7 **Rapid #: -8177647**

Odyssey
IP: 128.210.126.171/ILL

Status	**Rapid Code**	**Branch Name**	**Start Date**
New	IPL	Main Library	7/15/2014 6:51:22 AM
... note: only the last 3 transactions are shown below. view details			
Unfilled	PAU	Storage	7/16/2014 11:29:53 AM
Pending	IQU	CSEL	7/16/2014 11:30:05 AM
Batch Not Printed	IQU	CSEL	7/16/2014 12:12:12 PM

CALL #:	**TK7874 V558**
LOCATION:	**IQU :: CSEL :: sper**
TYPE:	Article CC:CCL
JOURNAL TITLE:	VLSI design
USER JOURNAL TITLE:	VLSI Design
IQU CATALOG TITLE:	VLSI design
ARTICLE TITLE:	The MacPitts Silicon Compiler: A View from the Telecommunication Industry
ARTICLE AUTHOR:	Fox, J.R.
VOLUME:	
ISSUE:	
MONTH:	May/June
YEAR:	1983
PAGES:	30-32 36-37
ISSN:	0279-2834
OCLC #:	IQU OCLC #: 7719803
CROSS REFERENCE ID:	[TN:1199076][ODYSSEY:128.210.126.171/ILL]
VERIFIED:	

BORROWER: **IPL :: Main Library**

RAPID ILL

The MacPitts Silicon Compiler: A View From the Telecommunications Industry

Jeffrey R. Fox, GTE Labs, Inc. Waltham, MA

MacPitts is a "silicon compiler" program that lets users design VLSI circuitry by specifying the circuit algorithmically in a high-level LISP-like language. The compiler generates the logic needed to implement a given function in the MacPitts target architecture, and produces two levels of output. The first is a technology-independent network listing, and the second is a CIF description that is compatible with the ARPA/MOSIS nMOS depletion-load process.

The MacPitts target architecture divides the chip into data-path and control sections. The input language and the data-path organization allow a high degree of parallel processing, which can increase circuit performance in high-speed signal-processing applications. Several telecommunications-oriented circuits, implemented earlier using traditional logic-design techniques with an nMOS standard-cell library, were redesigned with MacPitts; this article reports the design time and the chip size for each method. Some disadvantages of MacPitts for these applications were discovered, and several changes were made in the program to improve performance for telecommunications circuits.

The control section of MacPitts is implemented in a Weinberger-type NOR gate array. Relaying out the control-logic NOR functions in a standard-cell array saved a lot of space, and may have increased the speed of the circuit.

A Brief Overview

MacPitts (Siskind *et al.* 1982) was developed at the MIT Lincoln Laboratories by Siskind, Southard and Crouch. It accepts high-level LISP-like (Foderaro and Sklower 1981) algorithmic description of a circuit, and generates the logic circuitry needed to implement the specified function. MacPitts first produces a technology-independent network listing in its target architecture; *i.e.*, a chip consisting of a data-path and control-section and finite-state-machine sequencers. The silicon compiler can then use this network listing to lay out the circuit, and produce a CIF geometric description of the entire chip that is compatible with the 5-μm nMOS ARPA/MOSIS process. The fixed floorplan of MacPitts casually resembles that of a conventional microprocessor. Unlike conventional microprocessors, however, MacPitts lets several special-function sub-data-paths, control sections, and sequencers be compiled and combined to produce a chip that can perform any number of operations in parallel. These sub-processes communicate with each other, in a limited way, through internal signals. The MacPitts data path consists of functional "organelles" and registers. (The term "organelle" refers to a single bit slice of a register or operator.) These organelles operate on integer-type data, with functions such as *add, increment, word-and,* and *test-equality.* Data is stored in registers that are incorporated into the data path.

The control logic is implemented with an arbitrary depth of NOR gates in a regular array of wires and transistors called a Weinberger array (Weinberger 1967). Sequences are really mini-data-paths, with registers and incrementers combined to implement state counters.

Applicability of MacPitts to Telecommunications Circuitry

General Considerations

Many telecommunications applications involve encoding, processing, and decoding high-speed serial bit streams. The decoding process often involves clock recovery, error detection and correction, and serial/parallel/serial conversion. The encoding process usually involves coding according to an error-correction scheme, rate, and format conversion. Loop-back circuitry is often needed for testing. These algorithms can exploit functional parallelism and would seem amenable to a MacPitts implementation. We recognized from the outset that the present MacPitts program provides only for a single, synchronous clocking system, which would not permit the independent clocking of encoder/decoder pairs (as is often required in telecommunications). Because this difficulty was a MacPitts implementation problem, and not a theoretical roadblock, we ignored it in order to get a sense of the basic architectural issues.

Limited Storage Capability

MacPitts handles storage much the same way a general-purpose Von Neumann computer does. In a computer, results are stored either in main memory or in one of several special-purpose registers. Any operations that must be performed on this data require that the contents of the storage element be read, operated on by the machine's arithmetic and logic unit (ALU), and restored to a memory location or register. MacPitts

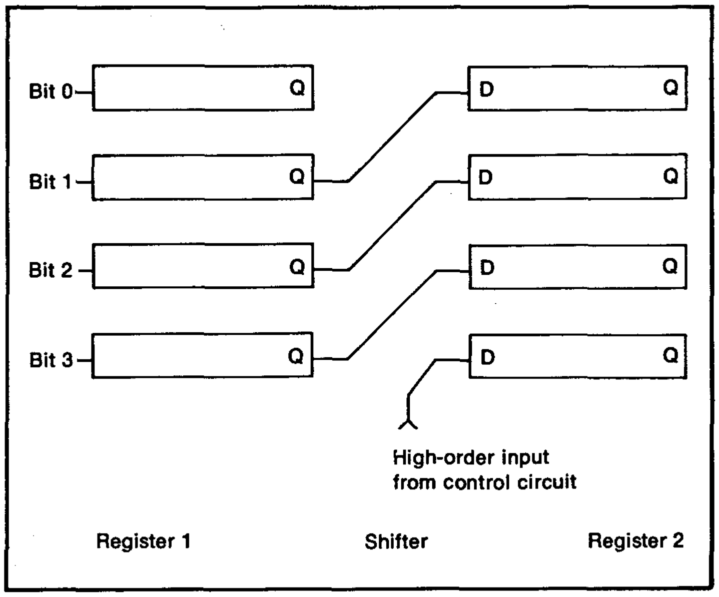
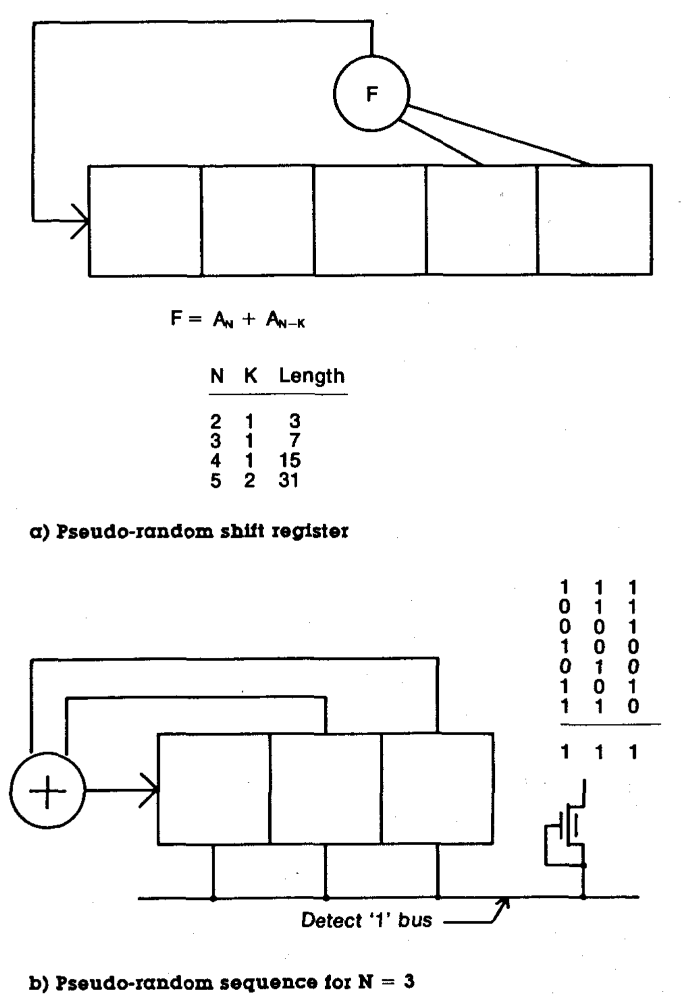


FIGURE 1. A bit-displacement shifter.



$F = A_N + A_{N-K}$

N	K	Length
2	1	3
3	1	7
4	1	15
5	2	31

a) Pseudo-random shift register

b) Pseudo-random sequence for N = 3

FIGURE 2. Pseudo-random counting.

stores all integer data in a special-purpose, globally named and addressable element called a "register." Operators in MacPitts are not limited to a single, general-purpose ALU.

MacPitts provides a library of optimized special-purpose integer arithmetic operators called "organelles." These organelles may be word-wide logical functions such as AND, OR, or XOR; arithmetic functions such as *increment, add,* or *subtract*; or fixed, hard-wired "bit-displacement" shifters. Figure 1 shows a bit-displacement shifter. MacPitts will instantiate only those organelles that are needed to perform the user's specified function.

The user can specify to MacPitts that several independent processes are to be executed in parallel. Each process causes the independent instantiation of all the organelles it needs to execute the specified algorithm. Because processes are executed in parallel, there can be no interprocess sharing of operator organelles. Within a process, however, MacPitts will automatically share operator organelles, to the extent permitted by the user's algorithm. Thus, if an adder is required in two distinct states in one process, only one adder organelle need be instantiated. The adder will be shared by the use of multiplexer input transistors.

Many telecommunications circuits require high-speed rate-conversion, event counting, and frequency division. A data-path architecture that only permits direct addressing of a general-purpose storage register precludes the use of an addressable, efficient, autonomous counter function that could otherwise be used in these applications. For example, to perform a modulus count, a register must be addressed. The register's contents must be fed to an incrementer organelle, tested for equality with the specified modulus by an equality-tester organelle, and then restored to the register. Extending the MacPitts storage concept to include the notion of a "functional state" would enable performance improvement of one order of magnitude in many telecommunications applications.

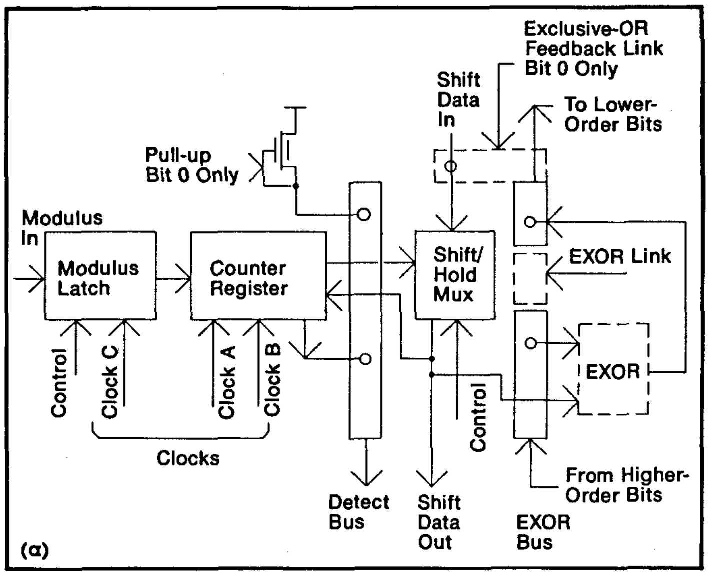
The Functional State

The concept of a functional state refers to the incorporation of a logical function into a storage device. The functional storage element would thus be responsible for both maintaining the state of a calculation and executing the necessary function. High performance could be achieved by eliminating the need to read, operate on, and re-write the data. The counting operation (required in many high-performance telecommunications circuit functions) is one example in which functional state could be applied in MacPitts, and would yield a significant increase in performance. Nevertheless, the original MacPitts only permitted storage in a single type of global register.

Event counting and rate-conversion can be implemented efficiently using a pseudo-random sequence in a polynominal-feedback shift register. A shift register is a particularly efficient, simple way to count, because no combinatorial logic exists between shift-register stages. The only combinatorial function required is the proper feedback expression. A pseudo-random counter is completely synchronous, has no ripple-through, and can be any desired length. It is often an order of magnitude faster than the original MacPitts register-adder counting method. A shift register with irreducible polynominal feedback can count a maximum of 2^N-1 states, where N is the number of shift-register bits. Figure 2 shows a typical pseudo-random counter and its sequence.

A pseudo-random counter capability was added to MacPitts, thereby introducing the concept of autonomous functional state to the data-path.

Counters operate during all MacPitts processes. Each counter can be synchronized to a specific state in the algorithmic flow of a chip, and has a dynamically programmable



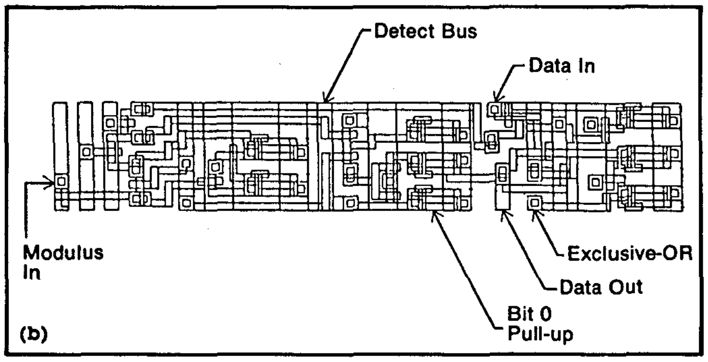


FIGURE 3. (a) Block diagram of a pseudo-random counter organelle. The EXOR circuit only exists on bits with feedback taps; on all other bits, the EXOR link is instantiated. (b) Pseudo-random counter organelle (30 x 176 lambda).

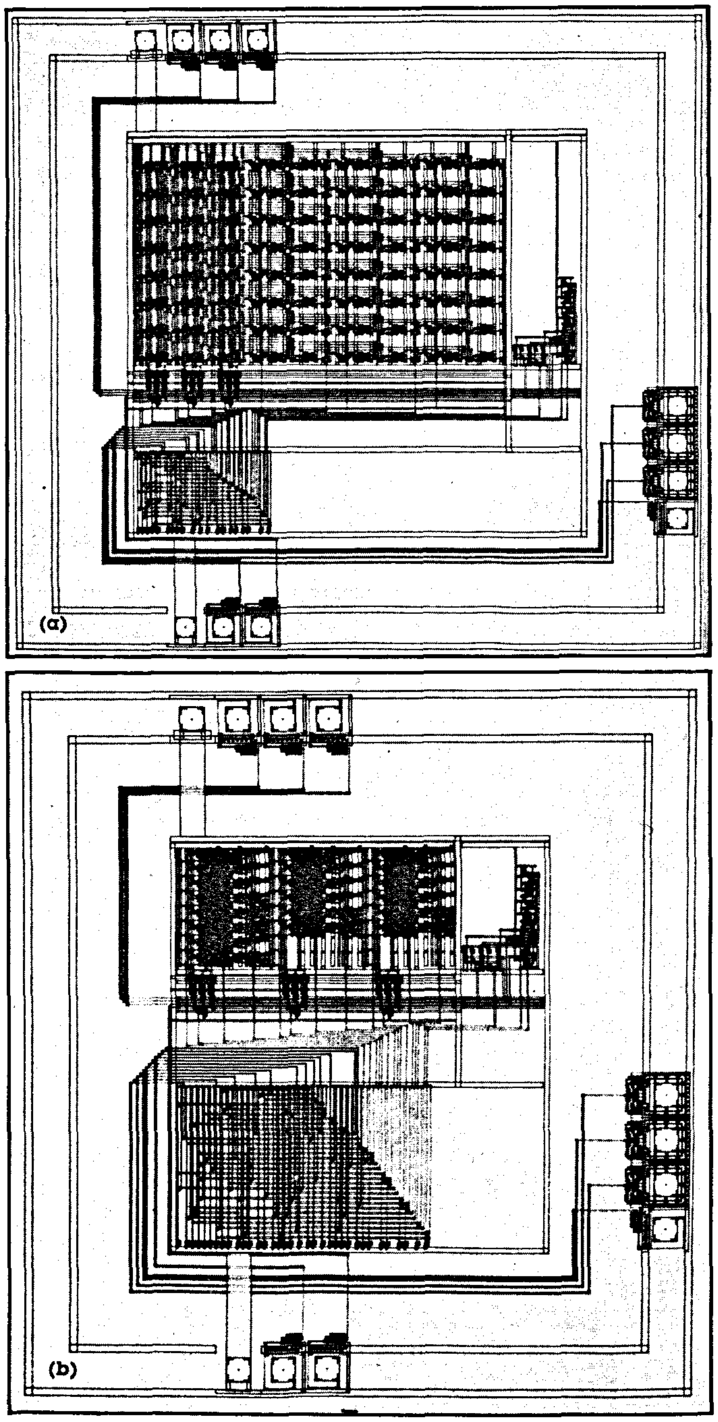


FIGURE 4. (a) Clock chip with registers and adders. (b) Clock chip with counters.

modulus. The user must give each counter a unique name. Each counter's name and maximum modulus (consistent with the data-path width) is defined with a MacPitts <def> statement, in much the same way registers are defined. MacPitts calculates the number of bits required to count the maximum declared value, and only instantiates the number of counter stages necessary—thus saving power.

The MacPitts library consists of a LISP procedural definition of the organelles. The original MacPitts library allowed the conditional instantiation of organelles exclusively as a function of the bit number. This capability was extended to let the organelle description calculate its instantiation form based on information passed to it by the user program. Thus, the counter organelle could calculate the number and position of exclusive-OR taps required for a given pseudo-random counter.

The counter is activated by a <sync> statement in the body of the circuit's algorithmic description. The <sync> statement specifies the counter's name, the input signal that will cause the counter to advance when it is a logic one, and the current modulus. MacPitts calculates the proper pseudo-random state to which to preset the counter, so that after <modulus> counts, the pseudo-random counter reaches all logic ones. This causes a wired-OR "detect" bus to notify the control logic that the counter has reached terminal condition. This in turn causes the re-loading of the current modulus to continue the count. Any number of <sync> statements (with or without the same modulus) can be used with one counter. Entering a state with a <sync> statement immediately sets the named counter to the specified modulus, regardless of the current state of the counter. Thus, counters are a semi-autonomous element, instructed to count and to signal according to the periodicity of the modulus. Figure 3 shows the counter organelle. Figure 4(a) is a clock chip implemented with registers. Figure 4(b) is the same clock chip, implemented with counters. The speed improved by a factor of ten.

Many organelles in the original MacPitts library were redesigned and laid out for speed and power optimization. The new library makes much wider use of wired-ORing and MOS complex-logic gates. For example, Figure 5 shows the original

continued on page 36

MacPitts Compiler continued from page 32

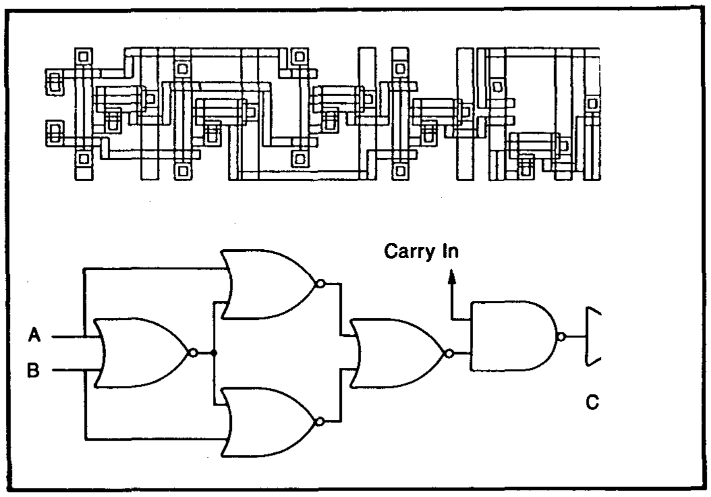


FIGURE 5. Original MacPitts equality organelle (31 x 150 lambda).

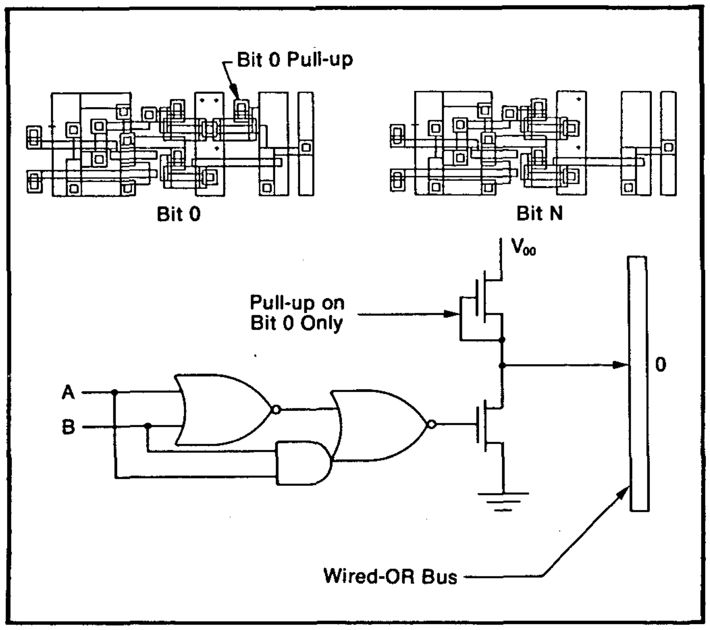


FIGURE 6. Wired-OR based equality organelle (28 x 80 lambda).

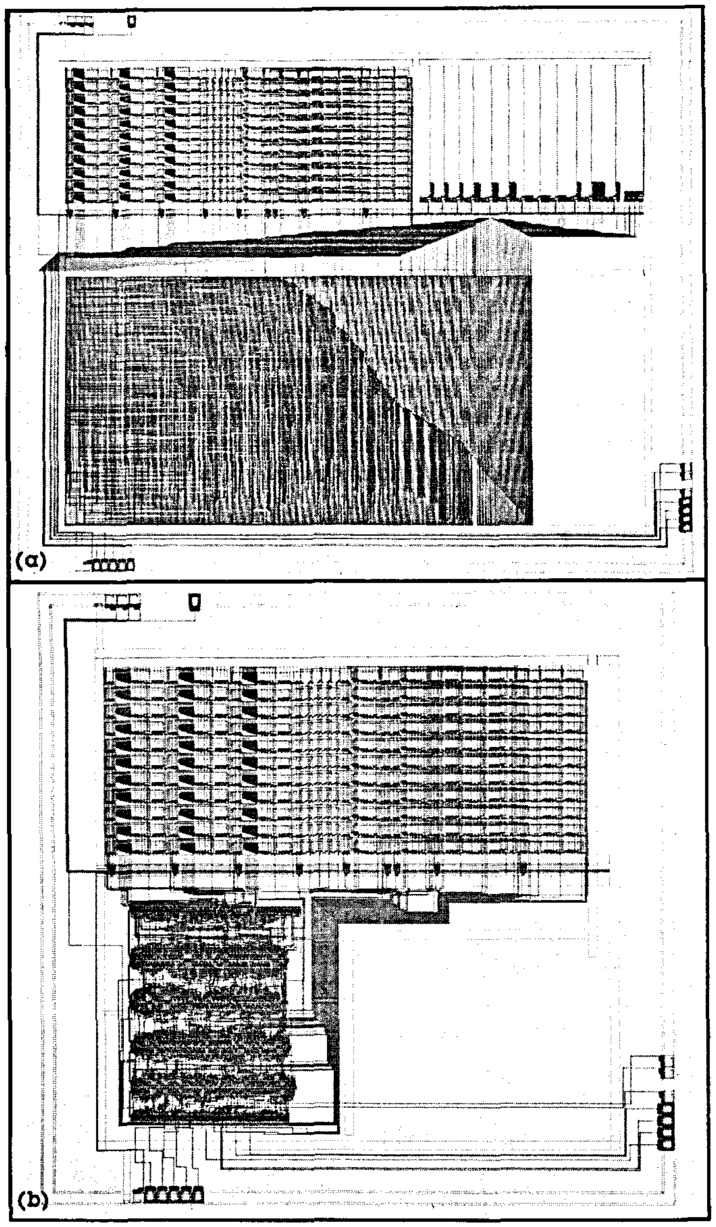


FIGURE 7. (a) DTMF3 receiver chip with Weinberger control layout. (b) DTMF3 receiver chip with FAMOS control layout.

word-equality-tester organelle. The number of logic levels is approximately two times the datapath width. Figure 6 shows the new wired-OR complex-logic version. It consists of three levels of logic, irrespective of word length.

Optimizing the Control-Logic Layout

A Weinberger array (Siskind *et al.* 1982, Weinberger 1967) is a regular architecture for laying out a collection of NOR gates having arbitrary input size. It consists of a set of pull-ups connected between V_{DD} and vertical wires that extend the length of the array. These wires form NOR-gate outputs. Adjacent and parallel to these output wires are ground wires that also run the length of the array. Input wires (made of a suitable material) enter the array vertically. Each NOR-gate input is formed by a pull-down transistor connected between an output wire and a ground wire, whose gate is connected to either an array input or the output of another NOR gate. Internal horizontal straps form the interconnections between gates. A Weinberger array can be synthesized very easily with *a priori* knowledge of its full size. Starting in one corner of the array, gates can be added by extending the array towards the opposite corner. The array grows horizontally with the number of outputs and inputs, and vertically with the number of horizontal interconnections between gates and NOR-gate inputs. This process can be continued to implement any number and complexity of NOR gates.

We investigated the alternatives to the Weinberger array. A MacPitts-designed and-laid-out version of the DTMF3 (Touchcall® receiver) chip was compared to the MacPitts-designed FAMOS (Schuler and Ulrich 1972, Mattison 1972) control layout of the same logic. Table 1 summarizes the results of this comparison, which is shown in Figure 7.

The data in Table 1 deserve explanation and comment. The standard cells and layout of the DTMF3 were done using an nMOS depletion-load 5-micrometer cell library and standard computer-aided logic-design techniques. Placement and routing were done by the FAMOS standard-cell layout program. (FAMOS is a standard-cell placement and routing program used at GTE Laboratories to lay out MOS integrated circuits.) It places arbitrary-sized cells in rows according to the strength

Name	Standard-cell design and layout	MacPitts design and layout	MacPitts design and data-path layout; standard-cell control layout	Control Only	
				Weinberger array	Standard cells implementing MacPitts NOR logic
HOB3 CODEC	12,535mils²	93,564mils²	61,020mils²	26,693mils²	7,737mils²
DTMF3 Dual-tone multi-freq-quency	36,660mils²	492,929mils²	265,684mils²	153,950mils²	25,948mils²*
	*Includes sequencers and flags				

TABLE 1. Size comparison of design and layout methodologies.

of their interconnections, and wires them according to Hightower's algorithm. The data paths of the MacPitts chips used in this experiment were entered as cells in the FAMOS library, so that FAMOS would automatically wire the existing I/O points. We wrote a computer program to convert the control-logic portion of the technology-independent MacPitts output to FAMOS format.

The total design and layout time was six man-months. The MacPitts design and layout is functionally equivalent to the FAMOS version; however, the MacPitts logic design is totally different. The algorithmic system description (which already existed for the standard-cell version) was entered into MacPitts. The total time required to translate the algorithm description to the MacPitts language, enter it, and obtain a plot was eight man-hours. If the time required to generate the algorithm description is included, the total MacPitts design time would be about five man-days.

The MacPitts design and data-path layout, in combination with a FAMOS standard-cell control layout, was produced by using the technology-independent network listing produced by MacPitts, extracting the control section, converting the listing to the FAMOS input format, and implementing the exact logic of the Weinberger array with standard cells. The MacPitts data path was then wired to the standard-cell control layout. The resulting chip size is somewhat deceptive. As shown in Figure 7(b), the control section is roughly square; there is much wasted space to the right of it. Ideally, the control layout should have consisted of two or three rows of cells exactly the width of the data-path. There is no real reason why it was implemented as shown. FAMOS, however, being ten years old, was not able to produce a circuit wide enough to match the data-path width. (This was due to array-overflow problems in the unstructured FORTRAN code.) A more up-to-date program would be able to produce the desired layout, resulting in a one-third decrease in the total chip area shown in column 3. To obtain a better comparison between the Weinberger array and the standard-cell layout the last column lists the areas of the exact logically equivalent control sections.

Although the Weinberger implementation shows a larger area than the standard-cell version, this is not a fundamental problem with Weinberger arrays. The Weinberger array used in this version of MacPitts was totally non-optimized. The only ordering performed allowed river-routing between the control section and the data path. This technique would create inordinately complex wiring—much larger than warranted. Furthermore, all processes were consolidated into a single Weinberger array, without regard for their degree of interconnectedness. A considerably more efficient control structure would result if:

a) the gates inside the Weinberger array were ordered according to their strength of connectivity, with channel routing between control and data-path sections; and
b) the Weinberger array were partitioned into multiple arrays (one Weinberger for each process). This would avoid the phenomenon of growth as the square of the number of gates.

Conclusion and Prospects

MacPitts drastically reduced the time required to design a complex LSI/VLSI circuit. For example, the standard-cell version of the DTMF3 took six man-months, but the MacPitts version took only eight hours. The LISP-like input language is quite comfortable and expressive for hardware design, and is certainly more useful for describing the function of a chip than a network listing is. Technology-independent output will greatly increase the applicability of MacPitts as new technologies are developed. At this writing, there is no way to specify real-time performance limits for the compiler. That problem is currently being addressed. With the addition of parallel-to-serial conversion semantics, multiple clocking, and performance features, MacPitts and its successors show promise of being *the* integrated-circuit design methodology for the VLSI era.

Acknowledgments

The author thanks the co-developers of MacPitts—J. Siskind, J. Southard, and K. Crouch of MIT Lincoln Laboratories—for making the program available to the MIT visiting-professional program. The author also acknowledges the contributions of MIT visiting professional A. Margolies, who designed several telecommunication circuits with MacPitts, and C.W. Hurlin, who re-laid out the entire organelle library. The author also thanks Professor Jonathan Allen of MIT, whose support and efforts made this work possible. Finally, the author acknowledges the assistance of E. Goldman of GTE Laboratories, who helped prepare this manuscript. □

References

Foderaro and Sklower. September 1981. *The FRANZ LISP Manual.*

Mattison. June 1972. "A High-Quality, Low-Cost Router for MOS/LSI," *Proceedings of the 9th IEEE DA Workshop*, Dallas, TX

Peterson, W.W. 1961. *Error-Correcting Codes*, MIT Press.

Schuler and Ulrich. June 1972. "Clustering and Linear Placement," *Proceedings of the 9th IEEE DA Workshop*, Dallas, TX

Siskind, Southard, and Crouch. January 1982. "Generating Custom High-Performance VLSI Designs from Succinct Algorithmic Descriptions," *Proceedings of the Conference on Advanced Research in VLSI*, MIT.

Weinberger, A. December 1967. "Large-Scale Integration of MOS Complex Logic: A Layout Method," *IEEE Journal of Solid State Circuits*, Vol. SC-2.

About the Author

Jeffrey Fox received the B.E. and MSEE from SUNY at Stony Brook in 1973 and 1974, respectively. He is currently Principal Investigator of the VLSI Design Project in the Physical Design Automation Department of the Computer Science Laboratory at GTE Laboratories, Inc. This work was performed while he was a visiting professional in the Microsystems Program at MIT. Jeffrey is a member of IEEE and Sigma Xi.