Design and Application of an Optimizing XROM Silicon Compiler

RICHARD W. LINDERMAN, MEMBER, IEEE, PAUL C. ROSSBACH, MEMBER, IEEE, AND DAVID M. GALLAGHER

Abstract—**This paper demonstrates that optimization techniques incorporated within a silicon compiler for read-only memories (ROM's) can achieve significant yield, power, and speed improvements by minimizing the number of transistors, drains, and metal interconnections in the ROM. Transistor minimization adopts a heuristic solution to the NP-complete graph partitioning problem with a powerful technique applicable to various ROM design styles and technologies. If diffusion mask personalization is permitted, the design can be further improved by solving the traveling salesman problem to minimize transistor source/drain regions. In table look-up ROM's compiled for 3- and 1.2-μm CMOS with diffusion mask programming, the compiler eliminated over 45 percent of the transistors and drains. Tested results show 3-μm CMOS ROM's have access times between 50 and 70 ns. ROM's with 1.2-μm features achieve simulated access times below 20 ns. A simple interface allows the optimizing compiler to easily work with other CAD tools such as microcode assemblers.**

I. Introduction

READ-ONLY MEMORIES (ROM's) provide nonvolatile storage of information on VLSI chips and are commonly used for microcode stores and table look-up. ROM's can be implemented on separate chips or as macrocells of larger processors. Constructing ROM's of any appreciable size requires a silicon compiler to assure design correctness. A silicon compiler also provides rapid turnaround on the ROM design. This is especially valuable when the ROM contents are subject to change during the chip design process, as is the case when microcode development proceeds in parallel with chip design. Finally, a compiler improves designer productivity by raising work to a higher level of abstraction.

This paper discusses a compiler that was initially targeted just to perform automatic layout, but later grew to also optimize the ROM to reduce the number of transistors, drains, and metal interconnections required. Opportunities for optimization derive from the regularity of a ROM which affords the designer several degrees of freedom. If the ROM address decoder is realized as the AND plane of a programmable logic array (PLA), the rows of the ROM can be arranged in any desired order. Similarly, the ROM outputs can be scrambled in any order, and flipped around the vertical axis if desired. Finally, the assignment of address lines to the decoders is flexible. The compiler accounts for all these possibilities, within constraints imposed by the user, and places the bits at the proper positions in the scrambled array.

Several optimization criteria are available. To obtain speed, power, and yield improvements, minimizing the number of transistors physically within, or connected within, the ROM is given top priority. If diffusion programming is allowed, only the necessary transistors will be placed within the ROM. If metal or contact personalization is used, the ROM can be reprogrammed later in the fabrication sequence, but a transistor must be fabricated for each ROM bit. In the latter case, we would minimize the number of "connected" transistors rather than the number of fabricated transistors. In either case, critical wordline and bitline capacitances are removed.

The number of transistors is minimized by applying "sign bits" to each row and column. For any given group of bits, the associated sign bit will be set if more than half of the bits in the group are ones. This ensures that in the worst case, at most half of the bits are one, thereby reducing the number of transistors in the XROM array. In this case, the XROM compiler computes eight sign bits for each wordline and one sign bit per output column. While, eight words are stored on each wordline, the sign bits are not assigned to individual words. Instead, the N columns are partitioned into 4 subarrays where the $N/4$ columns within each group are highly correlated to each other. This maximizes the effectiveness of the two sign bits which are applied to each group. The block diagram in Fig. 1 indicates the four subarrays. Two "sign bit" cells are indicated. Each contains four sign bits per row, two for each of the abutting subarrays. Optimally dividing the output columns into four groups is an instance of the graph partitioning problem which is in the class of computationally intensive NP-complete problems [1].

Manuscript received October 25, 1988. This work was supported in part by the Defense Advanced Research Projects Agency and by the Air Force Office of Scientific Research. This paper was recommended by Associate Editor M. R. Lightner.

R. W. Linderman was with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH. He is now with the Rome Air Development Center, Griffiss Air Force Base, NY 13441.

P. C. Rossbach was with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH. He is now with Motorola's Microprocessor Products Division, Austin, TX 78749.

D. M. Gallagher was with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH. He is now with the Air Force Operational Test and Evaluation Center, Kirtland Air Force Base, NM 87117.

IEEE Log Number 8930756.

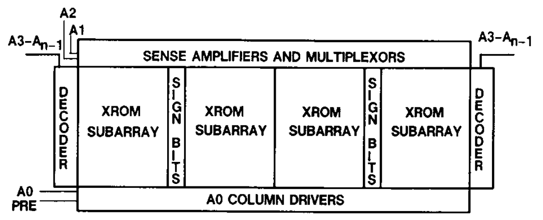


Fig. 1. XROM block diagram.

Techniques to reduce the number of transistors by scrambling addresses, adding column sign bits, and reducing column depth have been investigated in the past [3]. However, the row sign bits coupled with the optimization programs are the most effective methods to minimize transistors.

Following transistor minimization, the number of drains is minimized by exactly ordering the rows and columns within the ROM, abiding by the grouping constraints defined for transistor minimization. This is an instance of the *NP*-complete traveling salesman problem.

As the ROM is laid out, vertical metal lines are trimmed above and below the locations they are last needed. The layout program juxtapositions the XROM arrays with the wordline decoder PLA's, sense amplifiers, and other peripheral circuits. The PLA's and sense amplifiers are automatically personalized to reflect the final ordering of rows and columns dictated by the solutions of the optimization problems.

This paper discusses the basic structure of the XROM, the optimization routines, the layout program, and the user interface. Several compiler applications are discussed and testing results from fabricated XROM's are presented. The compiler can be retargeted to handle XROM cell libraries for a variety of fabrication processes. In this paper, we use examples from 3- and 1.2-μm CMOS technologies supported by the MOS implementation service MOSIS [5].

II. XROM Structure and Operation

The XROM floorplan, shown in Fig. 1, resembles many other memory macrocells. Of the N address lines, $A0$, $A1$, and $A2$, are used by the "column decode," the rest enter the "row decoder." Wordlines run horizontally across the ROM in polysilicon and also, on occasion, in second metal. The second metal shunts to the polysilicon after each of four subarrays to reduce the wordline series resistance, allowing the wordlines to be driven faster. Row decoders must be placed on each side of the XROM to match the small vertical pitch afforded by the XROM cell (typically 6 to 8 lambda).[1]

Bitlines run vertically into the 4 to 1 column multiplexors and sense amplifiers atop the ROM. These multiplexors are controlled by address lines $A1$ and $A2$. The $A0$ address line is fed in from the bottom of the array on lines interspersed with the bitlines, as shown in Fig. 2.

[1]Lambda is a scalable size parameter generally equal to half the minimum gate length. For a 3-μm CMOS process, lambda equals 1.5 μm.

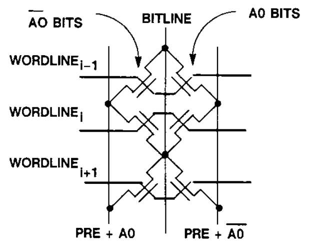


Fig. 2. XROM cell schematic.

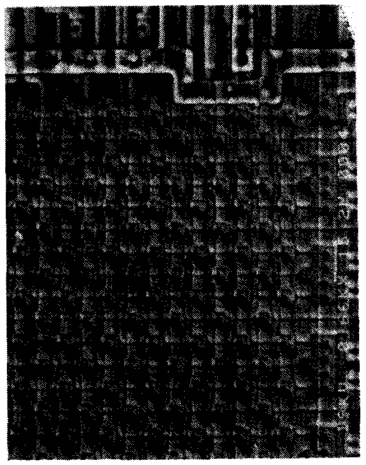
The name "XROM" comes from the "X" shape of the transistors attached to the bitline. The four-way sharing of drains results in lower parasitic capacitance and dense cell layout. The presence of a transistor denotes a "1" bit; if the transistor is missing, a "0" is stored [15].

We have fabricated XROM's both with and without constraining the designs to use only 90° angles. Fig. 3(a) shows the non-Manhattan, arbitrary angle design, where the basic storage cell requires only 12 lambda by 12 lambda. The Manhattan version, Fig. 3(b), requires 13 lambda by 16 lambda but affords the advantages of running second metal over the polysilicon wordlines to reduce resistance, and allowing Manhattan circuit extractors to model the switch level behavior of the XROM. While in theory, either cell type will function correctly, the aggressive, non-Manhattan design should be closely coordinated with a particular fabrication process. One attempt to fabricate the XROM was unsuccessful due to problems with the non-Manhattan design. We chose to continue development with the Manhattan cell set. However, given a target process, a non-Manhattan transistor extractor and a low resistance polysilicide, the deficiencies of the non-Manhattan approach are overcome, making its superior density most attractive.

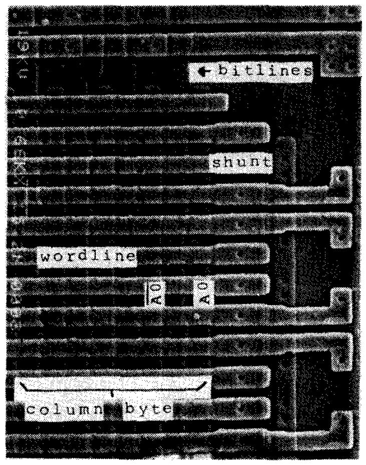
The XROM cells are personalized by placing diffusion strips to form transistors where "1's" are to be stored. This provides high density at the expense of flexibility compared to a second metal programmable ROM. The compiler discussed in this paper uses diffusion programming for three other reasons as well. First, the ROM's were fabricated through the MOSIS, negating any advantage of metal or contact level programming. Second, the ROM information was well defined by the end of the chip design cycle through simulation and placing any "volatile" information in a laser programmable ROM [14]. Finally, diffusion programming permits further optimizations to improve speed, power, and yield.

Fig. 4 shows a portion of a Manhattan XROM array. Note the trimming of the $A0$ columns and the widening of the wordlines where drains have been removed. This example, taken from a microcode XROM, demonstrates the large number of ROM features that can be removed if diffusion programming is permitted.

The XROM structure doubles the storage density through precharging. During *precharge*, the $A0$ input columns are driven to 5 V and the bitlines are pulled high through n-channel transistors. Depending on the pass



(a)



(b)

Fig. 3. 3-μm CMOS XROM photos. (a) Non-Manhattan cell set. (b) Manhattan cell set.

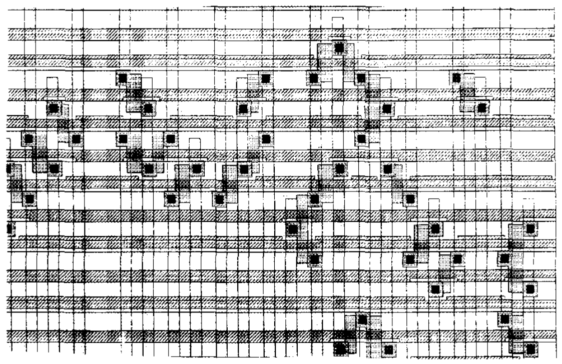


Fig. 4. XROM array section.

transistor voltage drop, the precharged voltage is somewhere between and 3 and 3.5 V for V_{dd} = 5 V.

Also, during the precharge period, the new address is decoded by the PLA row decoder to select the new word line. A NAND decoder avoids the static power dissipation problems of the NOR decoder approach and is fast enough to resolve the new wordline while the arrays are precharging. Both decoder approaches can be implemented in a PLA which allows the rows to be arranged in any desired order, thus providing a degree of optimization not readily available with a custom logic decoder.

When the *precharge* signal drops, either PRE + ***A*0** or PRE + $\overline{\boldsymbol{A0}}$ will follow, giving an opportunity to pull bitlines low. The bits in the word at addresses with ***A*0** = **1** are stored to the right of the bitline in Fig. 2. Fighting will occur if both the ***A*0** and $\overline{\boldsymbol{A0}}$ transistors are present. The resolved voltage will be less than 2.5 V which the sense amplifier will correctly interpret as a logic "0."

Since for every wordline, each bitline has both an ***A*0** bit and an $\overline{\boldsymbol{A0}}$ bit, the four bitlines input to each 4 to 1 multiplexor collectively store eight bits of information for each wordline. These bits are referred to as a "column byte." During each access, on the selected wordline, one bit of the column byte is read out, depending on the values of ***A*0**, ***A*1**, and ***A*2**. Fig. 3(b) indicates the width of a column byte. Note that second metal runs directly atop polysilicon and the two are shunted at the edge of each subarray.

The column multiplexing and sense amplifier circuitry is shown in Fig. 5. The four bitlines are multiplexed into each sense amplifier through series n-transistors controlled by the decoded ***A*1**, ***A*2** address lines. The multiplexor output is precharged to V_{dd} through a p-transistor. An additional p-transistor provides a small resistive pull-up to improve noise immunity and support single step operation.

The output of the sense amplifier can be inverted if the sign bit associated with the word read from the XROM is a "1." Also, every bit in the column can be inverted again by exchanging the WORD and $\overline{\text{WORD}}$ control lines to the transmission gates. This will be done if the column sign bit is a "1."

Precharging the bitlines to approximately 3.3 V while the sense amplifier is precharged to 5 V accelerates the sensing of a low going bitline. Shortly after the ***n*** device in the XROM array starts to pull down, the sense amp input will move from 5 to 3.3 V. The sense amp inverter is set to trigger at approximately 4 V.

The XROM compiler constructs the design up to and including the sense amplifiers but does not finalize the output circuitry since this varies greatly between applications. Typical output structures include simple buffers, tri-state bus drivers providing an additional level of multiplexing, or a pipeline register cell followed by buffers. Regardless of which output structure is selected, once the output bus is hooked up, the designer may want to constrain the optimizer on future runs to preserve the original column ordering, thereby avoiding rewiring following changes in the ROM contents. Therefore, the compiler supports a LOCK COLUMN option to support this need. The user can also lock columns on the initial compilation if a particular ordering of output columns is desired.

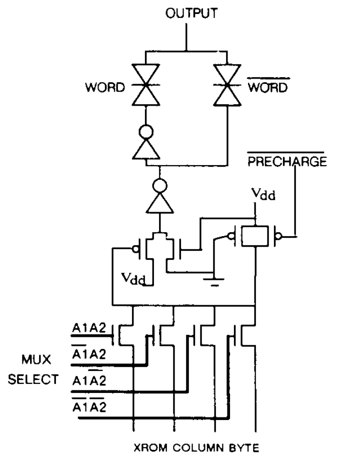


Fig. 5. Sense amplifier and multiplexor for column sign = 0.

The access time of the XROM is divided into two portions. While $\overline{\text{PRECHARGE}}$ is high, the bitlines and $\boldsymbol{A0}/\overline{\boldsymbol{A0}}$ lines are pulled up. Concurrently, the new address enters the row and column decoders and begins to propagate through to drive the new wordline. When $\overline{\text{PRECHARGE}}$ goes low the reading begins by pulling low the $\boldsymbol{A0}$ or $\overline{\boldsymbol{A0}}$ line followed by the bitline and the sense amplifier (assuming a one is read). The sign bit is read out in parallel with the data bit and is gated to control the output of the sense amplifiers.

III. Transistor Minimization

The regular structure of the XROM layout provides several unconstrained orderings which can be exploited to optimize the XROM. If the total number of devices or "1's" in the XROM can be significantly reduced, the yield, power dissipation, and speed of the XROM will be improved [12]. The yield of CMOS circuits is inversely proportional to the area of diffusions and channels on the chip [10], both of which are reduced by removing transistors. Power in CMOS circuits is proportional to the capacitance that is repetitively charged and discharged [2], thus power will also decrease as gates and drains are removed. The duration of $\overline{\text{PRECHARGE}} = 1$ can be reduced by removing capacitance from the worst-case bitline and $\boldsymbol{A0}/\overline{\boldsymbol{A0}}$ lines, and by reducing the number of transistors on the slowest wordline. Sensing during $\overline{\text{PRECHARGE}} = 0$ also benefits from reducing capacitance on the vertical lines.

If the XROM used no sign bits and did not invert the state of any row or column, the worst case would be an XROM full of "1's." On average, however, the ROM contents will be half ones. If a single sign bit is used for the entire ROM and there are originally more than $N/2$ ones, everything in the ROM is inverted and the sign bit is set. This guarantees a worst case of $N/2$ transistors, but the average case is still near to $N/2$ transistors.

Taking the sign bit technique to the extreme of providing one sign bit for each XROM bit, we can generate an XROM array with all zeroes. Unfortunately, the sign bit array duplicates the original XROM contents so nothing is gained. The use of one sign bit for each data word seems appealing since it would be applied to a small number of bits which perhaps exhibit some "correlation" to each other. This correlation could be in terms of sign extensions on two's complement numbers of small magnitude, or seldom used fields in microcode words which contain all zeros or all ones.

TABLE I
Correlation Value Metric

A0 ONES	A0bar ONES				
	0	1	2	3	4
0	-2 - 2j	-2 - j	-2	-2 + j	-2 + 2j
1	-1 - 2j	-1 - j	-1	-1 + j	-1 + 2j
2	-2j	-j	0	j	2j
3	1 - 2j	1 - j	1	1 + j	1 + 2j
4	2 - 2j	2 - j	2	2 + j	2 + 2j

Fig. 1 shows two special columns within the XROM arrays set aside for storing word sign bits. Each of the four subarrays has an $\boldsymbol{A0}$ sign bit and an $\overline{\boldsymbol{A0}}$ sign bit for each row. For example, given 12 sense amplifiers in a subarray, there will be 48 bits under control of the $\boldsymbol{A0}$ sign bit and 48 bits under the $\overline{\boldsymbol{A0}}$ sign bit on each row. Since there are four subarrays, there are a total of eight sign bits per row.

The area penalty for incorporating sign bits is less than $1/N$, where N is the number of XROM output columns and sense amplifiers. So for $N = 50$, there is less than a 2-percent area penalty in XROM size.

In addition to these word sign bits, the optimizer also uses column sign bits. If more than half of the bits in any column are ones, then the bits in the column are inverted and the polarity of the WORD SIGN control lines to the transmission gates in the sense amplifier are flipped. Therefore, column sign bits do not incur any additional area penalty; they are "hardwired" into the ROM sense amplifier cell design.

Column sign bits are particularly effective in ROM microcode stores where the default pattern for a microcode field contains ones. Default fields are used so frequently that nearly all the bits in these columns will be ones converted to zeros when the column sign bit is set.

The effectiveness of the XROM's word sign bits can be improved by rearranging the column bytes prior to application of sign bits such that strongly correlated bytes are grouped together. This maximizes the effectiveness of the sign bits and minimizes transistor count. For purposes of calculating the "similarity" between two columns, the correlation metric given in Table I has been adopted.

Each column byte consists of four $\boldsymbol{A0}$ bits and four $\overline{\boldsymbol{A0}}$ bits. The number of ones ranges from 0 to 4. We wish to group column bytes under a sign bit such that the number of ones is either very low or very high (in which case we will set the sign bit). By offsetting the number of ones in a column in a column byte by -2, the problem is transformed into maximizing the sum of the magnitudes. Each column byte is viewed as a complex value, with its real coordinate equal to the number of $\boldsymbol{A0}$ ones minus two, and its imaginary coordinate equal to the number of $\overline{A0}$ ones minus two. The correlation between two columns, x and y, is then the sum of magnitudes of the sum of each pair of complex values which can be written as

$$C_{x,y} = \sum_{i=0}^{\text{Rows}-1} \left|\text{Re}\,(x_i + y_i)\right| + \left|\text{Im}\,(x_i + y_i)\right|.$$

Once the correlation between all pairs of column has been calculated, we have a symmetric, fully connected graph with the columns as vertices and the correlations as the edge weights. The problem is to break the graph of N vertices into four graphs of $N/4$ vertices by removing edges with the least total correlation. This is the classical graph partitioning problem.

We adapted the graph partitioning heuristic of Kernighan and Lin to solve this *NP* complete problem [7]. The algorithm divides the graph in half, then it subdivides each half into quarters. A local minimum is found each time the algorithm is run. Repeated application of the algorithm from randomly selected initial guesses consistently produces an optimum or near optimum solution [12].

The effectiveness of the sign bits depends on which address bits are chosen for the roles of $\boldsymbol{A0}$, $\boldsymbol{A1}$, and $\boldsymbol{A2}$. Therefore, scrambling of input address bits is allowed in order to obtain the best possible solution, even if the LOCK COLUMN option is used. If the ROM size is a power of two, then all address inputs are candidates for these positions. If the ROM size is not a power of two, some of the most significant address bits will be ineligible. The optimizer determines the number of address bits to consider for these roles and solves the graph partitioning problem for each possible arrangement. The best solution is saved for subsequent processing. If $\boldsymbol{A0}$, $\boldsymbol{A1}$, and $\boldsymbol{A2}$ are chosen from amongst N address lines, then the optimizer examines $N(N-1)(N-2)/2$ arrangements. Fortunately, N is generally small because large ROM's are partitioned to avoid long wordline and bitline delays. As a worst case, consider a $1\text{K} \times 64$ bit partition, $N = 10$, giving 360 arrangements to consider.

In most cases compiled to date, the optimal selection of address lines has been some minor variation of what were already the least significant address bits. This is a reaffirmation of the principle of spatial locality.

A flowchart summarizing the device minimization procedure is depicted in Fig. 6. All possible selections of $\boldsymbol{A0}$, $\boldsymbol{A1}$, and $\boldsymbol{A2}$ choices are examined. Column sign bits can be set either before or after the graph partitioning. Both options are investigated.

The device minimization results are presented in Fig. 7. This shows the number of ones in the XROM arrays successively reduced through the following steps:

1) no sign bits;
2) one sign bit;
3) word and column sign bits;
4) check all $\boldsymbol{A0}$, $\boldsymbol{A1}$, $\boldsymbol{A2}$ choices;
5) solve graph partitioning.

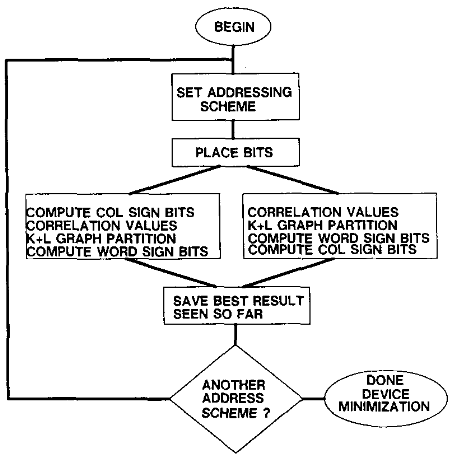


Fig. 6. Device minimization procedure.

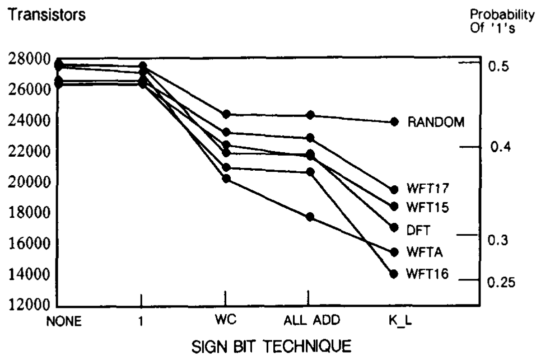


Fig. 7. Device minimization results (54K XROM's).

The data are taken from five ROM data test cases with sizes adjusted to each require 54K bits of information, and the averaged results from 50 random data test cases. The use of word sign bits and the application of the graph partitioning algorithm both netted substantial reductions in the number of transistors. As expected, the gains were not nearly as great for the random data cases, since by definition the data should be uncorrelated. However, in cases where there is correlation, the graph partitioning algorithm is highly successful.

For applications employing a further 4 to 1 multiplexor after the sense amplifiers, such as the WFT16 address table look-up, the graph partitioning routine finds the correlation between the corresponding bits in each of the four subfields and places these columns together under control of the same sign bit. In general, the optimizer finds and capitalizes upon these built-in correlations.

The sign bit approach, combined with graph partitioning, greatly reduces the worst-case number of transistors on any wordline, thereby improving speed. For the WFT16, before optimization and sign bits, the worst case was 256 transistors on a wordline for a particular selection of $\boldsymbol{A0}$, $\boldsymbol{A1}$, and $\boldsymbol{A2}$. After optimization, the worst-case loading was 134 transistors. This savings reduced the simulated risetime on the worst-case wordline from 5.4 to 3.0 ns. For the WFT17, the worst case was reduced from 241 to 169 transistors loading a wordline.

The user governs the number of times graph partitioning is solved from a random starting point for each possible selection of **A0**, **A1**, and **A2**. Experiments with between 2 and 500 random starting points indicate that the law of diminished marginal returns applies. Choosing 5 initial starting points has consistently delivered a solution within 2 percent of the solution obtained with 500 starting points. The random number generator is seeded to ensure that more starting points can only further improve upon solutions already found.

IV. Drain Removal

If programming at the diffusion mask level is permitted, further optimizations can be performed following transistor minimization to reduce the number of source/drain diffusion areas required within the ROM array.

Without performing any transistor optimization, there is a probability of $P_0^4 = 1/16$ that a drain on the input columns or on the bitlines has none of the four transistor referenced in Fig. 2. In this case, the drain could be removed entirely, reducing parasitic capacitance. Further, this allows the polysilicon wordlines to be widened, reducing series resistance. Above the last transistor on an **A0** column and below the lowest transistor on a bitline, the vertical metal lines can also be removed. Fig. 8 depicts the three varieties of **A0** column cells and bitline cells for the Manhattan cell set.

If we assume transistor optimization reduces the probability of "ones" to 0.33, then 20 percent of the drains should be able to be removed. If the probability of "ones" is 0.25, 32 percent of the drains should be unnecessary. For some microcode ROM stores, the device optimizer succeeded in reducing the probability of ones to 0.125, allowing 59 percent of the drains to be removed. Drains removed by this method could be identified simply by scanning through the data arrays after graph partitioning has been completed. However, by rearranging columns within the four subarrays and by arranging the rows in a particular order, the number of drains can be further reduced by optimization.

The problem of optimally arranging columns and rows to maximally cluster groups of four zeroes around **A0** and bitline drains is solved with a two step approach. First, columns within each subarray are rearranged to obtain the largest possible number of "zero pairs" which are adjacent zeroes on the wordlines. Then the rows are arranged to group zero pairs into clusters of four zeroes, thereby allowing drain removal. In the optimization process, it is the total number of bitline and **A0** drains which is optimized.

Within each column byte there are four bitlines, two **A0** input columns, and two $\overline{\mathbf{A0}}$ input columns (see Fig. 3(b)). There are four legal arrangements of these columns, all of which are tested by the drains optimizer to maximize

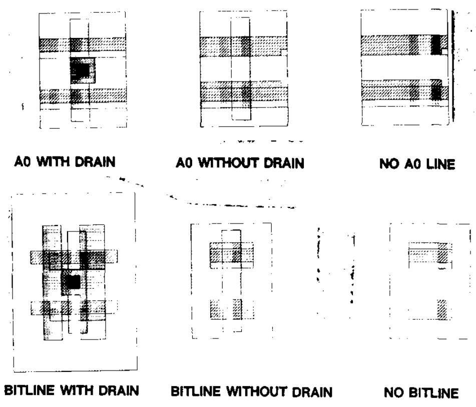


Fig. 8. Manhattan XROM cell types.

the number of zero pairs within the column. Then the $N/4$ column bytes within each subarray are optimally ordered, again to maximize zero pairs. The optimizer allows column bytes to be mirrored about the y-axis if this improves the number of zero pairs on the boundaries.

This is the traveling salesman problem for $N/4$ cities. The distance between cities is the number of nonzero pairs, and the objective is to minimize the distance the salesman must walk to visit each city exactly once. This will maximize the number of zero pairs.

After ordering the columns within each subarray, the rows are optimally ordered to maximize the number of locations with four zeros, where a drain can be removed. At this point, each even numbered row will match **A0** pairs with the row above and bitline pairs with the rows below and vice versa for the odd rows.

The traveling salesman problem is also *NP*-complete. We used the heuristic of Lin and Kernighan which resembles their solution to the graph partitioning problem [8]. A minor variation is required to handle the possibility of mirrored column bytes. The time required by their heuristic solution grows slightly faster than N^2. It employs backtracking, but does not require large amounts of memory. In practical cases, N may get as large as 256 because of the number of rows. We have found that such large problems may require one to two hours of 6 MIP CPU time. However, the investment pays off. Fig. 9 shows that solving the traveling salesman problem achieves significant additional reductions in the number of drains compared with simply removing them from a device optimized XROM (approximately P_0^4 removals.) These data were gathered from cases normalized to 54K bits, with drains for the sign bit locations not counted. Following actual layout, where the true sizes were used and the sign bit drains counted, the distribution of bitline and **A0** cell types given in Table II were obtained.

Note the extensive removal of drains from the microcode XROM's resulting from the paucity of transistors. The percentage of drains removed significantly exceeds P_0^4 in all but the last case where the probability of a one was reduced to 0.076. Bitline trimming was very suc-

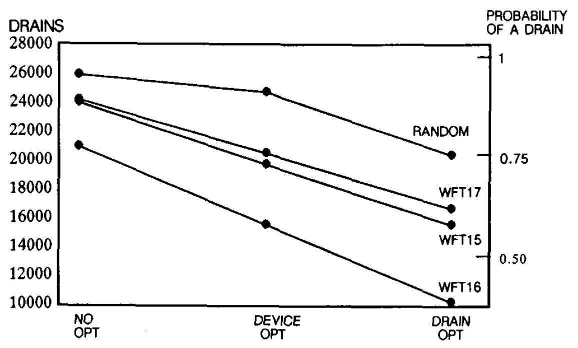


Fig. 9. Drain minimization results (54K XROM's).

TABLE II
DISTRIBUTION OF DRAIN CELL TYPES

CELL TYPE	WFT15 54K	WFT16 54K	WFT17 54.75K	μCODE1 14.25K	μCODE2 13.12K
A0 DRAIN	7636	5826	9126	954	758
A0 NO DRAIN	5753	4666	4476	847	1241
NO A0 LINE	243	3140	222	1543	1121
BITLINE DRAIN	8292	6750	9873	761	843
NO BITLINE DRAIN	5043	4097	3472	830	1073
NO BITLINE	297	2785	479	1753	1204
# OF TRANSISTORS	18357	13979	19445	1708	1006
$P_0^4 \times 100$	20	31	18	61	73
% DRAINS REMOVED	41.6	53.9	31.3	74.3	74.3

cessful in the microcode examples as expected. However, it was also very successful in the WFT16 because some unanticipated correlation was found and exploited by both optimizers.

V. AUTOMATIC LAYOUT

Arranging columns within column bytes, ordering of rows and columns, selecting sign bits, and permuting address lines must all be unscrambled by the silicon compiler to yield a correct layout. The compiler also handles the flipping, mirroring, and aligning of the XROM components. The compiler produces a set of files in the *magic* format [11], which can be translated to the *cif*, *caesar*, and *calma* representations.

The layout portion of the compiler determines where bitlines and $\boldsymbol{A0}$ columns can be trimmed. This technique, while not of great benefit in denser ROM's such as the WFT17, can lead to large reductions in metal line capacitance when ROM personalizations are sparse following optimization. This is demonstrated in Table II by the success of column trimming in the microcode stores.

In some cases, there are no transistors left under a sense amplifier following optimization. Here the output is solely determined by the word and column sign bits. The sense amplifier can be removed, reducing power dissipation and capacitive load on the precharge line.

The compiler places and personalizes the NAND PLA decoders with the selected row ordering and labels the address line inputs to indicate their scrambling. The sign bits are placed inside the XROM as shown in Fig. 1. The sense amplifiers are placed on top of the XROM with labels indicating the scrambling of columns. The $\boldsymbol{A0}$ column drivers are placed below the XROM arrays and corner cells are placed to route power, precharge, input, and output signals to the perimeter of the ROM. The user has only to route these signals to appropriate points on the rest of the chip. In most cases, an application specific output structure is placed on top of the sense amplifiers. Master-slave registers, tri-state drivers, and buffers have been created for this role.

VI. USER INTERFACE

ROM contents are passed to the compiler in a simple ASCII file of unsigned integer values. Each ROM word is broken into four integers so that ROM widths up to 128 bits can be supported. Above this width, the designer should split the information into separately compiled ROM's. This also avoids slowing the ROM access time.

The simple data input format allows the XROM optimizer to be easily interfaced to other CAD tools such as a microcode assembler tool, (e.g., GMAT [4]), and the VHSIC Hardware Description Language [6].

The user interface to the XROM optimizer is provided through the USER_PARAM.h header file. The user specifies the name of the input file containing the XROM data, the number of words in the input file, the number of bits per word, and the number of address bits. If the number of words in the input file is not divisible by 64, the data will be padded with zeros. This ensures an even number of rows in the XROM. In the header file, the user defines optimization parameters which determine the number of iterations the graph partitioning and traveling salesman optimization routines will be run from random starting points. The user may also specify that a previously determined column ordering be used by engaging the LOCKCOL option in the header file.

Once this header file has been set up, the source code of the main routine must be recompiled with the new parameters. As the compiler executes, each of the three main sections of the program, device minimization, drain minimization, and layout, create a file of statistics describing optimization results, worst-case wordline loading, distribution of cell types employed, row and column orderings, etc. This information can be used to create accurate SPICE models to simulate the actual worst-case delay path through the ROM.

The layout section of the compiler contains a reverse compiler which unscrambles the data bits just prior to layout to regenerate a copy of the input file. This provides a quick check of compiler correctness through the minimization routines.

The entire system, including layout, was validated by switch-level simulation using ESIM [13]. This tool, however, had some difficulty in simulating certain nodes where "fighting" takes place. This problem was circumvented by modifying the ESIM input file using the FIXROM tool developed at AFIT. For example, the circuit in Fig. 2 must be modified because transistors short together the $\boldsymbol{A0}$ and $\overline{\boldsymbol{A0}}$ lines through the bitline. In this case, the XROM wants to pull the bit line low when precharge drops, independent of the value of $\boldsymbol{A0}$. If this wordline is selected, "fighting" will occur on the node and the bitline will be pulled down. To simulate this circuit, the FIXROM tool searches the input file for two XROM cell transistors sharing a common drain. It then replaces them with two series transistors which allows ESIM to accurately simulate this circuit. ESIM also has trouble with an inverter which uses a static n-transistor resistive pulldown. FIXROM finds and replaces these with a standard inverters. Using these modifications, ESIM was able to fully simulate the XROM and verify the contents. A counter is usually attached to the XROM and cycled through all possible addresses to verify the XROM contents.

VII. Application Results

The diffusion programmed style of the XROM inherently provides excellent density due to the number of masks which may be involved in the personalization of the XROM. By reducing parasitics and providing a fast sense amplifier, fast access times are also available. The optimizations can further enhance speed, while decreasing power and increasing yield. The success of the transistor and drain optimization programs will be proportional to the amount of correlation in the data. The compiled ROM's have been the chosen alternative in several application areas.

A. XROM Test Chip Results

To measure the size versus speed tradeoff, the 3-μm CMOS XROM test chip shown in Fig. 10 was designed. The chip contains 12, 24, 48, and 72K XROM's storing table look-up evaluations for $1/x$, sine, cosine, and square root. Thirty-seven percent of the transistor sites were occupied, a higher percentage than the nonrandom test cases shown in Fig. 7. This indicates that less correlation was found in the table look-up data than was extracted from the microcode and Fourier transform coefficients. In this regard, the optimization results provide a measure of input data randomness.

The testing results given in Table III indicate that the minimum time required for precharging the XROM's was approximately 22 ns, independent of size. However, the minimum time required to read information from the XROM's following the fall of the PRECHARGE signal (i.e., the sense time) increased with the size of the ROM, as expected. For each XROM size, the minimum duration of PRECHARGE = 0 varied from column to column depending on the number of drains loading the respective bitlines. The access times include delay through output circuitry which includes 8 to 1 multiplexors, interconnect, and pad drivers.

In this case, the relatively high percentage of occupied transistor sites caused worst-case bitline loadings to approach $N/2$. Here, the optimized ROM's provide smaller speed improvements proportional to the worst-case bitline loading in the unoptimized ROM. Speedups are greater for the table look-up and microcode stores where the data were more highly correlated.

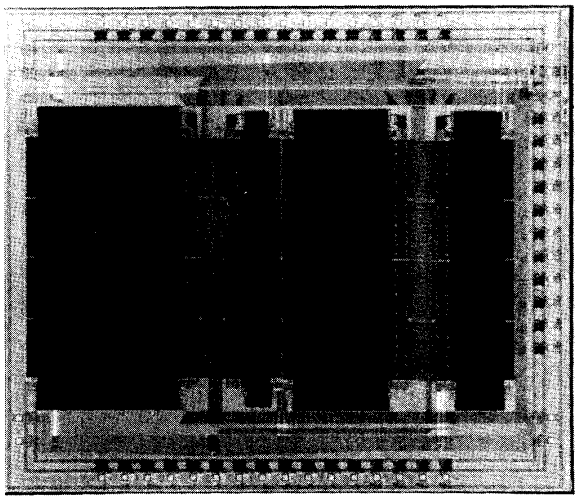


Fig. 10. XROM test chip.

TABLE III
Measured Access Time for the XROM Test Chip

XROM SIZE	PRECHARGE TIME	SENSE TIME	ACCESS CYCLE TIME (WORST CASE)
12K bit	21.4 ns	33-42 ns	64 ns
24K bit	21.7 ns	36-45 ns	67 ns
48K bit	21.7 ns	41-50 ns	72 ns
72K bit	21.3 ns	43-55 ns	77 ns

B. Table Look-Up XROM's

The WFT15, WFT16, and WFT17 54K XROM's, store addressing sequences to avoid complex computations [9]. Reducing the number of transistors on the WFT16 to 13 979 and the number of drains to 12 576 cut the power dissipation by 50 percent. Almost all of the power dissipation relates to charging and discharging capacitance on lines since only one pseudo-NMOS NAND gate in the decoder is active [12]. However, if source/drain leakage current is a significant component of power dissipation for a particular process, this too will be greatly reduced by the transistor and drains optimizations.

Optimization and layout program run times varies from less than 5 min on 12K bit ROM with two random starting points on each optimization problem, to 36 h or CPU time when running 96K XROMs with 50 random starting points. These runtime statistics were gathered on an Elxsi 6400 (6 MIP) CPU. On a 12 MIP CPU, the final WFT16 XROM was compiled from 500 device minimization starting points, and 10 starting points for drains minimization in 3 of CPU time. The optimizer requires relatively little main memory to run (typically 200–500Kbytes).

SPICE simulations for the WFT16 XROM using 1.2-μm CMOS models indicate 9 ns is required to precharge the array, decode the new address, and drive the wordline. Device optimization reduced the rise/fall time on the worst-case wordline from 5.4 to 3.0 ns. Sensing the information after precharge requires 5 ns. Allowing an additional 5 ns for propagation through the user defined output circuitry, XROM's of this size (54K bits) should have access times of 20 ns. This speed supports single cycle access up to 50 MHz.

C. XROM Microcode Stores

Following optimization, XROM's for microcode storage typically require transistors at 5–15 percent of the sites. The reduction caused by device minimization depends heavily on the definition of defaults for the fields of the microword. If all the defaults are already zeroes, the input to the compiler will already be predominately zeroes and application of sign bits will not make a substantial reduction. However, if default fields contain ones, the column and row sign bits will be quite effective since a high degree of correlation will be present. In either case, the drains minimization and column trimming techniques perform well, as the examples in Table II demonstrate, because the arrays are sparsely populated.

When a microcode field with a default value of "1" is inverted by a column sign bit, the worst-case speed of the ROM is significantly improved. Where nearly every location would have required a transistor, now only those places deviating from the default value require a transistor.

Testing results from a 3-μm CMOS Manhattan version of a 16K bit microcode store included as part of a larger microprocessor have indicated maximum access time less than 50 ns. This agrees with SPICE simulation results for that technology and ROM size, and is consistent with the results from the XROM test chip.

VIII. Conclusions

Optimization techniques can be applied to the silicon compilation of dense ROM's to net significant improvements in the speed, yield, and power dissipation. The device minimization procedure applies to a variety of ROM design methodologies. If the ROM is diffusion programmed, further optimization can be performed to significantly reduce the number of source/drain regions. Compiled XROM's have been fabricated and test results show performance consistent with simulation results. A simple interface allows the compiler to interface to a variety of higher level CAD tools. The XROM optimizing compiler can provide timely and correct designs for architectures requiring table look-up or microcode storage.

Acknowledgment

The Generic Microcode Assembler Tool (GMAT) written by Lt. R. Scott Hauser, greatly speeds the transition from microcode into an XROM design in a flexible and error free manner. His initiative in developing this collateral CAD tool is greatly appreciated.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*. Reading, MA: Addison-Wesley, 1985.

[3] K. M. Guttag, "Compressing control ROM for VLSI microprogrammed microprocessors," *IEEE Micro.*, vol. 13, pp. 115–121, Nov. 1980.

[4] R. S. Hauser, "Design and implementation of a VLSI prime factor algorithm processor," Sch. Engineering, Air Force Instit. Tech., Wright-Patterson AFB, OH, Dec. 1987.

[5] *MOSIS User Manual*, Information Sciences Institute, Univ. Southern Calif., Marina Del Rey, CA, 1988.

[6] *VHDL Language Reference Manual*, Intermetrics, Inc., AFWAL/AAD, Wright-Patterson AFB, OH, Aug. 1985.

[7] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.

[8] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Oper. Res.*, vol. 21, pp. 498–516, 1973.

[9] R. W. Linderman, C. G. Shephard, K. Taylor, P. W. Coutee, P. C. Rossbach, J. M. Collins, and R. S. Hauser, "A 70-MHz 1.2-μm CMOS 16-point DFT processor," *IEEE J. Solid-State Circuits*, vol. 23, pp. 343–350, Apr. 1988.

[10] D. G. Ong, *Modern MOS Technology: Processes, Devices, and Design*. New York: McGraw-Hill, 1984.

[11] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 87–100, Jan. 1984.

[12] P. C. Rossbach, "Control circuitry for high speed VLSI winograd Fourier transform processors," M.S. thesis, AFIT/GE/ENG/85D-35, Sch. of Eng., Air Force Instit. Tech., Wright-Patterson AFB, OH, Dec. 1985.

[13] C. Terman, "Esim: An event driven switch level simulator," in *1986 Berkeley CAD Tools User's Manual*, 1986.

[14] J. J. Tillie, "Laser programmable read-only memories," M.S. thesis, AFIT/GE/ENG/88D-4, Sch. Eng., Air Force Instit. Tech., Wright-Patterson AFB, OH, Dec. 1988.

[15] D. R. Wilson and P. R. Schroeder, "A 100ns 150mW 64K bit ROM," in *Proc. 1978 Int. Solid-State Circuits Conf.*, pp. 152–153, 273, Feb., 1978.

*

Richard W. Linderman (S'81–M'84) received the B.S.E.E., M. Eng., and Ph.D. degrees from Cornell University, Ithaca, NY, in 1980, 1981, and 1984, respectively.

He was assigned as an Assistant Professor of Electrical and Computer Engineering at the Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, where he taught graduate-level courses in computer architecture and VLSI. He is currently assigned to the Rome Air Development Center, Griffiss Air Force Base, NY, working in the area of space based real-time signal processing. He is also an adjunct associate professor with the Air Force Institute of Technology. His research interests include VLSI architecture and design, computer systems architecture, digital signal processing, and submicrometer VLSI device fabrication.

Captain Linderman is a member of Tau Beta Pi and Eta Kappa Nu.

*

Paul C. Rossbach (M'86) received the B.S. degree from the United States Military Academy, West Point, NY, in 1980, and the M.S.E.E. degree from the Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, in 1985.

He is currently with Motorola's Microprocessor Products Division in Austin, TX.

Mr. Rossbach is a member of Phi Kappa Phi, Tau Beta Pi, and Eta Kappa Nu.

*

David M. Gallagher received the B.S.E.E. degree from the U.S. Air Force Academy, Colorado Springs, CO, in 1978, and the M.S.E.E. degree from the Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, in 1987.

He is currently a Major assigned to the Air Force Operational Test and Evaluation Center, Kirtland Air Force Base, NM, where he is involved in testing electronic warfare systems.

Major Gallagher is a member of Tau Beta Pi and Eta Kappa Nu.