intel

ARTICLE REPRINT

AR-265

March 1983

Versatile Algorithm, Equipment Cut EPROM Programming Time

Don Knowlton
EPROM Product Line Manager
Intel Corp.

ORDER NUMBER: 230614-001

Versatile algorithm, equipment cut EPROM programming time

Programming high-density EPROMs for large-volume applications can entail significant time and expense. To minimize both, use programming software and hardware that recognize the different characteristics of individual EPROM cells.

Don Knowlton, Intel Corp

Using the capabilities of μP-based EPROM programming equipment and employing a new algorithm that recognizes differences among EPROM cells can dramatically reduce programming time for the newest high-density devices. The typical factor-of-six reduction this system can achieve results in considerable cost savings for large-volume applications. The time required to program an 8k-byte 2764, for example, drops from 7 min to an average of 1.25 min with the procedure. As a bonus, the technique helps ensure that the device receives adequate programming—in terms of memory-cell charge—to maintain long-term reliability.

Reducing programming time and costs for EPROMs has become increasingly important because the chips have become a cost-effective, easy-to-use alternative to masked ROM in high-volume applications requiring code flexibility or simplified inventory—a major switch from EPROMs' original small-volume prototyping applications. And volume usage makes EPROM programming a significant manufacturing consideration, subject to minimization efforts.

Higher densities increase programming time

Despite dramatic (eightfold) increases in EPROM storage capacity, early technology improvements (from p-channel MOS through 3-power-supply NMOS to single-power-supply NMOS) held EPROM programming times to less than 2 min (**Table 1**). Since the appearance of the 16k-bit EPROM, however, the minimum time for reliably programming each EPROM cell has remained constant at 45 msec, doubling the total device programming time with each doubling of density. The result is an increase in the per-unit cost of programming—involving either additional time (labor) or extra equipment.

The programming procedure currently in use for most EPROMs isn't yet a major burden on most users, though; it uses a nominal 50-msec pulse per EPROM byte, resulting in a total programming time of approximately 1.5 min for a 16k-bit chip. With the introduction of the 2764 (64k bits) and devices with even higher density, however, programming times have increased. A 256k-bit EPROM, for example, would require 24 min for programming by this conventional method.

TABLE 1— EPROM PROGRAMMING-TIME EVOLUTION

DEVICE	BYTES	PROGRAMMING TIME (MIN)
1702A	256	2.0
2708	1024	1.5
2716	2048	1.75
2732	4096	3.5
2764	8192	7.0
2764*	8192	1.25
27128*	16384	2.5

*USING ENHANCED PROGRAMMING ALGORITHM DESCRIBED IN THE TEXT.

Most EPROM cells program in much less than 45 msec, however. In fact, empirical data (**Fig 1**) shows that very few cells require longer than 8 msec for programming. Therefore, a procedure that takes into account the characteristics of individual EPROM cells can significantly reduce a device's programming time.

Arbitrarily reducing programming time is risky, though, because a cell's ability to achieve and maintain its programmed state is a function of this time (see **box,** "EPROM programming"). What's needed, therefore, is a way to verify the level to which individual cells have been programmed.

Fortunately, such techniques exist. By determining the charge stored in a cell relative to the minimum charge needed to program the cell to a detectable level, you can check for a program margin that assures reliable EPROM operation.

One way to check program margin involves varying the select-gate voltage, V_{CC}. And although you wouldn't

Reduce EPROM programming time with an intelligent algorithm

necessarily use this procedure in the actual programming of an EPROM, it serves to illustrate some EPROM characteristics and thus merits discussion.

An erased cell (having a ONE output) with no charge on its floating gate has a characteristic similar to an ordinary NMOS I-V curve (**Fig 2**). As you begin programming the cell (to a ZERO), the cell threshold as a function of select-gate voltage begins to increase. Thus, if you externally increase V_{CC}, connected to the select gate, you can determine cell threshold by observing the V_{CC} value at which a ZERO-to-ONE transition occurs on the output. If a transition occurs within V_{CC}'s normal operating range of 4.75 to 5.25V, the EPROM cell is inadequately programmed.

Some examples illustrate the process. First, consider a cell programmed for time t_1-t_0, storing a charge Q_1 on the floating gate:

$$Q_1 = \int_{t_0}^{t_1} i(t)\, dt.$$

If the charge isn't adequate to fully program the cell, the EPROM output exhibits a ZERO-to-ONE transition as V_{CC} increases through a level less than the 5.25V max operating voltage.

Now consider a second cell programmed for time t_2-t_0, storing a charge Q_2 on the floating gate. If increasing V_{CC} causes a ZERO-to-ONE transition at exactly 5.25V, the cell has exactly the amount of charge required for programming. However, the cell has no margin for tolerating charge loss or small variations in V_{CC} beyond the specification maximum, and such a device might not be reliable in long-term operation.

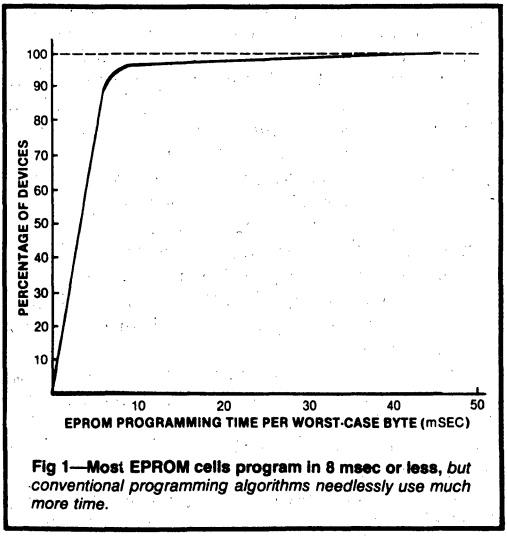


Fig 1—Most EPROM cells program in 8 msec or less, *but conventional programming algorithms needlessly use much more time.*

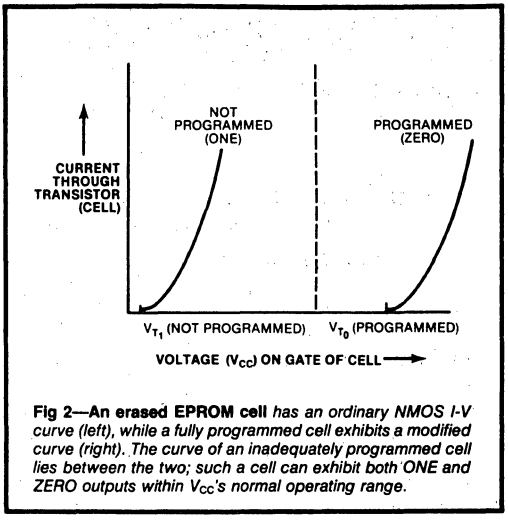


Fig 2—An erased EPROM cell *has an ordinary NMOS I-V curve (left), while a fully programmed cell exhibits a modified curve (right). The curve of an inadequately programmed cell lies between the two; such a cell can exhibit both ONE and ZERO outputs within V_{CC}'s normal operating range.*

Finally, consider a device programmed for time t_2+T-t_0. In this case, charge added by the additional programming time T increases programming margin. The ZERO-to-ONE transition detected while increasing V_{CC} thus occurs when V_{CC} is greater than 5.25V, providing assurance of reliability.

Margin checking doesn't occur in conventional EPROM programming, however. Instead, each EPROM cell receives a 45- to 55-msec write pulse, and manufacturers ensure program margin by screening out devices having bytes that don't program within 45 msec. This programming procedure is thus an open loop—no actual verification of margin occurs.

Improved algorithm uses closed-loop technique

In contrast, the trademarked inteligent Programming algorithm, a procedure devised by Intel for programming high-density EPROMs, guarantees reliability through the closed-loop technique of margin checking. It ensures that an EPROM cell's intended ZERO output won't become a ONE within V_{CC}'s normal operating range.

The algorithm begins by setting V_{CC} to 6V (higher than required for normal operation but necessary to provide programming margin), setting the programming voltage V_{PP} to 21V and then iteratively supplying 1-msec LOW-going programming pulses to the EPROM's program-disable pin $\overline{PGM}$. After each pulse, the algorithm checks the EPROM's output for the desired programmed value. If the output is incorrect, the algorithm repeats the pulse-and-check operation; incorrect output after 15 pulses causes rejection of the EPROM device.

If the EPROM is fully functional, however, one of the pulses results in proper EPROM output. At that point, the algorithm supplies still another programming pulse—this one four times longer than the combined length of the previously applied 1-msec pulses. This longer pulse helps ensure that the EPROM cell has adequate programming margin for reliable operation. Although the pulse can be as long as 60 msec, very few EPROM bytes require this much programming time. In fact, most EPROM bytes program with only one or two 1-msec pulses, so a typical total programming time per byte is 5 to 10 msec.

TABLE 2—
EPROM PROGRAMMER DESIGN RANGES

PARAMETER	MINIMUM	MAXIMUM
V_{CC}	4.5V	7.0V
V_{PP}	10.0V	25.0V
t_{PW}*	0.1 mSEC	60 mSEC

*PROGRAMMING-PULSE WIDTH

Use commercial programmers or design your own

Most commercial EPROM programming equipment can accommodate this programming algorithm with minor changes to hardware and software. For example, Data I/O Models 120A and 121A programmers require that firm's Revision D software; the company's Unipak, Unipak II and Mospak programmers require Revisions 004, 001 and 003, respectively. You need Revision B software from Pro-Log to use the algorithm on that firm's M980 control unit; you also need the PM9080 module and PA28-80 socket adapter.

If you design your own equipment to implement the algorithm, you must consider several application factors in weighing performance and flexibility against cost. For instance, if your application is dedicated to programming only the 2764, you can construct a system totally in hardware to minimize costs. You need only three fixed-voltage power supplies (5, 6 and 21V), and you can generate pulses with simple devices such as one-shots. Logic arrays or a μP can perform the counting and algorithm sequencing.

Systems that program several types of EPROMs require a more general approach; they're best implemented with a μP and programmable power supplies. Look-up tables in the program-store memory can access algorithms for various EPROMs in this type of system, and tables can contain pulse widths and voltage parameters. Most parameters are common to many EPROMs, so programming the various types is easy.

Moreover, you can employ software in such systems to count the 1-msec pulses and construct various other pulse widths to satisfy the programming algorithm's timing requirements.

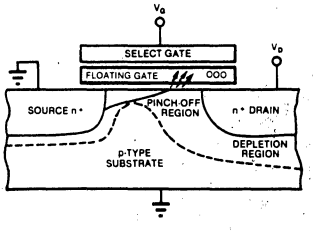
Supply-voltage generation, however, proves more difficult. An elegant and flexible solution involves using software-controlled programmable-output power supplies to deliver V_{PP} and V_{CC}, thus providing compatibility with future designs.

As an alternative, you could employ selectable supplies. Unfortunately, this less costly approach might

EPROM programming

Information storage in an EPROM results from electrically charging a floating gate in a stacked-gate MOS transistor. To charge the gate, you raise both the select-gate voltage V_G and the drain voltage V_D to a high positive level while holding source and substrate at ground potential **(figure);** suitable drain and gate voltages cause the transistor to operate in saturation.

The acceleration of electrons in the transistor's pinch-off region results from the high electric field. Some of these electrons acquire enough energy to enter the conduction band of the silicon dioxide (which electrically isolates the floating gate under normal operating voltages) and get attracted to the positive potential of the floating gate.



Information storage in an EPROM cell *results from the transfer of charge to an MOS transistor's floating gate during the programming operation.*

The basic equation describing the behavior of cell programming is:

$$i = C_{FG}\frac{dV}{dt},$$

where C_{FG} is the floating-gate capacitance. The charge accumulated on the floating gate is therefore:

$$Q = \int_{t_0}^{t_1} i(t)\,dt = C_{FG}V.$$

Thus, the amount of charge stored in the cell and the resultant level of programming are functions of the cell programming time $t_1 - t_0$.

Get a factor-of-six speedup with individual-cell programming

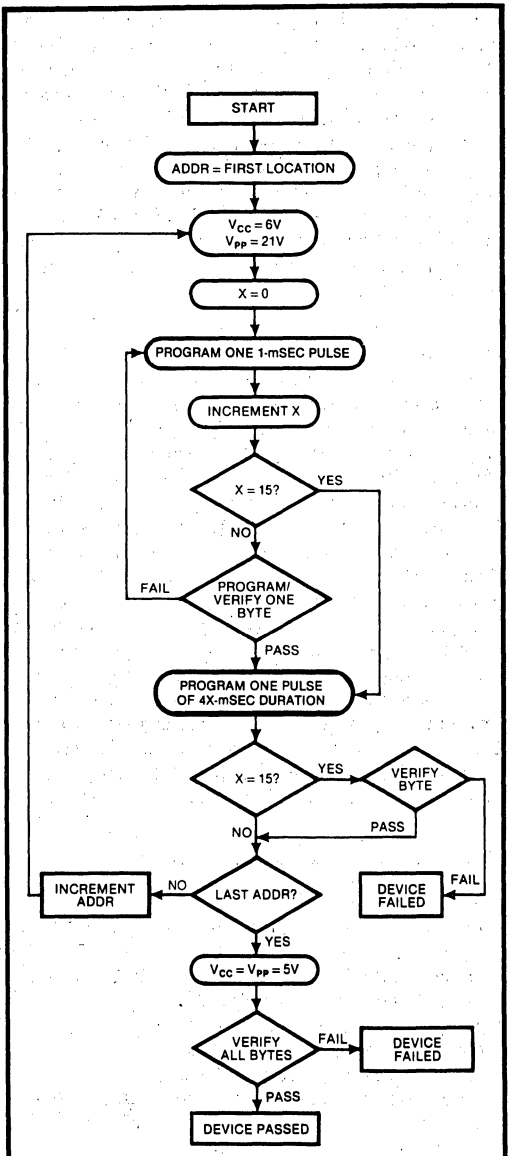
leave the equipment lacking the voltage required for programming future devices. **Table 2,** however, shows suggested voltage and timing ranges for equipment design that should meet anticipated requirements for Intel MOS EPROMs.

Equipment designed to program several EPROMs in parallel requires special design considerations. Devices that fail to program slow down the entire programming operation if each address is allowed to program for the maximum time (pulse counter=15). A ganged unit, however, should detect programming failures immediately and disable faulty devices to minimize programming time for the other EPROMs. This step is particularly critical in high-volume programming.

Good design practices help

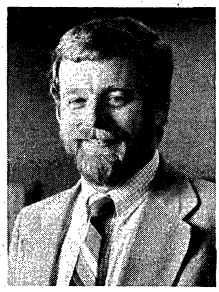
Furthermore, following many of the design rules used in dynamic-RAM applications can help you improve the reliability of high-density MOS-EPROM programmers. For example,

- Provide well-regulated power supplies at the EPROM. Short, low-inductance traces, high-frequency capacitive decoupling and ground-plane routing all reduce the effects of current surges and are necessary for high speed.
- Make sure that all signals are clean. Minimize undershoot, overshoot and glitches. Use an oscilloscope with at least a 150-MHz bandwidth to find potential problems.
- Observe absolute maximum specs on V_{PP}. Because V_{PP} is the highest voltage applied to the chip, it's usually the closest voltage to an EPROM's physical limits. Even small glitches exceeding absolute maximum ratings can result in chip destruction, so use a high-bandwidth oscilloscope to look for overshoot when setting V_{PP}. EDN



Intelligent programming algorithms, *like the one for Intel's 27128 described by this flowchart, cut programming time by more than 80%.*

Author's biography



Don Knowlton is EPROM product-line manager at Intel's Nonvolatile Memory Div (Santa Clara, CA), where he's responsible for EPROM strategic marketing. Before joining Intel, he worked at Advanced Micro Devices. Don holds BSEE and MSEE degrees, both from Purdue University. In his spare time, he enjoys playing 5-string banjo.