How EPROMS Work

From Transwiki

Contents

How EPROMs Work or:

Some Things You Wanted To Know About EPROMs, But Didn't Know Whom Or What To Ask

by Dr Ah Clem Memory

I have been asked many times by users of the pocket programmer (http://secure.transtronics.com/EPROM_Programmer.html) , if there is a good book that explains all the dirty little secrets about how EPROMs work. Well, it looks as if there isn't one. So, here is my humble attempt to provide a dose of information about them.

If you think this information contains any errors, I would be glad to hear about it; (email inform@xtronics.com (mailto:inform@xtronics.com?subject=Dr.-Memory)). This article is written with the philosophy that there is no such thing as a stupid question; only stupid mistakes from not asking stupid questions. Things only seem complicated until you figure out how they work, so lets dig in. (If you are familiar with binary, bits, bytes, hexadecimal, and ASCII you might want to skip the first section.)

What Is All This bits, bytes, Hexadecimal, Word, And ASCII Stuff About?

Binary refers to *base two* or a *two-state digit* called a *bit.* A *bit* is either on or off - represented as a '1' for *on* (the *set* state) and '0' for *off* (the *cleared* state). Eight *bits* together form a *byte* and are written as 00110101b (or sometimes %00110101). The 'b' stands for *binary,* and lets you know that we aren't talking about 110,101, the decimal number. A byte or multiples of bytes set the register size for microcomputers. *Hexadecimal* (*hex)* is a *base 16* way of representing one *byte*. Hexadecimal uses the digits 0,1,2,3,4,5,6,7,8,9,A, B, C, D, E, F. A byte requires just 2 *hex digits*. Thus, FFh is a two hex digit representation of a byte. FFh is the same as 11111111b (the little 'h' designates a hexadecimal value sometimes a preceding '$' is used as in $FF)

A *word* usually refers to two bytes or 16 bits and can also refer to a 32 bit or *wide word*. We are fast approaching the time when *64 bit words* or *doublewide words* will be common.

ASCII, is a code that represents the letters and numbers you can type on a computer keyboard. Each letter you press is represented by a unique set of 8 bits or one byte.

EPROM Size Rating

EPROMs are rated in *k-bits* where *k* is equal to 1,024 and the EPROM number generally (but not always) reflects the size. But when we talk about memory size, we speak in terms of *k-bytes*. To change bits to bytes, simply divide the number of bits by 8 to get bytes.

A couple of examples will clear things up:

The 2716 EPROM number ends in 16 and thus is 16 k-bits in size or 16 * 1,024 or 16,384 bits. Now, 16,384 bits divided by 8 gives us 2,048 bytes or 2 k-bytes. Thus a 2716 is a 16 k-bit EPROM, but is most often expressed as being 2 k-bytes in size.

Some EPROMs are word wide or 16 bits (2 bytes) wide. These EPROMs are also rated in bits, such as 27C1024 ; a 1 M-bit (Mega-bit), 1,024 k-bits, 128 k-bytes, or 64 k-word EPROM. Such EPROMs come in 40 pin packages to allow for the extra pins needed.

What is a Rom?

ROM stands for ***R***ead **O**nly **M**emory. They are programmed at the factory at the time of manufacture with a special mask, thus called a *masked ROM*. This is the cheapest way to manufacture ROMs once you need more than 10,000 at a time. The drawback is, if there is even one little bug in the software, that pile of 10,000 ROMs becomes worthless. Be aware that some masked ROMs are unreadable by EPROM programmers

What is a PROM?

PROMs (***P***rogrammable **R**ead **O**nly **M**emory) consist of an array of fuses and thus can only be programmed one-time. Programming is accomplished with a current (instead of a voltage as are EPROMs) and requires a different type of programmer.

What is an EPROM?

EPROM(**E**rasable **P**rogrammable **R**ead **O**nly **M**emory) can be programmed and erased enabling them to be re-used. Erasure is accomplished using an UV (Ultra Violet) light source that shines through a quartz *erasing window* in the EPROM package.

There also are *OTP* (***O***ne **T**ime **P**rogrammable) EPROMs, sometimes called OTPROMs (**O**ne **T**ime **P**rogrammable **R**ead **O**nly **M**emory), that are identical to an erasable EPROM, but lack an erasing window to reduce costs. To reduce the cost, these EPROMs come in a *windowless plastic carrier*, which is cheaper than the costly *ceramic package* required for the erasing window. They can be programmed one time only, so these are used after the code is bug free.

What is an EEPROM?

An *EEPROM* (***E***lectrically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory) is similar to an EPROM but the erasure is accomplished using an *electric field* instead of an UV light source. This eliminates the need of a window. Usually, EEPROM refers to a device that requires a programmer or special voltage to program it.

What is a Flash EPROM?

A *flash* EPROM is similar to an EEPROM except that flash EPROMs are erased all at once while a regular EEPROMs can erase one byte at a time. *In- circuit* writing and erasing is possible because no special voltages are required. To accomplish in-circuit operation, you have to write special application software routines. Flash EPROMs are also called *nonvolatile memory*..

OK, So tell me, just how does an EPROM work?

EPROM *memory cells* use *floating gate* technology. A *floating gate* is a gate with a special capacitor for its only electrical connection. This special capacitor takes on an electrical charge in a quantum physics effect called *tunneling*. The presence of a charge determines the value (1 or 0) of the *memory cell.* In our example below, a room with a very narrow door represents the *memory cell.* People in the room represent electrons with their associated charge. These people can only enter or exit through a much too narrow door with much pushing or shoving to represent the *tunneling* effect.

Think of a room with about 30 people acting as the electron charge. A full room of people represents a '1'; when empty a zero. When an EPROM is erased, all 30 people are pushed into the room and provide the charge that we call '1'. When we program an EPROM bit, we shove these people until they pop back out by applying a pulse of high voltage to the *memory cell.* This pulse drives the people out of the room changing the bit from a '1' to a '0'.

When programming a bit we can only change a 1 to a 0 because changing a 0 to a 1 requires erasing. To erase an EPROM, we apply an UV (Ultra-Violet) light (that shines directly on the chip) to drive our imaginary people back into the room. Erasure works on the whole EPROM not individual bits.

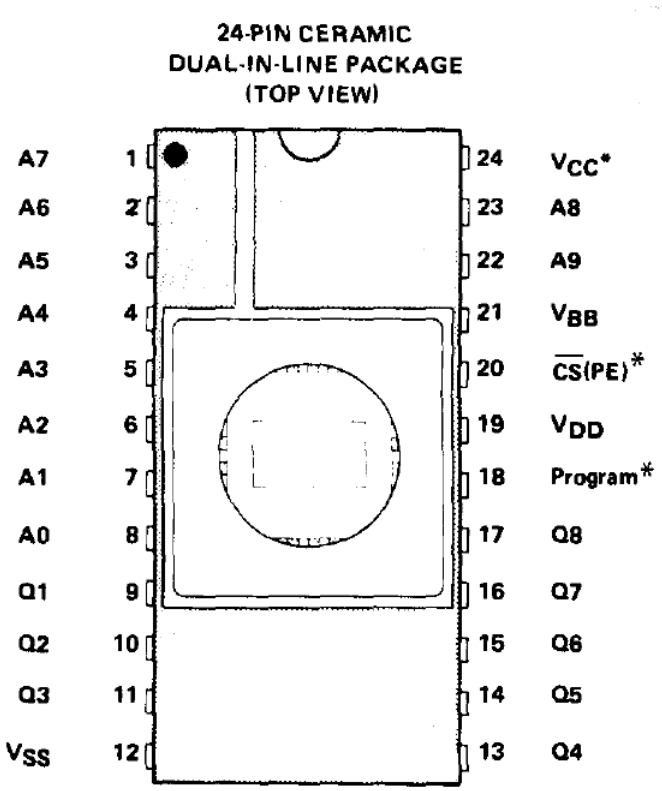
An *EPROM cell* is really an analog device. If it were digital, we would only have an empty room or a room with one person in it. Instead, a *cell sensor* circuitry compares how many people are in the room to a reference to determine if the cell represents a 1 or 0. With more than 15 people in the room, the cell is considered a 1.

When programming an EPROM you have to apply the specified programming voltage for the specified time. Too short a time or too low a voltage and not all the people get shoved out of the room. When you don't program an EPROM properly and you still have 5 people in the room, it will slow down the *cell sensor* circuitry, which slows down the *read access time* of the EPROM or might even corrupt the data. A similar thing can happen if you don't erase the EPROM long enough.

On the other hand, if you apply too high of a program voltage or over erase the EPROM, it is equivalent to blowing the doorframe right out of the wall! Our imaginary people now enter and exit the room, milling around on their own whims and we no longer have information storage! The long and short of it is you need to double check Vpp programming voltages and follow the recommended time for erasing (don't store unused chips in the eraser!). More about this later under the heading: *erasing an EPROM*.

(A little off the topic here, you may have seen little solid state recording devices, sometimes used in answering machines, that record voice grade audio. The trick to getting so much audio into storage is by storing analog values into EEPROM cells! The recorded cells have intermediate values that are not compared when read but out put as a voltage. When the memory cells are played back in consecutive order, the continuously changing values form an audio signal!)

A typical EPROM Pin Out



*For 2716 JL Only:

Pin:	
18	$\overline{CS}$ (Program)
20	A10
24	V_{CC} (PE)

V_{BB}	−5V power supply
V_{CC}	+5V power supply
V_{DD}	+12V power supply
V_{SS}	OV ground

In single voltage EPROMs two of the pins are power pins; one is +5V and the other ground. Single voltage EPROMs only require +5 to read them, but when programming they require a second voltage (Vpp). The old three-supply EPROMs require +5, +12 & -5Vdc just to read them. Yet a fourth voltage is required to program them! You no longer see three supply EPROMs in new products, but they are still out there in older equipment and are somewhat available for replacements. A good example is the three supply 2716 made by Texas Instruments, TI never made a 2716 single supply. There are also EPROMs that are pin compatible but do not have the same number such as the 2716 and the 2516 (T.I.'s version of a single supply 2716). But for those of you who program a lot of 2716's remember that T.I.'s 2716 are not the same as other manufactures.

Back to pins, we look at the OE (Output Enable) and CE (Chip Enable). The CE pin enables and disables the data output. When disabled most of the chip is in a low power sleep mode. The *access time* of a chip is given from the time CE becomes active until data appears.

The access time using the OE pin is a lot shorter than CE because the OE pin disables only the data output pins, but not the rest of the chip. This allows data to be accessed at a much higher speed at the cost of increased power consumption.

When programming, we use the *Vpp* line to supply the *programming voltage* to the EPROM. Other times, Vpp is kept at +5.

Most EPROMs have 8 *data lines* (bits D0 - D7) forming a byte wide *data bus*, some have 16 *data lines* (bits D0 - D15) which makes it a *word wide* data bus. Enabling both CE and OE causes all the bits to appear on the data lines; you can not enable individual bits. The data of the byte or word, appears on these lines as 0's (about .4 Volts) or 1's (about 4 Volts). The data lines of an EPROM are bi-directional; they are outputs when you are reading the EPROM and inputs when you are programming it.

The *address lines* form an *address bus* made of inputs that together select which location you want to read or program. Address lines start at A0 the LS (least significant) going up (A1, A2, A3□) with the address space doubling in size with each additional pin.

Next, we have to consider the ROM compatible EPROM's. These have two pins, OE and A16, that are switched around. 1Mega-bit Mask ROMs are in 28 pin packages and 1Mega-bit EPROMs are in 32 pin packages. By switching the two pins around, we can use a EPROM for testing and developing purposes. Once the code is completely debugged and you are ready to switch to masked ROMs, there is no reason to change the board layout. Simply put the EPROM in two pins lower in the socket (plug pin 1 into pin3, pin 2 into pin4 . etc.). Companies that make video games and Laser printers use ROM compatible EPROM's in great quantities. Part numbers to look for are 27100, 271000 & 27301. Remember, they are *not* pin compatible with 271001 or 27010.

Programming an EPROM

All EPROM bits are set to a 1 when erased (each byte = FFh). When programming an EPROM you can only change the 1s to 0s or clear bits. If you program a byte to 10100101 (A5h), only the 0s locations change state. Once a bit is changed to 0, you can't change it back to a 1 without erasing the whole EPROM. We can go back and change any of the 1s to 0s in a second programming session called patching. For example, the A5h byte (10100101) can be changed to 21h (00100001) but it can not be changed to F5h (11110101). Once more, remember programming clears bits, but only erasing the EPROM sets them.

Programming an EPROM requires a special program voltage called Vpp (25, 21 or 12.5 Volts). Parts that use a Vpp of 12.5 also require us to raise the Vcc from 5v to 6v during programming

Once the programming voltage is applied, we apply the address location and data to be programmed into the EPROM. The data comes from an EPROM image in a memory buffer This Image is loaded from another EPROM or disk file. Then using one of the programming algorithms, we apply a programming pulse. The best deal in programmers is the pocket programmer (http://secure.transtronics.com/EPROM_Programmer.html) .

Programming algorithms

Standard	This simplest of the algorithms, uses a set time of a 50mS programming pulse on each byte. After all bytes are programmed each byte is then verified or compared to the EPROM image in the buffer. If any byte does not verify then the EPROM is considered bad.
Fast or intelligent	This starts with a 1mS programming pulse to program a byte. Then the byte is checked against the programming data for verification. If it does not verify, the program pulse is doubled (keeping track of the accumulated pulse time) and the byte is programmed again. This process is kept up until the byte is programmed. The byte is then programmed again with a pulse time equal to the accumulated pulse multiplied by 3 to be sure it sticks. If the data never verifies and the pulse is has grown to 25mS, the EPROM is bad.
Snap!	Extreme caution must be used with this protocol. It will only work on the newer EPROMs. The Vpp & Vcc voltages are raised close to the maximum levels with a Vpp of 13v & Vcc of 6.5V. Using a 100uS programming pulse, the whole EPROM is programmed (similar to the Standard algorithm). Next a verification of the whole EPROM is done, if a byte does not verify it is programmed again using a 100Us pulse until the byte is successfully programmed. After 10 passes without verification the EPROM is considered bad. On the down side it also requires a very high accuracy Vpp supply and is not found on many programmers. (Some low end EPROM programmers have attempted this algorithm with disastrous results)
32 bit	The 32 bit algorithm requires newer EPROMs of 1Meg bit or larger (they have different internal workings). This starts by sending 4 bytes of data to the EPROM that are latched internally. Then a programming pulse is applied which programs all 4 bytes at once. After all location are programmed a verification pass is done and any bytes that do not verify are programmed again using the 8 bit Fast algorithm. The 32 bit algorithm programs large EPROMS 4 times faster, but the programming current is also 4 times as much and thus requires a special Vpp supply.

Erasing an EPROM

An EPROM can only be erased using a UV light. ***Read and follow the safety instructions that come with EPROM erasers and light bulbs***.

I have been asked:

Why do I need an EPROM eraser, can't I leave them out in the sun?

> Ah, yes you can, but it can take about 3 weeks of sunny weather!.

Well then, how about a fluorescent light?

> No problem, if you don't mind waiting a year or so.

Well I have a black light?

> The answer is still no.

An EPROM requires a specific frequency of UV 253.7 nanometers (2537 angstroms). In fact the high frequency UV light used will not pass through plastic or most glass. To pass this light, the window in the EPROM is made of quartz crystal. The correct frequency of a light source alone won't guarantee proper EPROM erasure. The intensity of the light source combined with the distance from the light source determines the intensity of the exposure. Light intensity varies inversely with the distances from the source. (The longer the distance the lower the intensity of the light.) You will find that the closer the chip is to the light source, the faster the EPROM will erase. Chip manufacture tell us, "the EPROM should be 1" from the light source with an intensity of 12mW/Cm2". The manufactures state a 1" distance to ensure that the light intensity is even over the entire EPROM, defused, with no shadows over any part of the EPROM silicon. (Yet, I have successfully used a 1/2" distance to speed my prototype work.)

Besides bulb to chip distance, bulb age also effects exposure time. Be aware of the age of the EPROM technology you are erasing. Older 1.3 micron technology takes longer to erase than .7 micron technology. With all these variables, the best way to determine exposure time is to run an empirical exposure test. First, expose the EPROMs for 1 minute and test for erasure. If they are not erased, expose them again for another minute. Keeping track of the total exposure time, repeat this process until they are erased. Now take the total time and multiply by 1.5. For example, if it takes 3 minutes we should use a 4.5 minutes exposure time to ensure good erasure without over erasing. Remember if you leave an EPROM in the eraser too long, it can remain erased forever and will no longer program.

Every time an EPROM is programmed and erased it wears out a little. Erasing cycles will slow down access times, but this is usually of no consequence unless you erase them an extreme number of cycles or for an excessive exposure time. An EPROM with slow access time, may still program and work fine on an EPROM programmer, but beware that the EPROM programmer does not require or test for a fast access time. A worn out EPROM may program correctly yet fail to work in the equipment that requires a fast access time.

Making Your Own Eraser? Or why not to

Germicidal light bulbs are used in hospitals for sterilizing equipment by killing germs. Germs are living cells. We are also made of living cells. ***Read and follow the safety instructions that come with EPROM erasers and light bulbs***. You are not likely to go blind from accidentally looking at the light for few seconds, but germicidal lamps should only be operated in a ***safety-interlocked enclosure.***

If you want to make your own eraser, you would need to get a germicidal light bulb that looks like a fluorescent lamp without the white coating on the inside. These are quartz tubes (remember the UV will not pass through glass). At a light supply house you can get a G15T (a 15 Watt lamp) The " G " stands for germicidal the same as " F " stands for fluorescent (even though it has no phosphors &*???) . Just the light bulbs costs about $30.00, and you will also need a 15 Watt fixture. Why not get a small ready-made eraser (http://secure.transtronics.com/D_ERASE.htm) for the same price? They are inexpensive, easier to use and have built in safety interlocks.

The Difference Between C & Non-C EPROMs

The only difference between a 27256 and a 27C256 is that the 27256 uses NMOS while the 27C256 uses CMOS technology. CMOS only consumes appreciable power when a signal is switching. NMOS uses N-Channel FET's with resistor elements while CMOS avoids the power wasting resistors by using both N and P-channel FET's. Thus CMOS avoids the production of heat allowing tighter placements of transistors than NMOS is capable of. The high density placement of CMOS reduces the interconnect path lengths and thus increases the speed. CMOS also shines when there is limited power such as in a battery system.

Some people have problems when programming CMOS EPROMs on older programmers because of the differences in programming voltages (CMOS has a 12.5V Vpp). CMOS EPROMs also require a supply voltage (Vcc) of *exactly* 6 Volts. CMOS parts are easier to erase but prone to die if over exposed to UV light.

Decoding EPROM Numbers

27(C)XXX are EPROMs or OTPROMSs.

57(C)XXX are EPROMs or OTPROMSs that allow the lower 8 address lines to be multiplexed with the Data lines. (Some MCU's multiplex the lower address and data lines together). These parts still program like 27(C)XXX EPROMs on the Pocket Programmer (http://secure.transtronics.com/EPROM_Programmer.html) because the software algorithm takes this into account.

28(C)XXX are EEPROMs with the C standing for CMOS.

28FXXX are Flash EEPROMs with the F standing for Flash. Do not confuse these with EEPROMs as they are not.

Cross list

The below parts may be the same -(according to manufactures cross guides) there may be differences including the programming algorithm used.

Manufacturer	AMD	AMIC		Atmel	Fujitsu	Hitachi	Hynix	INTEL	Mitsu-bishi	NEC	NSC	SGS	SST	ST micro	TI
prefix	AM	AE	ASD	AT	MBM	HN	HY	D	M5M	UPD	NM	M	SST	M	TMS
32K	2732				2732	2732		2732	27C32	D2732	27C32Q	2732			2732
64K	27C64				27C64	27C64		27C64	27C64	27C64	27C64Q	27C64			27C6
128K	27C128				27C128	27C128		27C128	27C128	27C128	27C128Q	27C128			27C1
256K	27C256			27C256	27C256	27C256		27C256	27C256	27C256	27C256Q	27C256	27SF256	27256	27C2
				28HC256											
512K	27C512			27C512	27C512	27C512		27C512	27C512	27C512	27C512Q	27C512	27SF512	27C512	27C5
	28F512			29C512									29EE512		
				49C512									39SF512	29F512	
1MEG	27C010		29F1008	27C010	27C1001	27C010		27C010	27C101	27C1001	27C010Q	27C1001	27SF010	27C1001	27C0
				29C010											
				49F001				28F001					39SF010	29F010	
	28F010			49F010				28F010							
				29C010									29EE010		
2MEG	27C020		29F2008	27C020	27C2001	27C020		27C020	27C201	27C2001	27C020Q	27C2001	27SF020	27C2001	27C0
	29F002			29C020 49F020 49F002			29F002	28F002					29EE020 39SF020	29F002	
4MEG	27C040				27C4001	27C040		27C040	27C401	27C4001	27C040Q	27C4001			27C0
	29F040				29C040 49F040			28F004					28SF040 39SF040	28SF040 29F040	
	29F040				29F040		29F040							29F040	
x16	29F400				29F400		29F400								
8MEG		29010 29001 290011												27C801	
	29F080				29F080		29F080								
x16	29F800				29F800		29F800								
16MEG		29002 290021													
32MEG		29400 29040A													
64Meg		29800													