An Architecture for Electrically Configurable Gate Arrays

ABBAS EL GAMAL, SENIOR MEMBER, IEEE, JONATHAN GREENE, JUSTIN REYNERI, ERIC ROGOYSKI, KHALED A. EL-AYAT, AND AMR MOHSEN, SENIOR MEMBER, IEEE

Abstract —**An architecture for electrically configurable gate arrays using a two-terminal anti-fuse element is described. The architecture is extensible, and can provide a level of integration comparable to mask-programmable gate arrays. This is accomplished by using a conventional gate array organization with rows of logic modules separated by wiring channels. Each channel contains segmented wiring tracks. The overhead needed to program the anti-fuses is minimized by an addressing scheme that utilizes the wiring segments, pass transistors between adjacent segments, shared control lines, and serial addressing circuitry at the periphery of the array. This circuitry can also be used to test the device prior to programming and observe internal nodes after programming. By providing sufficient wiring tracks segmented into carefully chosen lengths and a logic module with a high degree of symmetry, fully automated placement and routing is facilitated.**

I. Introduction

Mask-programmable gate arrays offer the architectural flexibility and efficiency to integrate thousands of gates, but require long development time and high nonrecurring engineering costs. On the other hand, the convenience of field programming is available with programmable logic device (PLD) technologies, but their architectures have not allowed integration of a wide variety of applications exceeding a few hundred gates [1], [2].

We describe a novel gate array architecture [3] which combines the flexibility of mask-programmable arrays with the convenience of field programmability. Its implementation is made possible by a two-terminal electrically programmable anti-fuse offering low resistance in its conducting state and small area.

The architecture supports a design style similar to conventional gate arrays, including fully automatic placement and routing algorithms attaining 85–95-percent utilization. This required considerable emphasis on symmetry and routability, which we touch on below.

The anti-fuse is so called because it irreversibly changes from high to low resistance when "blown" by applying a programming voltage across it. The anti-fuse, or fuse for short, has an ON-state resistance of approximately 500 Ω. The layout area of the fuse cell is generally limited by the pitch of the first- and second-level metal lines that connect to it; it is about the same size as a via.

Manuscript received August 22, 1988; revised December 2, 1988.
The authors are with Actel Corporation, Sunnyvale, CA 94086.
IEEE Log Number 8826132.

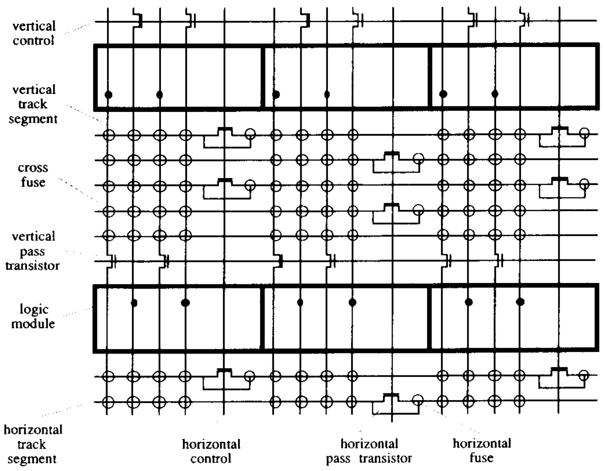


Fig. 1. Interconnect architecture.

This paper focuses on the architecture itself, which is fairly independent of the exact details of the particular CMOS technology and the anti-fuse. Other papers describe more fully the anti-fuse [4], a CMOS circuit implementing the architecture [5], and a study comparing the architecture's logic density to that of conventional gate arrays [6].

II. Programmable Interconnect Architecture

The general architecture, shown in Fig. 1, exhibits the familiar gate array organization: rows of logic cells interspersed with routing channels. There are, of course, several key differences.

The tracks in the channels are not simply empty areas in which metal lines can be arranged for a specific design. Rather, they contain predefined wiring "segments" of various lengths. Other wiring segments pass through the channels vertically. Each input and output of a logic module is connected to a dedicated vertical segment. Other vertical segments just pass through the modules, serving as feedthroughs between channels. (The number and lengths of segments in Fig. 1 are only suggestive.)

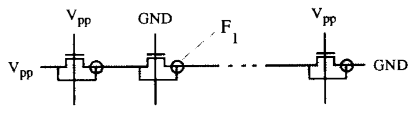


Fig. 2. Horizontal fuse programming.

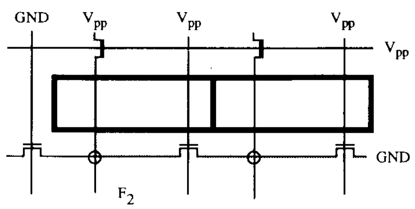


Fig. 3. Cross-fuse programming.

A fuse is located at each crossing of a horizontal and vertical segment. Programming one of these "cross fuses" provides a low-resistance bidirectional connection between the segments. Other fuses are located between adjacent horizontal segments within a track. When blown, these "horizontal fuses" connect the two segments to form a longer one. (Although not shown in the diagram, fuses may also be provided to connect adjacent vertical segments.)

In order to program a fuse, we need to apply high voltage across it. This is accomplished by an efficient addressing scheme that uses the wiring segments themselves, pass transistors connecting adjacent segments, and control logic at the periphery of the array. Fuse addresses are shifted into the chip serially.

As shown in Fig. 1, each column of "horizontal pass transistors" connecting horizontal tracks is controlled by a shared "horizontal control" line running across the array. Each row of "vertical pass transistors" is controlled by a "vertical control" line. The peripheral circuitry can drive the control lines and the segments at the end of each track.

Horizontal fuse programming is quite simple. In the example of Fig. 2, we apply programming voltage V_{PP} across the fuse F_1. All horizontal control lines except the one in the column containing F_1 are turned on by connecting them to V_{PP}, and the appropriate track segments are driven to GND and V_{PP} as shown. (Vertical fuses, if present, are programmed similarly.) Cross-fuse programming uses both horizontal and vertical control lines as shown in Fig. 3. Segments not driven to either GND or V_{PP} are left precharged to $V_{PP}/2$. Thus the voltage across fuses not being programmed is either zero or $V_{PP}/2$.

Some care is required to assure that a unique fuse is addressed. Fig. 4 shows how previously blown fuses can divert current along a "sneak path," in this case programming fuse F_5 through previously blown fuses F_3 and F_4 instead of programming F_2. Fortunately, we are not interested in blowing an arbitrary pattern of fuses (this is not a PROM!). For example, we are not concerned with programming a pattern that connects two outputs together since this does not form a useful net. If we consider only relevant patterns, it can be shown that programming the fuses in a carefully chosen order eliminates sneak paths. In general, fuses must be programmed starting from the center of the chip and moving outward, channel by channel. Determining the proper order is a bin sort operation, and can be done by software in linear time.

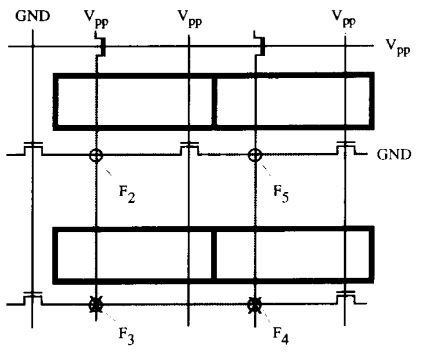


Fig. 4. A sneak path.

TABLE I

macro	4 transistor cells	modules
3 input NOR	2	1
4:1 mux, non-inverting	6	1
D latch with clear	4	1
D flip-flop with clear/set	7	2
full adder	10	2

The pass transistors and peripheral control logic are also used to test the chip; this is discussed in detail later.

III. Choice of the Logic Module

As outlined so far, the programmable interconnection architecture could be used with a variety of logic modules. Which would be best? This turned out to be a very difficult question, involving subtle trade-offs among routability, the logical capability of the module as perceived by the user, and delays due to capacitive loading in the routing segments.

The complexity of the module must be balanced with the routing overhead. Mask-programmed gate arrays provide very flexible and efficient routing. They therefore use a simple four-transistor cell. On the other hand, routing is very expensive in both area and delay with present programmable logic arrays. These generally use a module capable of implementing more complex functions [2]. The architecture outlined here has a cost of routing closer to a conventional gate array, suggesting a logic module of intermediate size. Because this is about the same complexity as conventional gate array hard macros, the designer can use a library like the familiar gate array cell libraries; there is no need to map logic into a more complex module. Table I lists several typical gate array macros and the numbers of four-transistor cells and logic modules required to implement them.

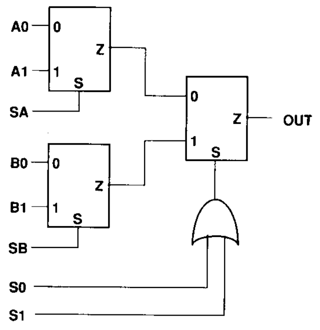


Fig. 5. Module function.

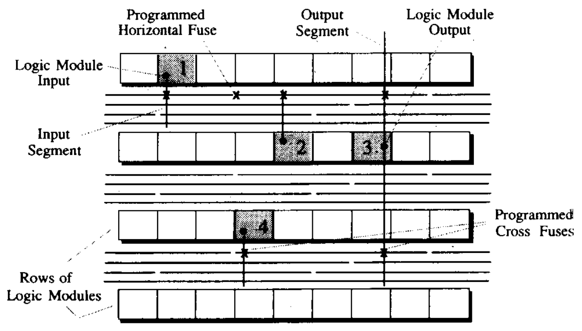


Fig. 6. A routing path.

Our chosen module, shown in Fig. 5, has eight inputs and a single output. It is composed of three two-to-one multiplexors, with an OR gate on the last stage's select input. Various macros, such as those in the table, are implemented by using an appropriate subset of the inputs and tying the remaining inputs high or low. Thus the module can implement all macros with two inputs, most with three inputs, many with four inputs, etc.

The module's output is connected to a vertical segment spanning several channels. Each input is connected to a short vertical segment spanning one channel. Four of these span the channel above the module, four the channel below. The use of short segments for the inputs reduces parasitic capacitance and hence delay.

Note that each input is accessible from either the channel above or below but not both. At first, this would appear to limit routability compared to a conventional "double-entry" gate array cell, in which signals may enter from either channel. However, there is nearly always more than one way to implement a macro. For example, there may be up to four distinct ways to implement a two-input gate: with both signals connecting to inputs in the top channel, with both signals connecting to inputs in the bottom channel, with one signal in the top and the other in the bottom channel, or vice-versa.

By letting the router choose an implementation that uses inputs accessed from convenient channels, the benefits of full double-entry symmetry are approached or sometimes attained. The degree of symmetry possible for a particular macro m implemented in a given module is reflected in the following measure S:

$$S(m) = \log_2(N(m))$$

where $N(m)$ is the number of distinct possible implementations of the macro m. Full double-entry symmetry would correspond to a value $S(m)$ equal to the fan-in of the macro. To evaluate the overall symmetry of a module, we average $S(m)$ over the macro library, weighted by relative macro usage $U(m)$ and the fan-in $F(m)$:

$$\frac{\sum_m U(m)S(m)}{\sum_m U(m)F(m)}.$$

This is the effective fraction of macro inputs in a typical design that have double-entry symmetry, and is an important criterion for choosing a module.

IV. Routing

Fig. 6 illustrates the routing of a net. The vertical segment connected to the driving module's output is connected by cross fuses to horizontal segments, which in turn connect to the segments associated with module inputs. In the top channel, a horizontal fuse is used to link two segments into a longer one.

The resistance of the blown fuses and the parasitic capacitance of the segments used form an RC tree, with the driver of the net as the root. Note that each input is driven through a maximum of three and generally two fuses to limit the delay. (If the number of series stages in the RC tree were allowed to increase further, the delay through the routing would increase rapidly.) The maximum number of fuses and the segment lengths (hence capacitances) can be altered to suit the chip dimensions and the resistance of the fuse technology.

In rare instances, it is not possible to place the macros so that all inputs on the net lie within the channels spanned by the output segment of the driving module. To handle this case, a few additional uncommitted long vertical segments are provided. The net is then routed from the output segment to a horizontal segment, then to the long vertical segment, then to another horizontal segment, and finally to the necessary input segments. To keep the number of fuses in series limited to four, no horizontal fuses are allowed in such nets. (If necessary, the architecture can be extended to provide a special fuse connecting the output directly to a long vertical segment passing over the driving module, thus eliminating the first horizontal segment and reducing the total number of fuses in series back to three.)

A means must also be provided to connect internal signals to the bonding pads of the chip. Each pad has a dedicated bidirectional buffer, which connects to the array through an associated I/O module. The I/O modules fit in the outer columns and rows of the array next to the logic modules. Each I/O module has two inputs, data and enable, and an output. The data and enable signals are sent to the output buffer of the associated bonding pad, and the module's output comes from the input buffer of the pad. Thus the I/O module can be configured to provide input, output, tristate, or bidirectional capability.

To minimize clock skew due to differential routing delay, one entire track (or more if needed) in each channel is set aside for clock distribution. These tracks are connected directly to buffers, so that each input presents a similar load driven through exactly one fuse.

An interesting theoretical question is whether more horizontal tracks are needed in each channel here (where the lengths of the wiring segments must be predetermined) than in mask-programmed routing (where the wiring is customized for the design). Surprisingly, a high probability of routability is obtained with only a few tracks above channel density.

This requires a careful choice of the lengths of the segments, based on statistics from an extensive suite of design examples. This was done by first determining the distribution of net lengths, i.e., the length each net would run along each channel if the constraint of fixed segmentation were absent (as in a conventional gate array). The distribution of physical segment lengths provided on the chip was chosen to obey similar statistics. Then the segmentation was "tuned" manually based on actual routings which obeyed the constraints it imposed.

To obtain good routing performance it is also necessary to take advantage of the symmetry of the macros where possible. For example, observe that if macro 4 in Fig. 6 permits its input to be routed from either the upper or lower channel, there is a better chance of finding a free horizontal segment to connect it.

V. Testing

To assure high programming yield, it is necessary to thoroughly test the chip for defects in the modules and fuses prior to programming. With a simple addition, the addressing circuitry used for programming suffices for this purpose as well.

Continuity of the tracks is easily verified by turning on all vertical and horizontal pass transistors, and using the peripheral circuits to drive the tracks from one end and read them back from the other. Testing for the absence of shorts between adjacent tracks is done in a similar way by applying a pattern of alternating ZERO's and ONE's.

Shorted or weak cross fuses are detected by turning on all horizontal and vertical pass-transistor lines, grounding all horizontal segments, and driving all vertical segments to some stress voltage. Horizontal fuses are tested column by column, with the same addressing method that is used to program them.

To verify the functional operation of the modules, we need to apply test vectors to their inputs and read their outputs. A vector is applied simultaneously to an entire row of modules by turning on all vertical pass transistors except those in the row being tested. Data are applied to the inputs in the channel above the row from the periphery at the top of the array, and to the inputs in the channel below the row from the bottom of the array.

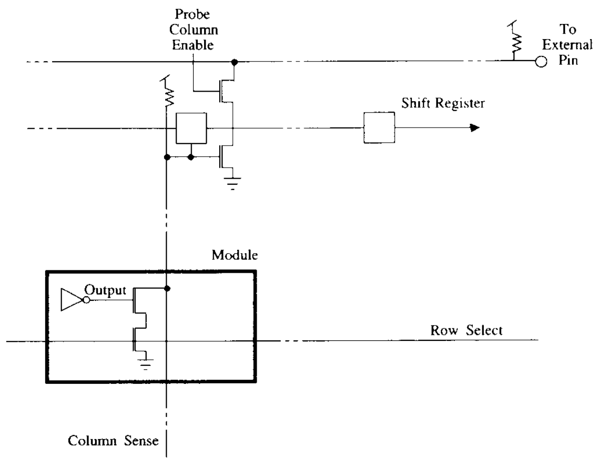


Fig. 7. Probe circuit.

Since the outputs of the modules share a vertical track with outputs of other modules above and below them, some other means is required to read the module outputs at the array periphery. As shown in Fig. 7, a select line is provided along each row of modules, and a sense line along each column. Activating the select line for the row of modules under test gates their output values onto the sense lines. The sense lines are loaded into a shift register at the top of the array.

This ability to read the output of any module at the array periphery is highly useful *after* programming as well. Only a small amount of extra circuitry is required to select one of the sense lines and make its value available at an external pin of the chip. Thus by shifting in the appropriate address, the user can observe any internal node of his design externally in real time. This virtual probe can be used and its address changed even as the programmed chip is operating in the user's system.

VI. Implementation: Silicon and Software

The architecture has been implemented in a CMOS device. For details, including the speed of the module in isolation and in an application, see [5].

Computer-aided design tools have been developed to support the architecture. Designs are entered as schematics or net lists using a cell library.

The placement and routing algorithms are specific to the architecture. As usual these are time consuming, taking up to a few hours on a low-cost workstation. They achieve 100-percent routing completion. (Even expert users have never been able to improve manually on the automatic router.) The probability of successful routing can be predicted by analyzing some statistics of the design.

Because the nets are *RC* trees, delays are not a simple function of capacitive load as with mask-programmed gate arrays. Nevertheless, we are able to quickly calculate precise delays at each input for post-layout simulation and timing verification.

Acknowledgment

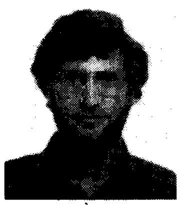
The authors gratefully acknowledge the technical contributions of J. Chang, D. Gluss, R. Guo, D. How, and F. Sohail.

References

[1] S. Wong, H. So, C. Hung, and J. Ou, "CMOS erasable programmable logic with zero standby power," in *ISSCC Dig. Tech. Papers*, Feb. 1986, pp. 242–243.
[2] H. Hsieh *et al.*, "A second generation user programmable gate array," in *Proc. Custom Integrated Circuits Conf.*, May 1987, pp. 515–521.
[3] A. El Gamal, K. El-Ayat, and A. Mohsen. "Programmable interconnect architecture," pending U.S. patent.
[4] E. Hamdy *et al.*, "Dielectric based antifuse for logic and memory ICs," in *IEDM Tech. Dig.* (San Francisco, CA), 1988, pp. 786–789.
[5] K. El-Ayat *et al.*, "A CMOS electrically configurable gate array," in *ISSCC Dig. Tech. Papers*, Feb. 1988, pp. 76–77.
[6] B. Osann and A. El Gamal, "Compare ASIC capacities with gate array benchmarks," *Electron. Des.*, vol. 36, no. 23, pp. 93–98, Oct. 13, 1988.

Abbas El Gamal (S'71–M'73–SM'83) received the B.Sc. degree in electrical engineering from Cairo University, Egypt, in 1972, and the M.Sc. degree in statistics and the Ph.D. degree in electrical engineering both from Stanford University, Stanford, CA, in 1977 and 1978, respectively.

From 1978 to 1980 he was an Assistant Professor of electrical engineering at the University of Southern California, Los Angeles. Since 1980 he has been with the Electrical Engineering Department of Stanford University where he is currently an Associate Professor. From 1984 to 1986 he was Director of the Systems Research Laboratory, LSI Logic Corporation, Milpitas, CA. He is a co-founder and Chief Scientist of Actel Corporation, Sunnyvale, CA.



Jonathan Greene received the Sc.B. degree in biology from Brown University, Providence, RI, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1983, where he performed research on configurable VLSI arrays, VLSI complexity, and information theory.

During 1984 he was with Hewlett-Packard Laboratories. From 1984 to 1986 he worked on cell design automation and module compilation at the LSI Logic Systems Research Laboratory in Palo Alto, CA. He is currently Manager of System Architecture at Actel Corporation, Sunnyvale, CA.

Justin Reyneri received the B.S. and M.S.E.E. mathematics degrees from Stanford University, Stanford, CA, in 1978, and the Ph.D. degree in electrical engineering, also from Stanford, in 1985, having done research in cryptology and information theory.

He is now Manager of System Development at Actel Corporation, Sunnyvale, CA, where he has worked since 1986. Prior to that he worked at LSI Logic's System Research Laboratory on automated data-path layout, and at Hellman Associates on communications security systems.

Eric Rogoyski received the B.S. degree in mathematics from the State University of New York at Stony Brook in 1972.

He was at IBM Corporation from 1974 to 1982 with his last position in EDS at East Fishkill, NY. He held various positions in physical design from 1982 to 1984 at the company he co-founded, California Automated Design Inc., and from 1984 to 1986 at Mentor Graphics Corporation. He is currently a software consultant at Actel Corporation, Sunnyvale, CA, supporting the architectural development and physical design of Actel's configurable technology.

Khaled A. El-Ayat received the B.Sc. degree in electrical engineering from the University of Cairo, Egypt, in 1968, the M.Sc. degree in electrical engineering and computer science from the University of Toronto, Canada, in 1971, and the Ph.D. degree in electrical engineering and computer science from the University of California, Santa Barbara, in 1977.

In May 1977 he joined Intel Corporation, Santa Clara, CA, to work in the Microprocessor Design Group, where he worked on the definition and development of industry standard microprocessor families such as 8086, 80186, and 80386. As a Project Manager at Intel, he was responsible for the design of the control structures of the 80386 Microprocessor. After leaving Intel, he cofounded Actel Corporation in Sunnyvale, CA, and was the Program Manager responsible for development of electrically configurable gate arrays. His present research interests include configurable logic and application-specific architectures, VLSI design, and microprocessor architectures. He has authored many articles and holds patents covering Actel's architecture and testability techniques. Presently he is a Chief Engineer working on the definition of the next generation of products.

Amr Mohsen (S'72–M'74–SM'84) received the Ph.D. degree in electrical engineering and applied physics from the California Institute of Technology, Pasadena.

He is the founder, President, and Chief Executive Officer of Actel Corporation, Sunnyvale, CA, and has more than 20 years of experience in the semiconductor industry. Before founding Actel, he was a Senior Engineering Manager in the technology division at Intel Corporation. He also worked on charge-coupled device development at Bell Laboratories and served as a consultant. He has authored more than 45 articles relating to semiconductors and is responsible for inventions covered by 20 patents.