver used as an inverting driver at an array output. The basic driver circuit may also be expanded into a complete 2-bit partitioning circuit as shown in Fig. 4. The basic circuit shown is repeated four times (with appropriate modification) to provide the signals needed. The only timing requirement here is that inputs to the array become valid during Φ1 and remain valid to the end of Φ2. Since PLA inputs are usually on buses (which become valid during Φ4 and remain valid to the beginning of Φ3), this again is no modification to already assumed timing.

The array circuitry was laid out in the ground rules as the original "random" processor. The most complex part of the layout was the 2-bit partitioning circuitry just described. Once this was done, the AND array device sizes were set to match the input line pitch of the partitioning circuits. The interarray driver was laid out and the OR array device size set to similarly match pitches. The circuits were then simulated under 2-sigma worst case conditions including skews and level shifts on the nominally 100 ns clock pulses. Skews and level shifts were taken as consistent with low cost bipolar clock drivers. Device parameters available to the designer (width-to-length ratios and padding capacitors in some circuits) were adjusted to achieve a working worst case circuit for maximally loaded array paths. When this was done, the PLA was considered complete and attention turned to other elements in the design.

Register Macros

A register macro (or one of several) was used for all long term data storage within the PLA-based processor. The macros vary in complexity from a simple D-type latch with a single data path to a more complex T-type latch with multiple data paths. In order to reduce power, all latches employed "pulsed power" in which load devices were turned on only during a single phase of the clock. The register macros were also designed to have a single read control line and a single write control line, to minimize wiring and control space.

D-Type Latch

The most frequently used register macro consisted of the D-type latch shown in Fig. 5. The latch was designed so that information could be read from the bus into the latch during Φ1 and written from the latch onto the bus during Φ4. When no read or write operation was requested, the latch will store information indefinitely, dissipating power only during Φ3, when the information in the latch is "refreshed" to compensate for any decrease in levels caused by leakage. Nominally, load devices dissipate power for only 12.5 percent of a clock cycle.

Information is conditionally read from the bus into the latch

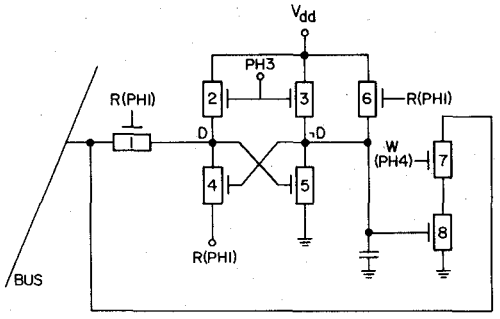


Fig. 5. Basic D-type latch used in PLA-based processor.

during Φ1. If a read control signal is generated, devices 1 and 6 will be turned on and the source of device 4 ($R\Phi 1$) which is normally low will also go high. Since Φ3 is low, devices 2 and 3 are turned off. Bringing the source of device 4 ($R\Phi 1$) high during a read operation makes it possible to transfer the information on the bus into the latch independently of the original state of the register. (One can see very quickly if this were not done and node $\neg D$ and the bus were both high during a read operation it is highly likely that the bus would be discharged through devices 1 and 4.) If the bus is high, node D will go high and node $\neg D$ will either remain low or be driven low through device 5. For the case where the bus is low, node D will go low (or remain low), since device 5 is now off node $\neg D$ will either remain high or charge to a high level through device 6. The WLR ratios of each of the devices have been carefully chosen under worst case conditions to meet these requirements. When the read control signal $R\Phi 1$ goes low the information which was previously on the bus is now stored in the dynamic latch, which is refreshed during Φ3 by pulsing load devices 2 and 3.

Information is conditionally written from the register onto the bus during Φ4. The write control line $W\Phi 4$ conditionally turns on device 7, causing the complement of $\neg D$ to be transferred to the bus which had been precharged during Φ3 through a device. Devices 7 and 8 have large WLR's since they are stacked and because of the possible high bus capacitance they must discharge.

A major driving force in the latch design was the minimization of the number of control wires. A much simpler latch design would be possible if two read control lines were used. This would allow the latch to be cleared during one phase and loaded during the next. The timing of the system overall appears to allow such operation, but it was not investigated in detail.

Toggle (and JK) Latch

The most complicated register macro consisted of the T type with multiple data paths shown in Fig. 6. The toggle register is used in connection with the IALU of the system, where incrementing is simplified by the toggle function. The toggle input to the register comes from the output of the IALU OR array. Information can also be loaded into the register from either of two buses.

The latch can be used either as a T or a JK. Truth tables for both modes of operation are given in Table I, where D and Dn

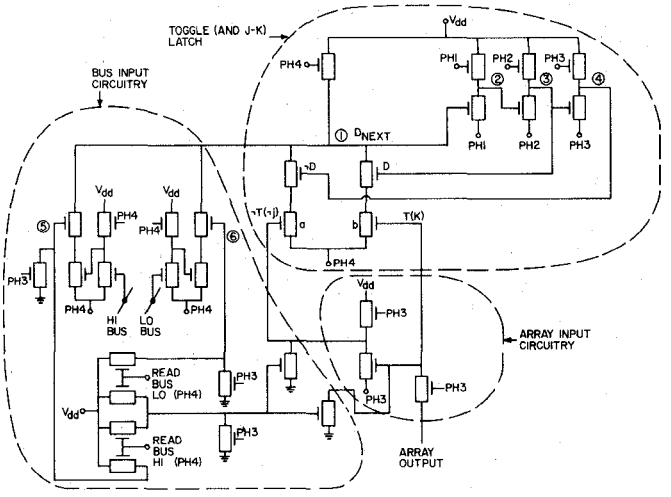


Fig. 6. JK/T latch with multiple input and output lines. This is the most complex single macro in the processor.

TABLE I
Latch Operation: T and JK Latches

¬T	T	D	¬D	Dn	¬Dn	
0	1	0	1	1	0	Toggle
0	1	1	0	0	1	Toggle
1	0	0	1	0	1	No change
1	0	1	0	1	0	No change

$\neg Dn = \neg T \neg D + TD$

(a) Toggle Latch (T)

¬J	K	D	¬D	Dn	¬Dn	
0	0	0	1	1	0	Set to 1
0	0	1	0	1	0	Set to 1
0	1	0	1	1	0	Toggle
0	1	1	0	0	1	Toggle
1	0	0	1	0	1	No change
1	0	1	0	1	0	No change
1	1	0	1	0	1	Set to 0
1	1	1	0	0	1	Set to 0

$\neg Dn = \neg J \neg D + KD$

(b) JK Latch Function

denote the datum stored in the register before and after the input is changed. The T register is really a subset of the JK as indicated in the table, so the latch operation will be described for the JK case. The table indicates that when no input is applied ($\neg J = 1, K = 0$) the state of the latch remains unchanged. When an input is applied to J only ($\neg J = 0, K = 0$) the latch either remains in the 1 state or switches to it. When an input is applied to K only ($\neg J = 1, K = 1$) the latch either remains in the 0 state or switches to it. When inputs are applied to both J and K ($\neg J = 0, K = 1$) the latch toggles to its complement state. At the end of Φ4 the new state of the latch is stored on node 1 of Fig. 6. By the end of the next Φ3 time this information has been transferred to node 3 and in complement form to node 4. Thus, it is available as shown to determine the next state of the latch during Φ4. During the time node 1 is changing, nodes 3 and 4 remain unchanged because of the isolation provided by the Φ1 "stage."

The array input circuitry as shown is used to generate the toggle signals (T and $\neg T$). If the latch were used as a JK, separate lines would be needed to generate $\neg J$ and K inputs to the latch.

The state of two buses (HIBUS and LOBUS) can be read into

the latch by means of the bus input circuitry. If READ.BUS.LO and READ.BUS.HI are not activated, nodes 5, 6, and 7 will be low during the up and down times of $\Phi 4$ and the input circuitry does not affect the state of the latch. However, if either READ.BUS.LO or READ.BUS.HI are activated the latch will take the state of the appropriate bus. For instance, if READ.BUS.LO goes high, nodes 6 and 7 will also go high. Input lines T and $\neg T$ will both go low turning off devices A and B. Since node 6 is high, device C is turned on and information on the LOBUS can be read into node 1 through the two-stage inverter circuit. This two-stage inverter is similar to the 2-bit partitioning circuitry previously described and has the same requirements for the discharge speeds of the stages.

Control Design

In a PLA, size implies delay. Therefore, the design of the control PLA emphasized a minimization of the number of words required to implement the control function. This was done manually by using two basic techniques. Where instructions could be made to have common sequences of steps this was done, and don't cares introduced as appropriate to allow one word of the control PLA to be shared among several instructions. Where, in addition, whole functions (such as I-Fetch) were used with sufficient frequency, the functions were coded as coroutines in the control PLA by adding a few input bits to the array to activate the coroutine, and by coding such coroutines to be invoked only by such added input bits, independently of the instruction being executed. In this way, it is possible to code such functions as I-Fetch only once in the control PLA, and yet to have the I-Fetch function run concurrently with other operations. These two methods, applied rather aggressively, led to a control PLA sufficiently small to meet the desired performance specifications.

In the end, two different control schemes were investigated. The first used a single control PLA as shown in the data flow, Fig. 1. Inputs to the control PLA came from the instruction register (INST), the ALU indicators (L, E, C), memory interface lines, and the interrupt interface. The rest of the inputs of the control are feedback lines from its own outputs which are used to implement a 2-bit counter (PLA cycles within a memory cycle), a microstep counter (sequencing of PLA cycles in an instruction), and a field of bits for activating the coroutines described earlier. The primary outputs of the control PLA consist of control wires for ALU, memory, IALU, buses, registers, etc.; most of these are pulsed output lines.

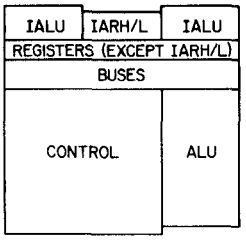
Each of the 45 instructions of the processor belongs to one of the several instruction groups such as byte instruction group, register instruction group, etc. Within a given group, portions of the microsteps are often the same for different instructions. To take advantage of the commonality, don't care bits can be introduced into the instruction field of the control so that one PLA word can be accessed by more than one instruction. Furthermore, certain actions are common among different instruction groups, such as ADD and SUBTRACT, which occur in both the register instruction group and the byte instruction group. In addition, the instruction fetch for the next instruction is common to all instructions and actually is performed while the last several steps of the instruction are performed. Hence, the concept of coroutine is adopted in which a group of PLA words are designated to accomplish a certain task such as ADD or I-FETCH and the coroutine field in the input is used to call or activate them. Note that the I-Fetch coroutine is called to run concurrently with the regular instruction tasks during the last memory cycle of a given instruction. This kind of parallelism certainly contributes to the fact that the PLA-based processor requires only two memory cycles for the BRANCH and JUMP macroinstructions, while three memory cycles are required in the original processor.

TABLE II
COMPARISON OF PERFORMANCE OF RANDOM AND PLA-BASED MICROPROCESSOR. TABLE SHOWS TIME IN MEMORY CYCLES PER INSTRUCTION. FOR THE CASE OF JUMP AND BRANCH INSTRUCTIONS, TIMES SHOWN ARE TAKEN/NOT TAKEN

OPERATION	INST	Original	PLA Based	
Add register	AR	3	3	
Subtract register	SR	3	3	
Load register	LR	3	3	
Store register	STR	3	3	
Load, decrement	LRD	5	5	
Load, increment	LRB	5	5	
Add byte	AB	3	3	
Subtract byte	SB	3	3	
Load byte	LB	3	3	
Store byte	STB	3	3	
Compare byte	CB	3	3	
And byte	NB	3	3	
Or byte	OB	3	3	
Xor byte	XB	3	3	
Add immediate	AI	2	3	+1
Subtract immediate	SI	2	3	+1
Load immediate	LI	2	2	
Compare immediate	CI	2	2	
And immediate	NI	2	2	
Or immediate	OI	2	2	
Xor immediate	XI	2	2	
Group immediate	GI	2	2	
Add 1	A1	2	2	
Subtract 1	S1	2	2	
Shift left	SHL	2	2	
Shift right	SHR	2	2	
Clear	CLA	1	1	
Transpose	TRA	1	2	+1
Input carry	IC	1	1	
Store indirect	STN	4	4	
Load indirect	LN	4	4	
Test and reset	TR	1	1	
Test and preserve	TP	1	1	
Input	IN	4	4	
Output	OUT	4	4	
Jump	J	3	2	-1
Jump on not =	JNE	3/1	2/1	-1/0
Jump on =	JE	3/1	2/1	-1/0
Branch	B	3	2	-1
Branch on not =	BNE	3/2	2	-1/0
Branch on =	BE	3/2	2	-1/0
Branch on high	BH	3/2	2	-1/0
Branch on low	BNL	3/2	2	-1/0
Branch and link	BAL	6	5	-1
Return	RTN	5	4	-1
Interrupt	INT	10	10	

The memory cycles required for executing the instructions of the original and PLA-based processor are given in Table II on a per instruction basis. In general, the overall performance of the two processors is identical.

The PLA to implement the machine control was completely implemented, including all 45 instructions and interrupt processing, with 137 PLA words. Control inputs consisted of 14 primary inputs and 10 feedback inputs. The AND array of the control PLA is 48 bits (devices) wide. For the OR array, the 10 bits of feedback wires are also doubled (set and reset) to re-



CONTROL	23786
ALU	10662
IALU	5974
REGS	4143
BUSES	6000
TOTAL	50565
ORIGINAL	44521
DIFF	6044

Fig. 7. Overall diagram of chip layout using a single control PLA. The subdivision of the control PLA into several smaller PLA's results in the reduction of total area at the expense of a somewhat more complex wiring problem.

quire 20 PLA outputs; in addition, there are 76 control lines used to activate various transfer paths in the rest of the system. The total control PLA is therefore 137 words × 144 bits, or 19 728 bits total.

Subsequent effort was made to divide up the single control PLA into several small PLA's to minimize the total bit count in the control, and hence save chip area. The single control PLA was divided into three sections and minimized using the programs of Hong *et al.* [3]. Three subdivided control PLA's were obtained which perform the identical function as the original single array. Sizes of these arrays (words × bits) are 59 × 77, 48 × 93, and 36 × 124, for a total of 13 471 bits. This subdivided control thus reduces the total control bit count by 32 percent.

Chip Layout

Fig. 7 shows the overall layout of the PLA-based processor chip using a single control PLA. The PLA and register areas in the layout have been calculated from detailed layouts of the register latches and PLA circuits. In this overall layout, the IALU PLA has been subdivided into two sections, each having half of the words of the original IALU. There is a small overhead penalty paid for this because of the necessity of duplicating some of the 2-bit partitioning circuits; the sectioning achieves a simpler overall layout.

Wiring space in the area between the registers and the major PLA's is estimated from the known number of data and control wires: it was assumed that a single wiring space could serve two control wires, as most control wires merely go to different registers. The subdivided control design was not completed in detail. An examination of the area saved (in control bits not required) and lost (again, the repetition of partitioning circuits) suggests that the overall chip area will be reduced by from 10 to 15 percent.

Power

Chip power for all dynamic circuits was calculated assuming full voltage swings on all active lines at a 1.25 MHz rate. The PLA's were all considered active whether they were serving a useful function or not. Thus, the ALU PLA is dissipating power even when its outputs are not gated to any bus; a modification of the drivers used to clock the PLA's can provide a mode of operation in which all unused PLA's are turned off. This type of circuitry will provide its greatest advantage in the design with the subdivided control PLA, where unused control functions may also be turned off.

TABLE III
Summary of Results

PARAMETER	Original	PLA based	Relative Utilization
Speed (us/cycle):	3.2	3.2	1.00
Power (mW):	400	131	3.05
Area (sq.mil):	44521	50565	0.88

Summary of Results

A summary of the results of this study appear in Table III. This table compares the power, speed, and density of the original processor with the PLA-based processor. The "relative utilization" compares the two designs, the original being used as a reference. Normalization is such that a large number indicates a better use of the appropriate aspect of the technology. The result is extremely exciting: the PLA-based processor exhibits nearly the same characteristics as the original design in all aspects except power, where it is significantly superior. The results are even more exciting when it is realized that they do not represent the limit of what can be done with PLA-based macros. Mention has been made already of the possibility of subdividing the control PLA to achieve higher density, and modifying circuits to turn off unused PLA's to reduce power. These techniques can also impact speed by reducing capacitance. They have not been explored in detail in this study.

It must be emphasized that these results are derived from a specific low-end test design. It is not clear how the results apply in a performance dominated environment. In the low end, however, it is evident that the PLA-based macro design concept can successfully compete with more conventional random logic approaches.

Acknowledgment

The authors wish to thank D. Lee, IBM Office Products Division, for his substantial help in mapping the ideas inherent in this study into the MOSFET technology of the original design. Without this "real world" base, the study would have remained purely academic. The leadership and encouragement of D. Critchlow is also acknowledged.

References

[1] H. Fleischer and L. I. Maissel, "An introduction to array logic," *IBM J. Res. Develop.*, vol. 19, p. 98, 1975.
[2] J. C. Logue *et al.*, "Hardware inplementation of a small system in programmable logic arrays," *IBM J. Res. Develop.*, vol. 19, p. 110, 1975.
[3] S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.*, vol. 18, p. 443, 1974.

[4] P. W. Cook, D. L. Critchlow, and L. M. Terman, "Comparison of MOSFET logic circuits," *IEEE J. Solid-State Circuits*, vol. SC-8, p. 348, 1973.
[5] R. A. Wood, "High-speed dynamic programmable logic array chip," *IBM J. Res. Develop.*, vol. 19, p. 379, 1975.

Peter W. Cook (S'61-S'70-M'71), for a photograph and biography, see page 255 of the April 1979 issue of this JOURNAL.

Chung W. Ho (S'63-M'66) received the B.S. degree in electrical engineering from the National Taiwan University, Taipai, Taiwan, China, in 1961, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, in 1963 and 1966, respectively.

He joined the staff of the Electronics Research Laboratory, University of California, Berkeley, in 1963. In 1966, he was an acting Assistant Professor in the Department of Electrical Engineering, University of California, Berkeley. In 1967, he became a member of the Research Staff, Thomas J. Watson Research Laboratory, IBM Corporation, Yorktown Heights, NY, where he is currently Manager of Semiconductor Packaging Technology. His current interests include computer hardware studies, semiconductor technology, and computer aided circuits analysis and design.

Dr. Ho received the IEEE Best Paper Award, Syracuse, NY chapter, in 1967, and the IBM Outstanding Contribution and Invention Achievement Awards, in 1975 and 1977, respectively.

Stanley E. Schuster (S'61-M'65), for a photograph and biography, see page 267 of the April 1979 issue of this JOURNAL.

The Punchthrough Device as a Passive Exponential Load in Fast Static Bipolar RAM Cells

JAN LOHSTROH

Abstract—**It is shown that the punchthrough device can be used as a passive exponential load in fast static bipolar RAM cells. The advantages are that the cell standby/read current ratio can be very large (> four decades), and that the cell area can be very small due to the fact that the punchthrough load is a vertical device. In this cell, a very small standby power dissipation (<0.1 μW) is combined with a short access time (<10 ns). A static noise margin calculation for the cell is included. The general standby/read switching behavior of memory cells with exponential loads is explained. The cell sensitivity to α-particles is discussed.**

I. INTRODUCTION

NONLINEAR load devices in static bipolar memory cells allow a relatively large sense current compared with the standby current. In such cells, a fast address access time and a short write time can be combined with a reasonably low standby power dissipation in the cells. In recent years, several approaches to passive nonlinear load devices have been developed. In a favorite method, a load resistor is bypassed, either with a slow silicon p-n diode (Rathbone *et al.* [1]) or with a fast Schottky diode (Mukai *et al.* [2]). In these cells, with resistors and diodes in parallel as loads, the ratio between read current and standby current can be 10-20 (Hotta *et al.* [3]).

Manuscript received April 10, 1979; revised June 16, 1979.

The author is with Philips Research Laboratories, Eindhoven, The Netherlands.

Another approach uses a switched resistor load by introduction of a series resistor with the bypass Schottky diode; in that case, the ratio between read current and standby current can be 500 (Inadachi *et al.* [4]).

In an earlier publication, the same switching action was obtained with an extra n-p-n transistor to achieve a ratio of 200 (Taniguchi *et al.* [5]).

In all approaches, the cell standby current is determined by the value of a resistor. The resistance value has to be accurate to achieve a well-defined noise margin for a given cell current. If standby currents below 1 μA are required, the resistance must be at least 250 kΩ. Accurate resistances of that high a value can be made by ion implantation, but take a relatively large silicon area. Polysilicon resistors can be used (Okada [6]); they can be made very small, but their accuracy decreases with increasing sheet resistances (de Groot [7], Seto [8]). An easier solution is to increase the nonlinearity of the loads to a pure exponential characteristic with a nonideality factor m larger than 1 [9].

In that case, read/standby current ratios of four to six decades can be realized. This will be the approach of this paper.

It must be stated that large read/standby current ratios can also be realized with current sources (p-n-p's, for instance) as loads (Wiedmann and Berger [10]-[12]), but for these cells it is essential that one of the n-p-n driver transistors goes into

0018-9200/79/1000-0840$00.75 © 1979 IEEE