



AP-316

**APPLICATION
NOTE**

Using Flash Memory for In-System Reprogrammable Nonvolatile Storage

**SAUL ZALES
DALE ELBERT
APPLICATIONS ENGINEERING
INTEL CORPORATION**

January 1996

Order Number: 292046-004



Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641
or call 1-800-879-4683

USING FLASH MEMORY FOR IN-SYSTEM REPROGRAMMABLE NONVOLATILE STORAGE

CONTENTS	PAGE	CONTENTS	PAGE
1.0 INTRODUCTION	1	4.0 SOFTWARE DESIGN FOR ISW	8
1.1 PROM Programmer vs System-Processor Controlled Programming	1	4.1 System Integration — Boot Code Requirements	8
1.2 Information Download and Upload	1	4.1.1 ISW Flag Check	9
Version Updates (Download)	1	4.2 Communication Protocols and Flash Memory ISW	9
Data Acquisition (Upload)	1	4.3 Data Accumulation Software Techniques	10
2.0 DEVICE FEATURES AND ISW APPLICATION CONSIDERATIONS	2	4.4 Reprogramming Routines	10
2.1 Flash Memory Pinouts	2	4.4.1 Quick-Erase Algorithm	10
2.2 Command Register Architecture	4	Algorithm Timing Delays	10
Simplified Processor Interface	4	High Performance Parallel Device Erasure	11
Command Register Reset	5	4.4.2 Quick-Pulse Programming Algorithm	12
Data Protection on Power Transitions	5	Algorithm Timing Delays	12
2.3 V _{PP} Specifications	6	High Performance Parallel Device Programming	12
3.0 HARDWARE DESIGN FOR ISW	6	4.4.3 Pulse Width Timing Techniques	13
3.1 V _{PP} Generation	6	Software Methods and Examples	13
3.1.1 Regulating Down from Higher Voltage	6	Hardware Methods	13
3.1.2 Pumping 5V up to 12V	6	5.0 SYSTEM DESIGN EXAMPLE: AN 80C186 DESIGN	14
3.1.3 Absolute Data Protection — V _{PP} On/Off Control	7	6.0 SUMMARY	15
3.1.4 Writes and Reads during V _{PP} Transitions	7		
3.1.5 Other V _{PP} Considerations	7		
3.1.6 V _{PP} Circuitry and Trace Layout	8		
3.2 Communications — Getting Data to and from the Flash Memory	8		



CONTENTS PAGE

**APPENDIX A:
ON BOARD PROGRAMMING DESIGN
CONSIDERATIONS** A-1

**APPENDIX B:
V_{pp} GENERATION CIRCUITS** B-1

**APPENDIX C:
LIST OF DC-DC CONVERTER
COMPANIES** C-1

CONTENTS PAGE

**APPENDIX D:
DETAILED PARALLEL ERASE FLOW
CHART** D-1

**APPENDIX E:
DETAILED PARALLEL
PROGRAMMING FLOW CHART** E-1

**APPENDIX F:
DETAILED SYSTEM SCHEMATICS** F-1



1.0 INTRODUCTION

Intel's ETOX™ II (EPROM tunnel oxide) flash memory technology uses a single-transistor cell to provide in-system reprogrammable nonvolatile storage. Reprogramming entails electrically erasing all bits in parallel and then randomly programming any byte in the array. This new technology offers designers alternatives for two of industry's needs: 1) a cost-effective means of updating program code; and 2) a solid-state approach for non-volatile data accumulation or storage.

This application note:

- introduces you to the concepts of in-system writing;
- discusses the hardware and software considerations for reprogramming flash memories in-system;
- offers a checklist for integrating Intel's flash memories into microprocessor- or microcontroller-based systems; and
- shows an example of an 80C186 design which incorporates flash memory.

1.1 PROM Programmer vs System-Processor Controlled Programming

While soldered to a printed circuit board, one of two sources controls flash memory reprogramming: 1) a PROM programmer connected to the board, or 2) the system's own central processing unit (CPU). These are called on-board programming (OBP), and in-system writing (ISW), respectively. With OBP, the PROM programmer supplies the programming voltage (V_{pp}) and the programming intelligence; with ISW, V_{pp} is generated locally and the system itself drives the reprogramming process. Both methods offer a variety of benefits. However this application note focuses on ISW.

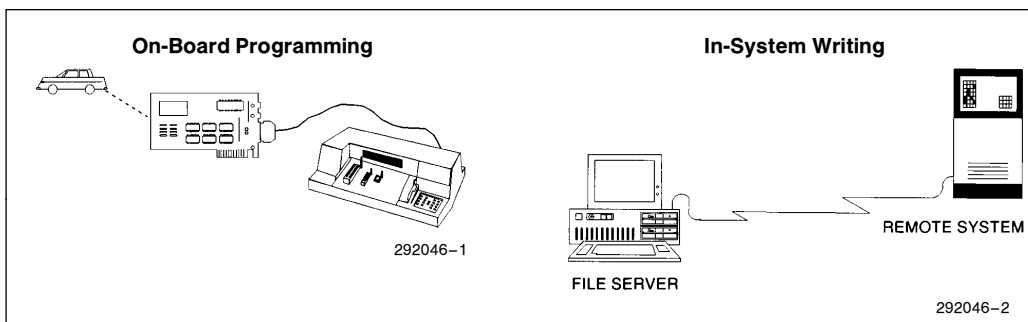


Figure 1. These diagrams illustrate OBP and ISW. In OBP, a PROM programmer updates a system's flash memory. The ISW diagram shows a host updating remote flash memory via serial link. The remote system performs the flash reprogramming with its own CPU.

NOTE:

See Appendix A for OPB design considerations.

1.2 Information Download and Upload

ETOX II flash memory technology programs extremely quick, permitting "on-the-fly" programming with unbuffered 19.2K baud data input. The remote ISW system handles the serial communication link for the host interface, as well as the flash memory reprogramming.

Version Updates (Download)

Flash memories enable code version updates using simple hardware designs. Beyond the basic system, a local V_{pp} supply is all that is needed for remote code download.

A central host computer can download program code to many remote systems. Flash memory offers this capability without the drawbacks of other technologies. It is solid-state and nonvolatile, thus eliminating mechanical component wear-out (common with disk drives) and the risk of losing updates (a concern with battery-backed RAM). These aspects of flash memory offer major advantages in automated factories, remote systems, portable equipment and other applications. Finally, flash memories provide this capability at a much lower cost than byte-alterable EEPROM and battery-backed SRAM.

Data Acquisition (Upload)

Intel's flash memories allow single-byte programming for data accumulation applications. A remote data-logger uploads its information to a central host via serial link. The flash memory device is then in-system erased

for resumption of data acquisition. This is useful in an advanced electrical power meter, for example. It could be configured to track and monitor power usage and report the data to a central computer for billing and utility management. This reduces the cost of manual door-to-door meter reading.

2.0 DEVICE FEATURES AND ISW APPLICATION CONSIDERATIONS

This section gives a brief overview of Intel's flash memory features and explains how they facilitate ISW design.

2.1 Flash Memory Pinouts

The 32-pin DIP memory site is forward-compatible from the 256K bit to the 2 Mbit flash memory density. It fits into the 27C010 Mbit EPROM pinout and requires no multiplexed pins. Also, with just a single circuit-board jumper trace, a 28-pin EPROM can be placed in the lower pins of the 32-pin flash memory site. (See Figures 2A and 2B, Flash Memory Pinouts.) For more information on intertechnology pin compatibility see Ap Brief AB-25.

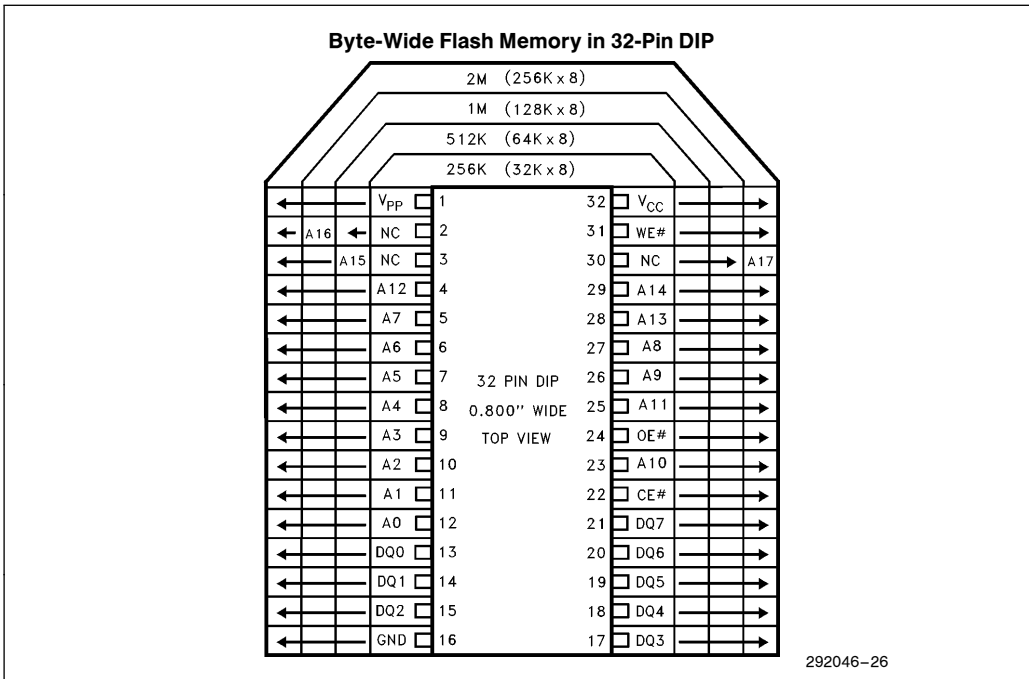


Figure 2A. Flash Memory Pinouts

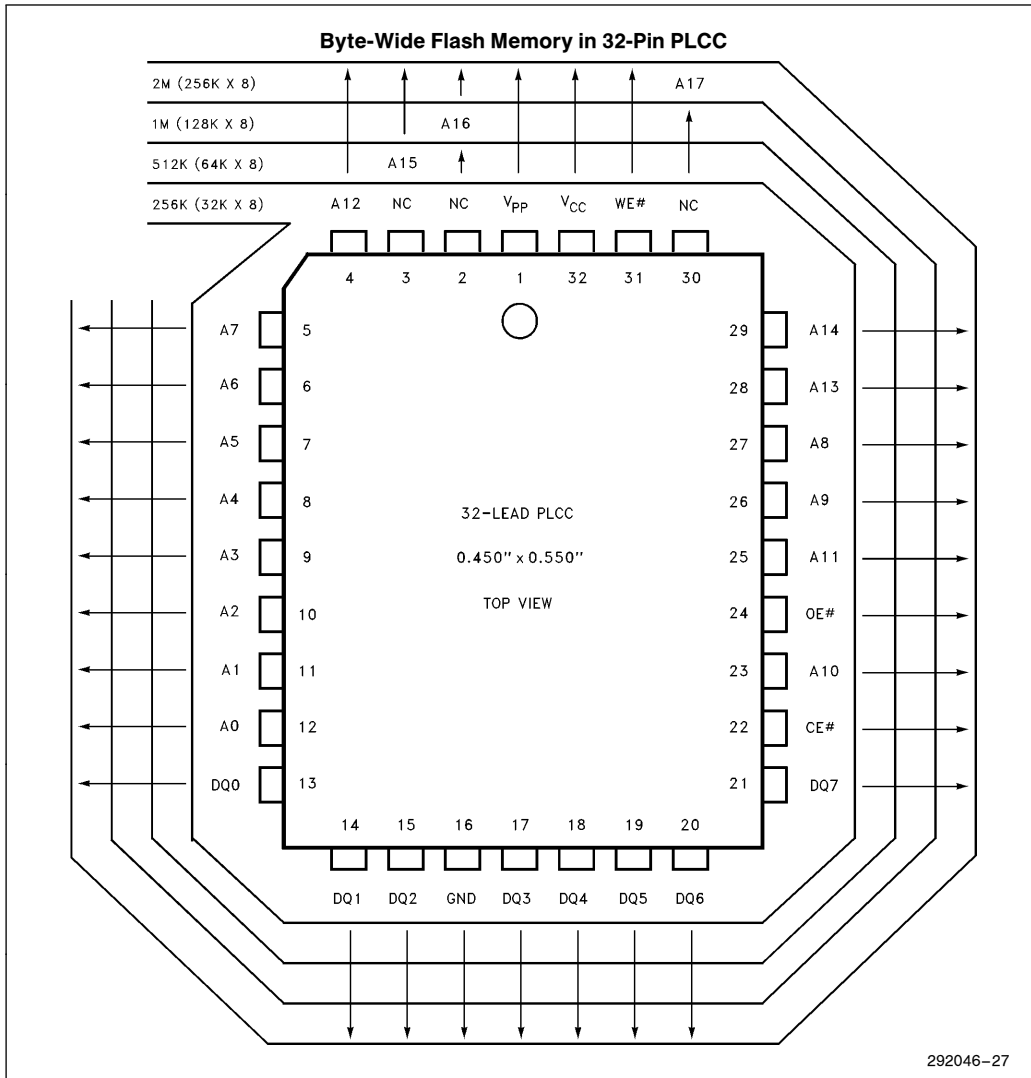


Figure 2B. Flash Memory Pinouts



Table 1. Command Register Instructions

Command	Bus Cycles Req'd	First Bus Cycle			Second Bus Cycle		
		Operation	Addr(1)	Data(2)	Operation	Addr(1)	Data(2)
Read Memory ⁽³⁾	1	Write	X	00H	Read	Valid	Valid
Read Intelligent Identifier	1	Write	X	90H	Read	00/01H	ID
Set-Up Erase/Erase	2	Write	X	20H	Write	X	20H
Erase Verify	2	Write	EA	A0H	Read	X	EVD
Set-Up Program/Program	2	Write	X	40H	Write	PA	PD
Program Verify	2	Write	X	C0H	Read	X	PVD
Reset ⁽³⁾	2	Write	X	FFH	Write	X	FFH

NOTES:

1. Addresses are latched on the falling edge of the Write-Enable pulse.

EA = Address of memory location to be read during erase verify.

PA = Address of memory location to be read during program verify.

2. EVD = Data read from location EA during erase verify.

PD = Data to be programmed at location PA. Data is latched on the rising edge of Write-Enable.

PVD = Data read from location PA during program verify. PA is latched on the Program command.

3. The second bus cycle must be followed by the next desired command register write, given the proper delay times.

2.2 Command Register Architecture

Simplified Processor Interface

Intel's command register architecture simplifies the processor interface. The command register allows CE#, WE#, and OE# to have standard read/write functionality. All commands such as "Set-up Program" or "Program Verify" can be written with standard system timings. Raising V_{PP} to 12V enables the command register for memory read/write operation, while lowering V_{PP} below V_{CC} + 2V restores the device to a read only memory.

Writing to the register toggles an internal state-machine. The state-machine output controls device functionality. Some commands require one write cycle, while others require two. The command register itself does not occupy an addressable memory location. The register simply stores the command, along with address and data needed to execute the command. With this architecture, the device expects the first write cycle to be a command and does not corrupt data at the specified address. Table 1 contains a list of command register instructions.

The following sections describe the commands in relation to device operation. For more information on the command register see the appropriate flash memory data sheets, and Section 4.4 "Reprogramming Routines".

Read Memory Command—00H

This command allows for normal memory read operations with V_{PP} turned on. After writing the command and waiting 6 μs, the CPU can read from the memory

at system speeds. Once placed in the read mode no further action is required on the command register for subsequent read operations.

Read Intelligent Identifier Command—90H

Most PROM programmers read the device's intelligent identifier to select the proper programming algorithm. On EPROMs, raising A9 to the V_{PP} level configures the device for this purpose. Since this is unacceptable in-system, you can read the flash memory intelligent identifier by first writing command 90H. Follow this by reading address 0000 and 0001H for the manufacturer and device ID. Reset the device with the Read Memory command after you have read the identifier.

Set-Up Erase/Erase Commands—20H

Write this command (20H) twice in succession to initiate erasure. The first write cycle sets up the device for erasure. The device starts erasing itself on the second command's rising edge of Write-Enable. Erasure is stopped when the CPU issues the Erase Verify command or when the device's integrated stop timer times out. Integrated stop timers provide a safety net for complex system environments. In these environments, s/w timer accuracy may be difficult to achieve. Some method of timing is still required, however the timer need only meet a minimum specification (10 ms). This is far easier than calibrating a timer to meet both a minimum and maximum specification (10 ms ± 500 μs).

NOTE:

Prior to erasure, it is necessary to program all bytes to the same level (data = 00H). See the Quick-Erase algorithm for more details.

Erase Verify Command—A0H

The erase command erases all bytes of the array in parallel. After each erase operation, all bytes must be verified to see if they erased. Write the Erase Verify command (A0H) to stop erasure and setup verification.

Alternatively, you may allow the internal stop timer to halt erasure. You must still issue the Erase Verify command to set up verification.

The device latches the address to be verified on the falling edge of WE# and the actual command on the rising edge. Wait 6 μ s before reading the data at the address specified on the previous write cycle.

The flash memory applies an internally-generated reference voltage to the addressed byte. Reading 0FFH from the addressed byte in this mode indicates that all bits in the byte are erased with sufficient margin to V_{CC} and temperature fluctuations.

If the location is erased, then repeat the Erase Verify procedure for the next address location. Write the command prior to each byte verification to latch the byte's address. Continue this process for each byte in the array until a byte does not return 0FFH data, or the last address is accessed.

In the case where the data read is not 0FFH, perform another erase operation. (Refer to Set-up Erase/Erase). Continue verifying from the address of the last verified byte. Once you have accessed the last address, erasure is complete and you can proceed to program the device. Terminate the erase verify operation by writing another valid command (e.g., Program Set-up).

Set-up Program/Program Commands—40H

Write this command (40H) twice in succession to initiate programming. The first write cycle sets up the device for programming. The device latches address and data on the falling and rising edges of the second write cycle, respectively. It also begins programming on the rising edge. You stop the programming operation by issuing the Program Verify command, or by allowing the integrated program stop timer to time out. This timer works similar to the erase stop timer. Again, a minimum specification replaces a tougher minimum/maximum combination (10 μ s–25 μ s).

Program Verify Command—C0H

Flash memory devices program on a byte-by-byte basis. After each programming operation, the byte just programmed must be verified. Write the Program Verify command (C0H) to stop programming and set-up verification. Should your software allow the integrated stop timer to halt programming, the software must resume the algorithm with the Program Verify command. The

device executes this command on the rising edge of Write-Enable. The program Verify command stages the device for verification of the byte last programmed. No new address information is latched.

The flash memory applies an internally-generated reference voltage to the addressed byte. Wait 6 μ s for the internal voltages to settle before reading the data at the address programmed. Reading valid data indicates that the byte programmed successfully.

Command Register Reset—FFH

Flash memories reset to the read mode during power-up, and remain in this mode as long as V_{PP} is less than V_{CC} + 2V. If your system leaves V_{PP} turned-on during a *system reset*, then incorporate a command register *device reset* into the hardware initialization routines. This is necessary because the CPU might be controlling programming or erasure when the system reset hits.

Write the reset command (0FFH) twice in succession to reset the device. The double write is necessary because of the state-machine reprogramming structure. For example, suppose the *system* is reset after a Set-up Program command. The flash memory state machine expects the next write cycle to contain valid address and data for programming, followed by another write cycle for program verification. The first Reset command will be mistaken for program data but will not corrupt the existing data. This is because the command (data = 0FFH) is a null condition for flash memory programming. Only data bits programmed to zero pull charge onto the memory cell and change the data. The second write cycle actually resets the device to the read function. Following the second reset cycle, you can write the next command (Read, Program Set-up, Erase Set-up, etc.).

If the V_{PP} supply is turned off upon system reset, the software reset is not required. The flash memory will reset itself automatically when V_{PP} powers down.

Data Protection on Power Transitions

The command register architecture offers another benefit in addition to simplified processor interface—during system power-up and power-down it protects data from corruption by unstable logic. Erasure or programming require V_{PP} to be greater than V_{CC} + 2V and the proper command sequence to be initiated. For example the CPU must write the erase command twice in succession. The odds of this occurring randomly are slim. Additionally, should V_{PP} ramp to 12V prior to V_{CC} ramping past 2.5V, the device will lock out all spurious writes and internally block 12V from the flash memory cells. For even greater security, you can switch V_{PP} as discussed in Section 3.13.

2.3 V_{pp} Specifications

Flash memories, like EPROMs, require a 12V externally-generated power supply for reprogramming. Intel's V_{pp} specifications 12.0V ±0.6V (5%) is compatible with most off-the-shelf (or available in-system) power supplies. (Note, Section 3.1 discusses V_{pp} generation techniques, and Appendix B shows different circuit alternatives.)

It is essential to use the specified V_{pp} when reprogramming the flash memory device. Once the command to erase, program, or verify is issued, the device internally derives the required voltages from the V_{pp} supply. The command register controls selection of internal reference circuitry tapped off of V_{pp}. An improper V_{pp} level causes the references to be wrong, degrading the performance of the part.

(When programming U.V. EPROMs, V_{CC} is raised to 6.5V. On flash memories, the V_{pp} reference circuitry and command register architecture provide the same function while keeping V_{CC} and V_{pp} at static levels. An incorrect V_{CC} level during U.V. EPROM programming poses similar hazards to improper V_{pp} levels on flash memories.)

The hardware design section discusses various methods for generating V_{pp}.

3.0 HARDWARE DESIGN FOR ISW

Covered in this section are the following:

- Description of ISW-specific functional system blocks including memory requirements
- V_{pp} generation techniques
- Communication Considerations

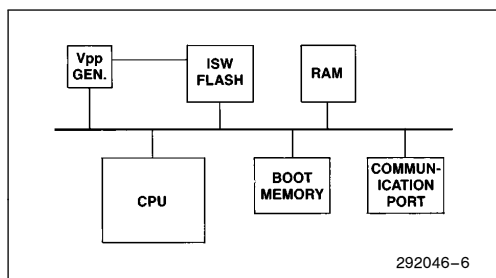


Figure 3. System Block Diagram

System Level Hardware Requirements for ISW:

- processor or controller
- limited amount EP/ROM or other flash memory devices for boot code, communications s/w, and reprogramming algorithms

- limited amount of RAM for variable storage (i.e., stacks, buffers, and other changing parameters)
- data import capability (i.e., serial line, LAN, floppy disk)
- flash memory for nonvolatile code or data storage needs
- V_{pp} generator or regulator

All of the functional blocks in Figure 3 are typical of any embedded or reprogrammable system with the exception of the V_{pp} generator. Some microcontrollers have on-chip EP/ROM, RAM and a serial port. With these devices, implementation of the ISW capability requires little additional hardware.

The next section discusses V_{pp} generation techniques and communications design considerations.

3.1 V_{pp} Generation

A static V_{pp} is needed to reprogram flash memories. The V_{pp} voltage can be generated by:

- 1) regulating it down from a higher voltage;
- 2) pumping it up from a lower voltage (i.e., charge pump, DC/DC converter, etc.); or
- 3) designing or specifying the system's 12V supply with the required ISW tolerances and specifications.

Sufficient current for reprogramming should be considered when selecting your V_{pp} generation option. Parallel reprogramming for flash memory in 16-bit or 32-bit systems will require, respectively, 2X or 4X additional current capability.

3.1.1 REGULATING DOWN FROM HIGHER VOLTAGE

V_{pp} is obtained from a higher voltage by using a linear regulator. Given the higher voltage, regulation offers the least expensive method of generating V_{pp}. Standard three terminal 12V ±1%, ±2%, ±4% non-adjustable regulators are available off-the-shelf. Some regulators have on/off control built-in. (See Appendix B, V_{pp} Circuit #1.) All regulators require a minimum input voltage greater than the output voltage. (See Appendix B, V_{pp} Circuit #2 and #3.)

3.1.2 PUMPING 5V UP TO 12V

V_{pp} can be obtained by pumping V_{CC} and regulating it to the proper voltage. A voltage charge-pump can be designed and built by using a charge-pump integrated circuit and some discrete components (see Appendix B, V_{pp} Circuit #4) or by using a monolithic DC/DC converter (see Appendix A, V_{pp} Circuit #5).

When using adjustable circuits containing discrete components, design the output voltage so it falls within the V_{pp} specifications for all corners of the components'

skew (i.e., $V_{CC} \pm 10\%$; $R_x \pm 1\%$, $R_y \pm 1\%$, etc.). Include the resistors' temperature coefficients in the calculation matrix. Note that each of the various components can add error to the V_{PP} supply.

The monolithic DC/DC converter shown in Appendix B Circuit #5 fits into a 24-pin socket. It offers the advantages of close temperature tracking and ease of implementation. It has also been characterized at temperatures and meets all the V_{PP} specifications. Appendix C contains a partial list of vendors selling DC/DC converters.

Most DC/DC converters are only 50–60% efficient, so heat dissipation may be a concern. Some discrete boost circuits such as Appendix B, Circuit #4, offer much higher efficiency (70–85%). Circuit #4 as shown can supply 200 mA. Smaller inductor and capacitor component values and higher frequency boost converters can be used where less power is required. For example, designs which reprogram one or two flash memories simultaneously might use the LT1172. (Contact Linear Technologies for more information.)

In all V_{PP} generation methods, a capacitor on the input voltage terminals reduces the output noise voltage. Some power supplies (Appendix B, Circuits #3 and #4) specify a large-valued capacitor to decrease the Effective Series Resistance (ESR). Place a 0.1 μF capacitor within 0.25 inches of each flash memory's V_{PP} input (in addition the one on the V_{PP} generator's input).

NOTE:

The ESR is inversely proportional to the capacitance value and the rated working voltage. To lower the ESR choose a capacitor with a large capacitance and a high working voltage (i.e., above 100V).

3.1.3 ABSOLUTE DATA PROTECTION— V_{PP} ON/OFF CONTROL

With V_{PP} below $V_{CC} + 2V$ or V_{CC} below 2.5V, internal circuitry disables the command register and eliminates the possibility of inadvertent erasure or programming. Switching the V_{PP} supply off provides the secondary benefits of improved power and thermal management.

There are two ways to switch V_{PP} on and off:

- 1) directly switch the V_{PP} generator's output, or
- 2) switch the input voltage supplying the regulation circuit.

Any switching circuit will cause a voltage drop, so choose a switch with this drop in mind. Some power supplies have asymmetrical tolerances on 12V (i.e. +5%, -4%). Flash memory allows the 12V supply to drop as low as -5%. The 1% difference between the supply and the device requirement allows the switch to have an ON resistance voltage drop of 0.12V. Continuing with this example, assume the system only reprograms one flash memory at a time. The current through

the switch into the flash is $I_{PP} = 30 \text{ mA}$. Solving for the allowable resistance across the switch: $R = V/I = (0.12V)/(30 \text{ mA}) = 4 \text{ Ohms}$. See Figure 4. Example Voltage Drop Across Switch. Note, one can reduce the effective $R_{DS} (ON)$ by placing 2 or more FETs in parallel if necessary.

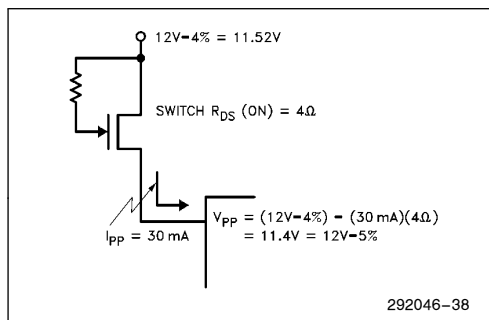


Figure 4

Controlling the input voltage of a DC/DC converter with a MOSPOWER FET is another straightforward approach. (See Appendix B, Circuit #5.) Choose the FET switch carefully. It should have a very low on-resistance to minimize the voltage divider effect of the converter and FET switch. If the voltage across the FET switch is too high, the converter will not have the proper input voltage to meet its specifications. Always design the switching circuit with sufficient margin to maximum V_{PP} and V_{CC} load currents.

3.1.4 WRITES AND READS DURING V_{PP} TRANSITIONS

After switching V_{PP} off, the CPU can read from the flash memory without waiting for the capacitors on V_{PP} to bleed off. To do this, write the Read Memory command prior to issuing the V_{PP_OFF} instruction. Alternatively, the device resets automatically to read mode when V_{PP} drops below $V_{CC} + 2V$.

Raising V_{PP} to 12V enables the command register. You must wait 100 ns after V_{PP} achieves its steady state value before writing to the command register. Remember that the steady state V_{PP} settling time depends on both the power supply slew rate and the capacitive load on the V_{PP} bus.

3.1.5 OTHER V_{PP} CONSIDERATIONS

The V_{PP} pin is an MOS input which can be damaged by electrostatic discharge (ESD). In OBP applications, an external power source supplies V_{PP} and then is removed. Electrostatic charge can build up on the floating V_{PP} pin. You can solve this problem by one of two means: 1) tie the pin to V_{CC} through a diode and pull-up resistor (Figure 5a) or through a resistor to ground (Figure 5b). With either approach use a 10 K Ω or larger resistor to minimize V_{PP} power consumption.

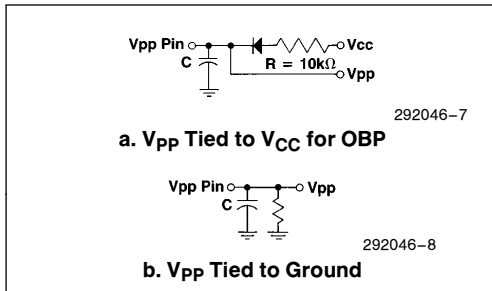


Figure 5

NOTE:

Typically EPROMs require V_{PP} to be within one diode drop of V_{CC} for optimal standby power consumption. Either approach can be used with the flash memory.

ISW applications do not require this ESD protection as most regulators and charge pumps contain a voltage divider on the output stage. A divider provides a resistive path to ground even with the supply turned off. (Note: check the schematics of the V_{PP} supply chosen.) However, if you directly switch the V_{PP} supply, add the resistor to ground; the switch isolates the V_{PP} pin and allows charge to build up.

3.1.6 V_{PP} CIRCUITRY AND TRACE LAYOUT

You should lay out V_{PP} circuitry and traces for high frequency operation since programming power characteristics exhibit an AC current component. Use the following standard power supply design rules:

- Keep leads as short as possible and use a single ground point or ground plane (a ground plane eliminates problems).
- Locate the resistor network (or a regulator) as close as possible to the adjustment pin to minimize noise pick-up in the feedback loop. The resistor divider network should also be as short as possible to minimize line loss.
- Keep all high current loops to a minimum length using copper connections that are as wide as possible. (This will decrease the inductive impedance which otherwise causes noise spikes.)
- Place the voltage regulator as close to the flash memory as practical to avoid an output ground loop. Excessive lead length results in an error voltage across the distributed line resistance.
- Separate the input capacitor return from the regulator load return line. This eliminates an input ground loop, which could result in excessive output ripple.

3.2 Communications—Getting Data to and from the Flash Memory

The flash memory does not care about the origin of the data to be programmed. The data could be downloaded from a serial link, parallel link, disk drive, or generated locally as in data accumulation applications.

While most systems communicate via serial link, sending a font to a printer's flash memory is an example of a parallel interface. In either format, designers must decide whether or not to buffer the incoming data. Error-free serial protocols will require buffering for reconstruction of information packets. With equal capacity of RAM and flash memory in a system, the download time would only be limited by the speed of the communication link.

Both worst case and typical analysis must be done for real time download and un-buffered programming. The maximum transmission rate is 19.2K baud assuming worst case programming times. The time between characters at 19.2K baud is 520 μ s; the worst case byte programming time is approximately 0.5 ms (including software overhead). Typical byte programming takes 16 μ s which allows for much higher unbuffered transmission rates. However, a single byte can take up to the full 400 μ s specified time (plus software overhead), so you should not base transmission rate on typical programming times.

Partial buffering or FIFO schemes can also be implemented to increase transmission rates. An argument for buffering is reduction of interconnect time and costs.

4.0 SOFTWARE DESIGN FOR ISW

Covered in this section are the following software requirements:

- system integration of ISW
- reprogramming considerations for single- and multiple-flash memory based designs.

4.1 System Integration—Boot Code Requirements

Boot code in remote systems should contain various ISW-specific procedures in addition to standard initialization and diagnostic routines.

The most dependable boot code for remote version updates contains some basic communications capability and the ISW reprogramming algorithms. Thus, a data-link disruption while reprogramming would be recoverable. For manufacturing flexibility, this boot memory could be an OBP 256K flash memory.

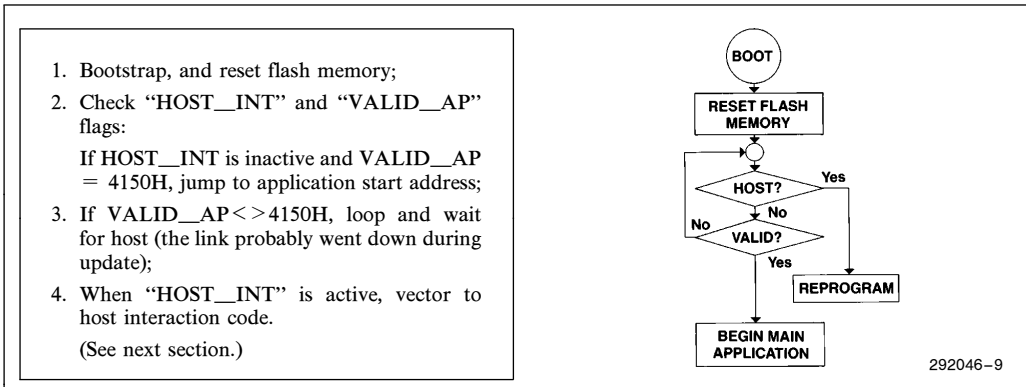


Figure 6. Example of ISW Integration to the Boot Sequence

An alternative to storing these routines in a separate boot device is storing them in the flash memory containing the program code. Prior to erasure, the CPU would transfer the ISW routines to system RAM and execute from there. This type of approach is suitable for battery-operated equipment or systems with back-up power supplies.

The communication link could be disrupted during reprogramming, leaving the device in an unknown configuration. Therefore, the boot code should reset the flash memory and check two ISW flags. The following section discusses the flag check concept.

4.1.1 ISW FLAG CHECK

After resetting the flash memories and initializing other system components, the CPU should check the communications link for a host interrupt. We will call this the HOST_INT flag. Had the communication link gone down prior to completion of downloading, then the host would have to re-establish contact to complete the task.

Assuming no HOST_INT request has been made, the boot protocol then checks a data sequence in the flash memory signifying a valid application (VALID_AP). You program this sequence into the memory array after confirmation of a successful download. If a download is interrupted midway through erasure or programming, then the VALID_AP flag locations will not contain the VALID_AP code. On the next system bootstrap the CPU recognizes this and holds up system boot until valid code is programmed. In Figure 6 an example flag protocol uses the VALID_AP sequence of 4150H (ASCII codes for "AP").

4.2 Communication Protocols and Flash Memory ISW

The remote download communications protocol must guarantee accurate transmission of flash memory in-

structions and program code. This protocol can be as simple as a read-back technique or as complex as an error-free transmission protocol. (See Figure 7 for possible system-level flash memory instructions.)

A simple read-back technique optimizes download for boot code memory needs and ease of implementation. The embedded CPU echoes the flash memory instruction (i.e., Erase or Program) to the host, and waits for a confirmation prior to execution. After programming the update, the remote system checks the update by transmitting it back to the host for confirmation. The remote system then programs the VALID_AP sequence. Note that programming and reading back 64 Kbytes at 19.2K baud takes about 0.57 minutes per direction:

$$(65,536 \text{ bytes}) * (10 \text{ bits/byte}) * (1 \text{ sec}/19.2 \text{ Kbits}) * (1 \text{ min}/60 \text{ sec}) = 0.57 \text{ minutes.}$$

Implementing either software- or hardware-based error-free communications protocol improves transmission efficiency. It eliminates the possibility of errant data being programmed if not buffered and checked, and optimizes the download process for transmission time. Additionally, file compression and decompression routines can improve the transmission rate.

- General ISW instructions include:**

 - STATUS CHECK
 - INITIATE REPROGRAMMING
 - MOVE ISW ROUTINES FROM FLASH MEMORY TO RAM
(If not resident in separate boot memory)

Data accumulation-specific commands include:

 - RETRIEVE DATA
 - ERASE FLASH MEMORY

Figure 7. Sample System-Level ISW Instruction Set

Status Check

The host should request a status update from the remote system prior to sending a reprogramming instruction. Depending on the response, the host may break the link and reconnect later, or it may send an erasure or data-upload command. This type of handshaking is necessary when system downtime for reprogramming might not be acceptable. An example of this is an automated factory where robots handle caustic chemicals.

4.3 Data Accumulation Software Techniques

Data can be accumulated in a remote environment with flash memory and then uploaded to a host computer for manipulation. You can adapt various standard data-logging techniques for use with flash memory. With any technique, you determine the next available memory location by reading for erased data (0FFH). This address would only be located once on system bootstrap and then recalled from RAM and incremented as needed.

Given a repeating data string of known length and composition, program start and stop codes at either end of the string. Do not pick 00H or 0FFH data for these codes because they are used during erasure. The start and stop codes enable the CPU to differentiate between available memory for logging and logged data equal to 00H or 0FFH.

For non-regular data input, you can address this same issue by programming the logged data followed by the variable identifier. Again, do not pick 00H or 0FFH data for the variable identifiers.

With any technique, the host computer separates and manipulates the data after the uploading operation.

4.4 Reprogramming Routines

Intel's ETOX flash memories provide a cost-effective updatable, non-volatile code storage medium. The reliability and operation of the device is based on the use of specified erasure and programming algorithms.

Intel offers reprogramming software drivers to make it easy for you to design and implement flash memory applications. The software is designed around the CPU-family architectures and requires minimal modification to define your system parameters. For example, you supply the memory width (8-bit, 16-bit, or 32-bit), system timing, and a subroutine for control of V_{pp} .

NOTE:

Contact your nearest sales office for details.

If you prefer to implement the algorithms yourself, they are outlined in the device data sheets. Command register instructions required for the various operations are included in the data sheet flow charts.

The following sections describe both single-device and multiple-device parallel reprogramming implementations.

4.4.1 Quick-Erase Algorithm

Flash memories chip-erase all bits in the array in parallel. The erase time depends on the V_{pp} voltage level (11.4V–12.6V), temperature, and number of erase/write cycles on the part. See the device data sheets for specific parametric influences on reprogramming times.

Note that prior to erasing a flash memory device the processor must program all locations to 00H. This equalizes the charge on all memory cells insuring uniform and reliable erasure.

Algorithm Timing Delays

The Quick-Erase algorithm has three different time delays:

- 1) The first is an assumed delay when V_{pp} first turns on. The capacitors on the V_{pp} bus cause an RC ramp. After switching on V_{pp} , the delay required is proportional to the number of flash memory devices times $0.1 \mu\text{F}/\text{device}$. V_{pp} must reach its final value 100 ns before the CPU writes to the command register. Systems that hardwire V_{pp} to the device can eliminate this delay.
- 2) The second delay is the "Time Out TEW" function, where TEW is the erase timing width. The function occurs after writing the erase command (the second time) and before writing the erase-verify command. The erase-verify command or the integrated stop timer internally stops erasure.
TEW for ETOX II flash memories is a minimum of 10 ms. This delay can be either software or hardware controlled. Either way, the minimum nature of the timing specification allows for interrupt-driven timeout routines. Should the interrupt latency be longer than the minimum delay specification, the stop timer halts erasure.
- 3) The third delay in the erase algorithm is a $6 \mu\text{s}$ time out between writing the erase verify command and reading for 0FFH. During this delay, the internal voltages of the memory array are changing from the

erase levels to the verify levels. A read attempt prior to waiting 6 μ s will give false data—it will appear that the chip does not erase. Repeatedly trying to erase verify the device without waiting 6 μ s will cause over-erasure. This delay is short enough that it is best handled with software timing. Again, note that the delay specification is a minimum.

High Performance Parallel Device Erasure

In applications containing more than one flash memory, you can erase each device serially or you can reduce total erase time by implementing a parallel erase algorithm.⁷ You save time by erasing all devices at the same time. However, since flash memories may erase at different rates, you must verify each device separately. This can be done in a word-wise fashion with the command register Reset command and a special masking algorithm.

Take for example the case of two-device (parallel) erasure. The CPU first writes the data word erase command 2020h twice in succession. This starts erasure. After 10 ms, the CPU writes the data word verify command A0A0h to stop erasure and setup erase verifica-

tion. If both bytes are erased at the given address, then the CPU increments the address (by 2) and then writes the verify command A0A0h again. If neither byte is erased, then the CPU issues the erase sequence again without incrementing the address.

Suppose at the given address only the low byte verifies FFh data? Could the whole chip be erased? The answer is yes. Rather than check the rest of the low byte addresses independently of the high byte, simply use the reset command to mask the low byte from erasure and erase verification on the next erase loop. In this example the erase command would be 20FFh and the verify command would be A0FFh. Once the high byte verifies at that address, the CPU modifies the command back to the default 2020h and A0A0h, increments the address by 2, and writes the verify command to the next address.

See Figure 8 for a conceptual view of the parallel erase flow chart and Appendix D for the detailed version. These flow charts are for 16-bit systems and can be expanded for 32-bit designs.

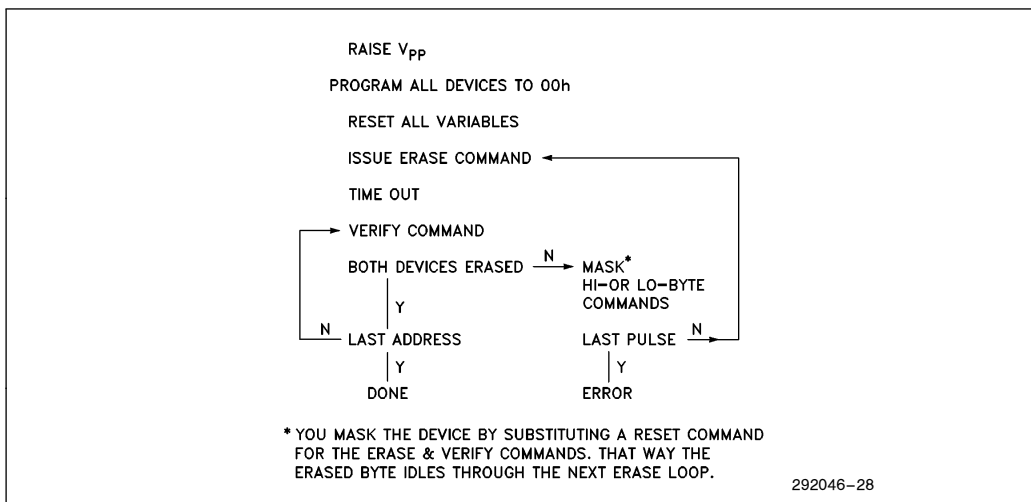


Figure 8. High Performance Parallel Erasure (Conceptual Overview)

7. Parallel Erasure and Programming require appropriate choice of V_{pp} supply to support the increased power consumption.

4.4.2 Quick-Pulse Programming Algorithm

Flash memories program with a modified version of the Quick-Pulse Programming algorithm used for U.V. EPROMs. It is an optimized closed-loop flow consisting of 10 μ s program pulses followed by byte verification. Most bytes verify after the first pulse, although some may require more passes through the pulse/verify loop. As with U.V. EPROMs, this algorithm guarantees a minimum of ten years data retention. See the device data sheets for more details on the programming algorithm.

Algorithm Timing Delays

The Quick-Pulse Programming algorithm has three different time delays:

- The first and third— V_{pp} set-up and verify set-up delays—are the same as discussed in the erasure section. In this case the third delay is for the transition between writing the Program Verify command and reading for valid data.

- The second delay is the “Time Out 10 μ s” function, which occurs after writing the data and before writing the program-verify command. This write command internally stops programming. The section entitled “Pulse Width Timing Techniques” gives 86-family assembly code for generating a 10 μ s timer routine.

High Performance Parallel Device Programming

Software for word- or double-word programming can be written in two different manners. The first method offers simplicity of design and minimizes software overhead by using a byte programming routine on each device independently. Here you increment the address by 2 or 4 when addressing 1 of 2 or 4 devices, respectively. The second method offers higher performance by programming the word or double-word data in parallel. This method manipulates the command register instructions for independent byte control. See Figure 9 for conceptual 2-device parallel programming flow chart and Appendix E for the detailed version.

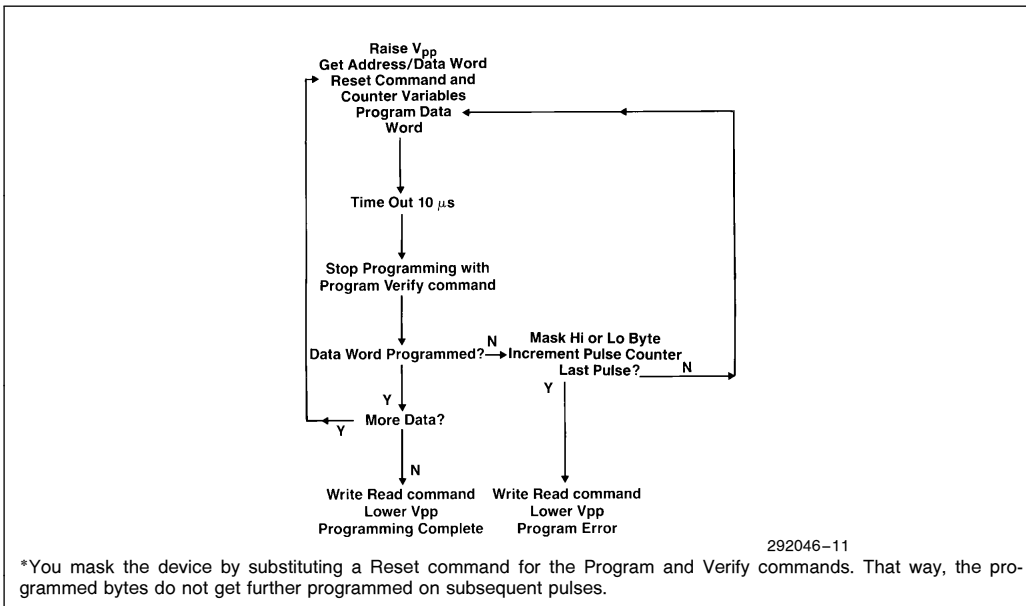


Figure 9. Parallel Programming Flow Chart (Conceptual Overview)

NOTE:

Word or double-word programming assumes 2 or 4 8-bit flash memory devices.

Parallel Programming Algorithm Summary:

- Decreases programming time by programming 2 flash memories (16 bits) in parallel. The algorithm can be expanded for 32-bit systems.
- Eliminates tracking of high/low byte addresses and respective number of program pulses by directing the CPU to write data-words (16-bit) to the command register.
- Maintains word write and word read operations. Should a byte on one device program prior to a byte on the other, the CPU continues to write word-commands to both devices. However, it deselects the verified byte with software commands. An alternative is to independently program high and low bytes using hardware select capability.

4.4.3 Pulse Width Timing Techniques

Software or hardware methods can be used to generate the timing required for erasure and programming. With either method you should use an in-circuit emulator (ICE™) and an oscilloscope to verify proper timing. Also remove the flash memory device from the system during initial algorithm testing.

Software Methods and Examples

Software loops are easily constructed using a number of techniques. Timing loops need to be done in assembly language so that the number of clock cycles can be obtained from the instructions.

In order to calculate a delay loop three things are needed—

- 1) processor clock speed,
- 2) clock cycles per instruction, and
- 3) the duration of the delay loop.

As an example, the 80C186 divides the input clock by 2. With a 20 MHz input clock the processor's internal clock runs at 10 MHz. This translates to a 100 ns cycle time. Delays can be made by loading the CX register with a count and using the LOOP instruction. The

LOOP instruction takes 16 clock cycles to execute per pass. It decrements the CX register on each pass and jumps to the specified operand until CX equals zero.

When writing a delay loop consider all instructions between the start and end of the delay. If a macro is written that delays 10 μs, add the clock cycles for all instructions in the macro.

Here is an example of a 10 μs delay and the calculation of the constant required for a 10 MHz 80C186.

```

WAIT_10 μs:
    push cx          ;10 clock cycles
    mov cx,DELAY    ;4 clock cycles
    loop $          ;see calculation
    pop cx          ;10 clock cycles
    
```

1. Start to End = 10 μs/cycle time
= 10 μs/100 ns
= 100 cycles
2. Loop Instruction = 100–24 cycles
= 76 cycles
3. Loop Cycles = 76
= (15 × [DELAY – 1] + 5)
4. Solving for DELAY = 6

Hardware Methods

Using an Internal Timer—

Many microcontrollers and some microprocessors have on-chip timers. At higher input clock speeds these internal timers have a resolution of 1 μs or better. The timers are loaded with a count and then enabled. The timer starts counting and when it reaches the terminal count a bit is set. The CPU executes a polling algorithm that checks the timer status. Alternatively, a timer-controlled interrupt can be used. After the timer has been set and the interrupt enabled, the CPU can be programmed to wait in idle mode or it could continue executing until the timed interrupt.



One thing to take into account when using interrupts is the time required for the CPU to recognize and interrupt request (interrupt latency). This is important when figuring the timer value, because the time seen by the part will be the programmed delay plus the minimum interrupt latency time.

The 80C186 has three 16-bit timers on-chip. Timer #2 can be a prescaler for the other two timers, which extends timers #0 and #1 range out to 2^{32} . By using two timers, 10 μ s pulses and 10 ms pulses can be easily achieved.

Using an External Timer—

External timers can take many forms. One popular example is the 82C54 (CHMOS Programmable Interval Timer) which has three 16-bit timers on-chip. One timer can be used as a prescaler for the others so that a count of 2^{32} can be achieved as with the 80C186 internal timers.

5.0 SYSTEM DESIGN EXAMPLE: AN 80C186 DESIGN

A general purpose controller and/or data acquisition system was built to demonstrate 86-based ISW. The 80C186 CPU drives the system, which contains 16 Kbytes of EPROM (two 27C64's), 64 Kbytes of flash memory (two 28F256A's), 64 Kbytes of SRAM (two 32K x 8's) three 8-bit ports (82C55A), one serial port (82510), and a 5V to 12.0V DC/DC converter. Three 74HC573's demultiplex the address/data bus and latch the byte high enable line (BHE#) and the status lines (if needed). Two data transceivers (74HC245) simulate the worst case data path for a system requiring added drive capability. If the transceivers are not needed they can be replaced with wired headers. See Appendix F for detailed schematics parts list, and changes for the 28F512 or 28F010.

The 80C186 reset (output) drives the reset input on the 82510, 82C55A, and the OE# inputs on the address latches and data transceivers. The reset line goes inactive 5 clock cycles before the first code fetch. Also, the CPU's write signal is split into byte-write-high and byte-write-low to allow for byte or word writes.

The 80C186 has on-chip memory and peripheral chip selects. Two of the memory chip selects are dedicated. One is the Upper Chip Select (UCS#, dedicated for the boot area) and the second is the Lower Chip Select (LCS#, for the interrupt vector table area). See the memory map in Figure 10.

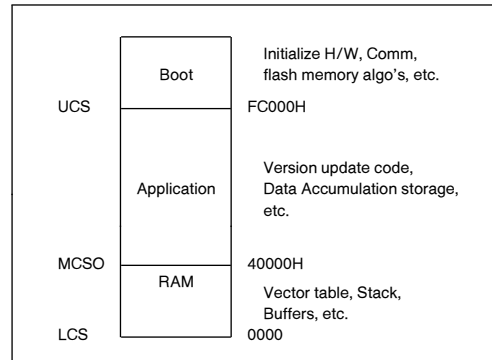


Figure 10. 80C186 Memory Map

The permanent code was placed in an EPROM in the UCS memory segment; this code includes routines for hardware initialization, communications, data uploading and downloading, erasure and programming algorithms, I/O drivers, ASCII to binary conversion tables, etc. This would be useful for systems reconfigured for different communication protocols as the last step prior to shipment.

Code and constants that might change are placed in the 64 Kbytes of flash memory. Application examples include operating systems, code for rapidly advancing biomedical technologies such as blood test software, engine-control code and parameters, character fonts for printers, postage rates, etc. The RAM is used for the interrupt table, stack, variable data storage, and buffers.

The three 8-bit ports on the 82C55A peripheral controller can be used for control and/or data acquisition. It powers-up with all port pins high. Similarly, all port pins go high after warm resets as well. Because the pins are high after a power-up/reset, an open collector inverter was used to control the MOSPOWER switch which in turn controls V_{pp} . You must drive the FET switch to one rail or the other to guarantee its low on-resistance. V_{pp} is turned off during power-up or reset as a hardware write protection solution. The DC/DC converter supplies V_{pp} .

The 82510 is a flexible single channel CHMOS UART offering high integration. The device off-loads the system and CPU of many tasks associated with asynchronous serial communications.

The part can be used as a basic serial port for the host serial link, or can be configured to support high speed modem applications. For more information on the 82510 see the 82510 data sheet and AP-401 “Designing with the 82510 Asynchronous Serial Controller”.

Software was written to download code and data parameters (code updates) from a PC to the demo board through the PC’s COM1 port (serial port). The system also can upload data (remote data acquisition) to the PC via the same link.

Once the download code and data has been programmed it can not be lost, even if power should fail. This is because Intel’s ETOX II flash memory technology is based on EPROM technology and does not need power to retain data.

The end result: rugged, solid state, low power nonvolatile storage.

6.0 SUMMARY

Intel’s flash memories offer designers cost-effective alternatives for remote version updates or for reliable data accumulation in the field or factory. Designers will also benefit from time savings in any kind of code development—no 15 minute waits for U.V. EPROM erasure.

This application note covers the basics of in-system writing to flash memories and can be used as a check list for systems other than the 80C186 design shown. The basic concepts remain the same: a CPU controls the reprogramming operations; a 12V supply must be applied to the flash memory for erasure and programming; and a communications link connects the host to the remote system and supplies the code to be programmed.



**APPENDIX A
ON-BOARD PROGRAMMING DESIGN
CONSIDERATIONS**

INTRODUCTION

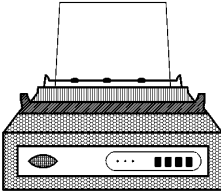
On-board programming¹ (OBP) with Intel's flash memory provides designers with cost reduction capabilities for alterable code storage designs. When used in conjunction with on-board programming, flash memory presents opportunities for savings in two areas: greater testability in the factory, which translates to improved outgoing quality and reduced return rate; and quicker, more reliable field updates, which translates to decreased product support cost.

This appendix:

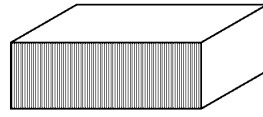
- outlines the design considerations associated with on-board programming, and the improvements afforded by Intel's flash memory;
- offers guidelines for converting current 64K EPROM OBP designs;
- designs an 8-bit system for on-board programming;
- suggests some 16-bit flash design considerations; and offers information on OBP equipment and vendors.

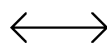
1. With on-board programming, non-volatile memory is programmed while socketed or soldered on the application board, rather than before hand as a discrete component. This programming method is also called in-module or in-circuit programming, and has been practiced by some major corporations since 1981. See sidebar on following pages for more information on U.V. EPROM OBP usage.

**HOST APPLICATION
(Printer Shown Here)**



BOARD-PROGRAMMER





292046-29

On-Board Programming Manufacturing Example—A printer is customized via OBP for international markets: 1. printer assembly completed, diagnostics code programmed and tested, and unit stored in inventory; 2. order arrives for printer with foreign language font; 3. diagnostics code flash-erased, and desired font programmed; 4. printer ships to customer.

INTEL'S FLASH MEMORY—DESIGNED TO MEET YOUR OBP NEEDS

Intel's flash memory simplifies OBP code updates by offering designers the command register architecture. As described in section 2.2, this architecture offers the full reliability of EPROM off-board programming without the hassles of elevating V_{CC}.

5 Volt V_{CC} Erasure and Programming Verification

Unlike EPROM OBP, flash memory enables V_{CC} to remain at 5.0V throughout all operations. Internal circuitry derives the erasure and programming verification levels from the voltage on V_{pp} rather than from V_{CC}. These verify modes enable use of a single V_{CC} bus for the entire board, as opposed to the two buses needed for U.V. EPROM OBP. (See sidebar entitled EPROM OBP).

EPROM OBP

EPROM OBP has been a proven manufacturing technique since 1981. Ingenuity and clever circuit design have enabled manufacturers to overcome the hurdles associated with OBP and enjoy the benefits.

In many cases, Intel's flash memory simplifies today's solutions and offers new capabilities to advance the state of OBP technology. The following paragraphs outline the hurdles and a few of the solutions in use today.

EPROMs require program verification at an elevated V_{CC} to insure long-term data retention. PROM programmers easily accommodate this requirement, and it is generally invisible to the end-user.

REPLACING CURRENT EPROM OBP DESIGNS WITH FLASH MEMORY

Hardware Considerations

A slight hardware modification is required to adapt most of the current EPROM OBP designs for use with Intel's flash memory. Simply convert the EPROM memory sites from 28 to 32 pins. All other board-design criteria used for EPROM OBP apply to flash memory as well. (For discussions of these criteria see section entitled New OBP Designs).

Standard EPROM OBP requires the board designer to bus PGM# to the edge connector. With flash memories' command register architecture, this same trace enables electrical erasure and programming, only now the line is called Write Enable (WE#). The timing for WE# is similar to that of read accesses, although that is handled via software changes.

Another potential hardware change is on the board programmer side of the design—the V_{pp} supply. Many EPROMs program with 12.5-13.0V V_{pp} supplies. Intel's ETOX II flash memory requires 11.4-12.6V V_{pp}. This change should not be an issue since the V_{pp} supply on many board programmers is programmable.

Mixed memory systems containing both conventional U.V. EPROM and flash memories require special consideration. This type of memory design requires separation of the Chip Enable (CE#) control lines between the EPROM and flash devices to allow for independent reprogramming control and access. The PGM# and

WE# lines can be common if the board programmer can give the appropriate timings to either type of device.

Software Considerations

Manufacturers who program EPROMs on-board today will need new board-programmer software to take advantage of flash memory's feature set, specifically software for the Quick-Erase and Quick-Pulse Programming algorithms.

Benefits of Converting Your EPROM OBP Design to Flash

The most pressing reason to convert from a standard EPROM to flash memory is the total cost savings. To appreciate this, you must consider your way of doing business at the board and system levels—from the factory to installation and repair in the field. In the factory, boards can be tested with a diagnostics program in the flash memory and then erased and reconfigured for shipment in the same step. Improved testing will decrease the probability of field failures and costly customer returns. Simplified test and rework methods will decrease your inventory holding costs. Also, if in the process of converting to flash memory you include the ability to OBP via a cable-connector, service calls for code updates will be quicker, more reliable, and cost less money. Your serviceman would simply connect the programming equipment to the system without dismantling it to remove the EPROMs. (See section entitled The System/Board-Programmer H/W Connection for details.)

EPROM OBP (cont'd)

With OBP, the EPROM board-programmer handles the elevated-V_{CC} requirement easily as well. However, when V_{CC} is greater than 5V, logic devices populating the same board may draw excessive current and not operate predictably.

One solution to this issue involves running separate V_{CC} traces to the board's edge connector—one for EPROM programming, and one for powering up the rest of the board.

A second consideration when designing for EPROM OBP has been accessing manufacturer and device codes.

The identifier mode requires forcing A9 to 12V. This translates to adding extra isolation, which implies the increased costs of buffers and extra board space.

NEW OBP DESIGNS

Design Considerations

As with EPROM in-circuit programming, flash memory board programming requires the use of a board-programmer. Unlike U.V. erasure for standard EPROM OBP, electrical erasure enables flash memory OBP without removing the board from the system.

We will look at designing a board that is to remain powered-up in the system during erasure and reprogramming. The key concept is to design the board in such a way that the programmer can take control of the system during code updates. The implementation of such a design is straightforward, easy, and suited to automated production assembly.

Taking Control

The board-programmer needs to take control of the system's address bus, data bus, control lines, etc. to update the code without damaging the system. (See Figure 2. System to Board-Programmer Interface.) Taking control simply means isolating the rest of the system from these lines.

Various methods of isolating the memory from the system include using tristate buffers, latches, or even the capabilities designed into microprocessors (μ P) and microcontrollers (μ C). For example, Intel's 86-based μ P family has HLD/HLDA signals that were set-up for multiprocessor system designs where bus control is a major concern. The HLD signal, when acknowledged, tristates the address, data, and control lines. Although not designed for multiprocessor environments, Intel's MCS[®]-51 and MCS-96 microcontroller families have Reset capabilities to help simplify this same task.

One issue to be aware of when using a CPU's reset control function is that it may switch from the reset to active condition at a non-standard logic level. This only presents a problem if the address/data buffer takes longer to activate than the CPU, and the CPU attempts to fetch code from a memory device isolated from it.

One approach to insure successful programming takeover (i.e. without bus contention) is to have the board-programmer's lines in a high impedance state during connection to the system. Once connection to the system has been secured, the serviceman could hit a button on the board-programmer to start the system takeover procedure. Then when total control has been established, the programmer would commence with erasure and reprogramming.

Aside from the flash device's isolation from the system, various CPU control lines (MEMRD#, WE#, PSEN#, etc.) may need isolation as well. If active during Reset, these lines may put the CPU into an unspecified state. When designing a board for OBP, check the μ C/ μ P data sheets carefully for any special reset conditions.

Printed Circuit Board Guidelines for V_{CC} and V_{pp}

Programming conventional EPROM and flash memories takes 30 mA of current on V_{CC} and V_{pp}, due to the nature of hot-electron injection. Most of the charge transfers to the memory cell's floating gate in a short current spike during the first pulse. You should design both the V_{CC} and V_{pp} traces with A.C. current spikes in mind. Wherever possible, limit the inductance by widening the two traces. Bypass capacitors (0.1 μ F) should be placed as close as possible to the memory device's V_{CC} and GND pins, as well as the devices V_{pp} and GND pins. The capacitor on V_{cc} decreases the power supply droop. The capacitor on V_{pp} supplies added charge, and filters and protects the memory from high frequency over-voltage spikes².

2. For a complete discussion of electrical noise, grounds, power supply distribution and decoupling see Ap-74—High Speed Memory System Design Using the 2147H, and AP-125—Designing Microcontroller Systems for Electrically Noisy Environments.

EPROM OBP (cont'd)

Some users of OBP get around this issue by programming all EPROMs with a common algorithm. However, this practice compromises the device's reliability, and should not be done.

A better solution than ignoring the identifier is to choose a qualified EPROM vendor and program with its algorithm only.

One subtle concern with EPROM OBP that designers often overlook is U.V. board erasure.

→ U.V. EPROM board erasure requires removal of the board from its host system. This incurs the hidden costs of labor, lower yields due to handling, and the reliability risks of dismantling a system. Flash memory decreases these costs by enabling a greater degree of factory automation, and increases the flexibility afforded by OBP.

The System/Board-Programmer Hardware Connection

In most U.V. EPROM OBP applications, designers use the board's edge-connector as the programmer interface. This approach is the lowest cost solution for standard EPROM technology because U.V. erasable devices require system disassembly for erasure anyway. With flash memory, you can eliminate the system dismantling and capitalize on the erase feature by adding a cable connector to the board for reprogramming purposes. The connector should extend from the board through the system's chassis, and should be easy to reach by a serviceman.

Various types of cables exist on the market that could be used to connect programming equipment to the system. The key design consideration when choosing the type of cable is elimination of all transient noise that would interfere with the programming or erasure process.

Three types of noise interference and methods to diminish the noise are as follows:

1. line to line cross-talk (due to board-programmer's drivers that drive sharp step functions on adjacent address lines); solved with either ribbon cables, having alternate lines grounded, or with braided twisted-pairs that have a ground line for each active signal;
2. programmer line-driver-to-board impedance mismatches leading to transmission line effects of signal reflection, and interference; solved by limiting cable length, decreasing programmer switching speed (or allowing longer settling time between address switches) or by using matched line drivers on the programmer and high impedance buffers on the board end, or by using series termination resistors on the driving end of the cable (i.e.—board-programmer end, with the exception of the bi-directional data bus which needs series resistors at both ends);
3. rf pick-up in electrically noisy environments; use either shielded cable such as coax, ribbon cable with solid copper ground plane, or a new type that has recently become available called Flex cable.

Braided twisted-pair cables when kept under three feet in length generally reduce cross-talk to acceptable levels. This type of cable offers the most cost-effective solution which works well in most applications. Depending on the environment, the programmer and your design, you may need a combination of solutions, such as braided twisted-pairs with series termination.

3. Note that the flow-through latch on the data bus is not needed with the 80C31, but is drawn as an example for CPU's that can not tristate their data bus.

4. The isolation buffer is required on PSEN# in this design because the 80C31 goes into unspecified states when the Reset and PSEN# lines are active simultaneously. To avoid any possible problems, buffer PSEN#.

5. MEMWR = > bus isolation control of PSEN# and the data bus.

At first all of these alternatives may seem expensive or superfluous, but keep in mind that the cost of a single cable and programmer gets amortized over the total number of systems programmed.

AN 8-BIT BUS DESIGN EXAMPLE

An example of an in-circuit reprogrammable controller board is an 80C31, two 28F256A's and some glue chips. (See Figure 3. for a system block diagram. See Appendix A. for a detailed system schematic.)³ The important issues for erasure and reprogramming are as follows:

1. the board-programmer must have uncontested access and control of the flash memory array; and
2. the microcontroller must be reset (un-active) during the erasure and programming cycles.

SYSTEM DESIGN

Bus Control Circuitry

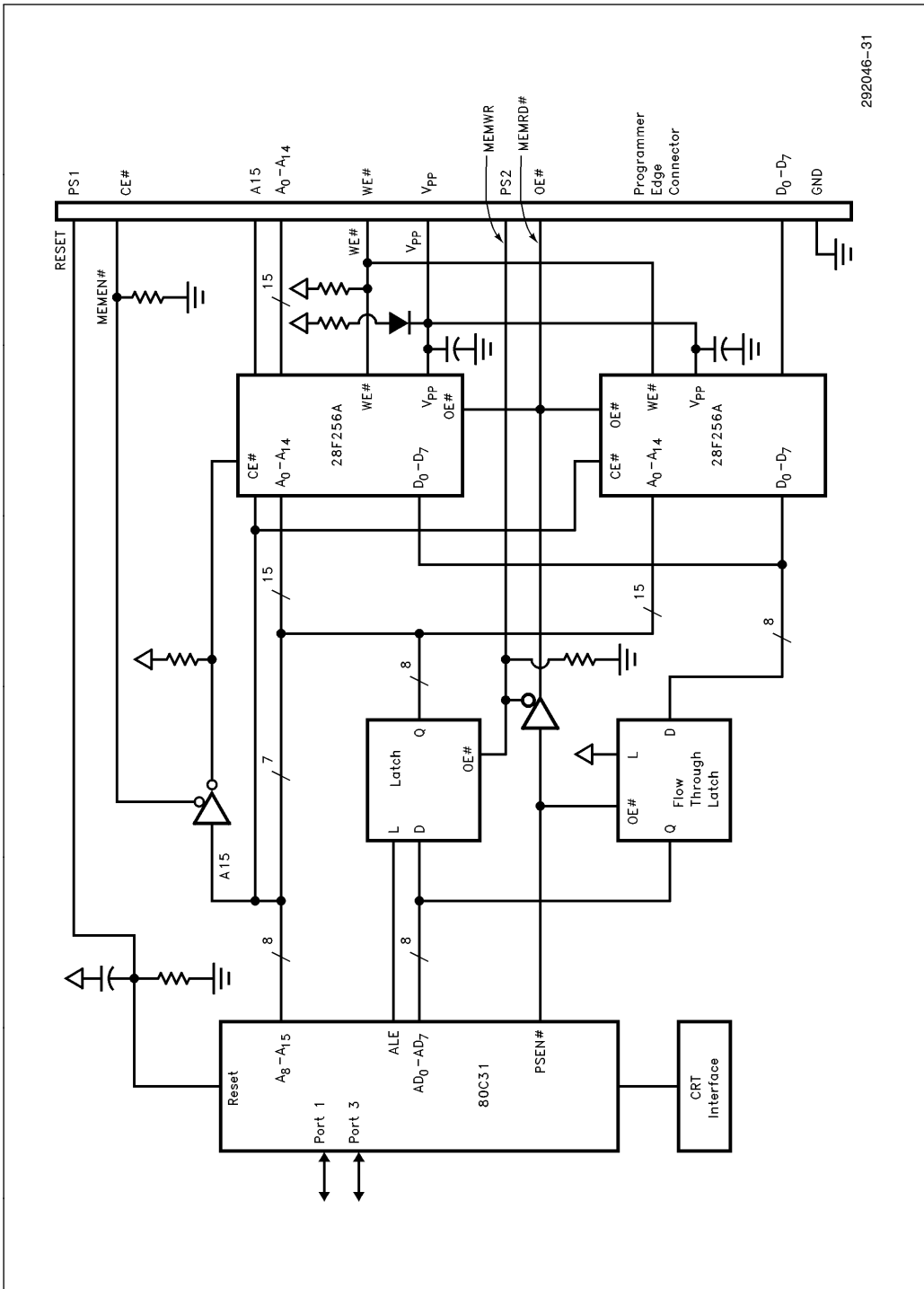
The 80C31 has an active-high reset pin, which tristates the address and data buses. Route this line (RESET) to the programming connector. Tie the OE# pins on the low-order address latch (74HCT573), and the PSEN# buffer-enable (74HCT125)⁴ together, and route that line MEMWR⁵ to another pin on the programmer-interface connector.

During normal system operations when the μ C reads program code from the 28F256 devices, the pull-down on MEMWR keeps the address latches and PSEN# buffer active. During flash memory OBP, the board-programmer drives MEMWR active-high, which disables these outputs, and isolates the address bus and PSEN# from the programming signals.

The board-programmer must independently control the RESET and MEMWR traces because they disable at different V_{IL} values (2.5V for RESET vs 0.8V for MEMWR). If controlled by the same 5V supply, on power-up or after a reset condition the μ C would try to execute code while still isolated from its code source—specifically before the address latches and PSEN# buffer activate.

Address Decode Circuitry

This design shows two 28F256A flash memories. Systems with more than one memory device typically decode the CPU's high-order address to select a particular device.



292046-31

Figure 3. System Block Diagram

This is accomplished as illustrated. When A15 is low, the lower 32K bytes are selected. The output of the inverter drives the other 28F256A's chip enable. This type of memory architecture promotes power savings by disabling all memories but the one being addressed.

To accomplish this two-line memory control architecture, route the inverter's input A15 to the 80C31 and to the programmer interface connector.⁸ The board-programmer controls the inverter's output enable with MEMEN#.⁹ The MEMEN# line performs the function normally performed by CE# in component programming. When driven to a logic "1" level MEMEN# pulls the inverter's output high. This deselects all memory devices controlled by that I.C. During normal read and standby operations, the pull-down on MEMEN# keeps the decoder enabled.

Erase and Programming Control Circuitry

In this design, V_{PP} and WE# are active only during reprogramming. At other times, the two inputs would be inactive. Simply tie the WE# line to V_{CC} through a pull-up resistor. The pull-up limits the current to the board programmer during reprogramming. (Recall that WE# is active low.) Flash memories allow V_{PP} to be at 12V, V_{CC} or ground for read operations. This design ties V_{PP} to V_{CC} through a diode and resistor to allow for EPROM OBP compatibility. If this option is not required, simply tie V_{PP} to ground through a current-limiting pull-down resistor.

Returning Control to the Host System

The board-programmer should return system-control to the host processor in an organized manner. First it should lower V_{PP} from 12V to 5V, or ground. Then the board programmer should place its address and data

buses into a high impedance state. Next PS2, which controls MEMWR should be tristated thus disabling the PSEN#/Address latch isolation. Finally the board-programmer should switch PS1, which drives the RESET line to reactivate the μ C. This sequence guarantees that the μ C will begin operation at a known program code location.

16-BIT BUS DESIGN CONSIDERATIONS

An example of an On-Board programmable 16-bit system board would be an 80C186 microprocessor, two 28F010 flash memories, RAM, and some glue chips. The basic hardware design considerations would be the same as those in the previously discussed 8-bit bus example.

There are a few issues with 16-bit designs that do not arise in 8-bit designs. For the programmer to take control of the system, it must tristate and reset the μ P as well as tristate the bus buffers and latches. The HOLD and RESET lines of Intel's 86-based family of microprocessors have been designed with bus isolation in mind for use in multiprocessor systems.

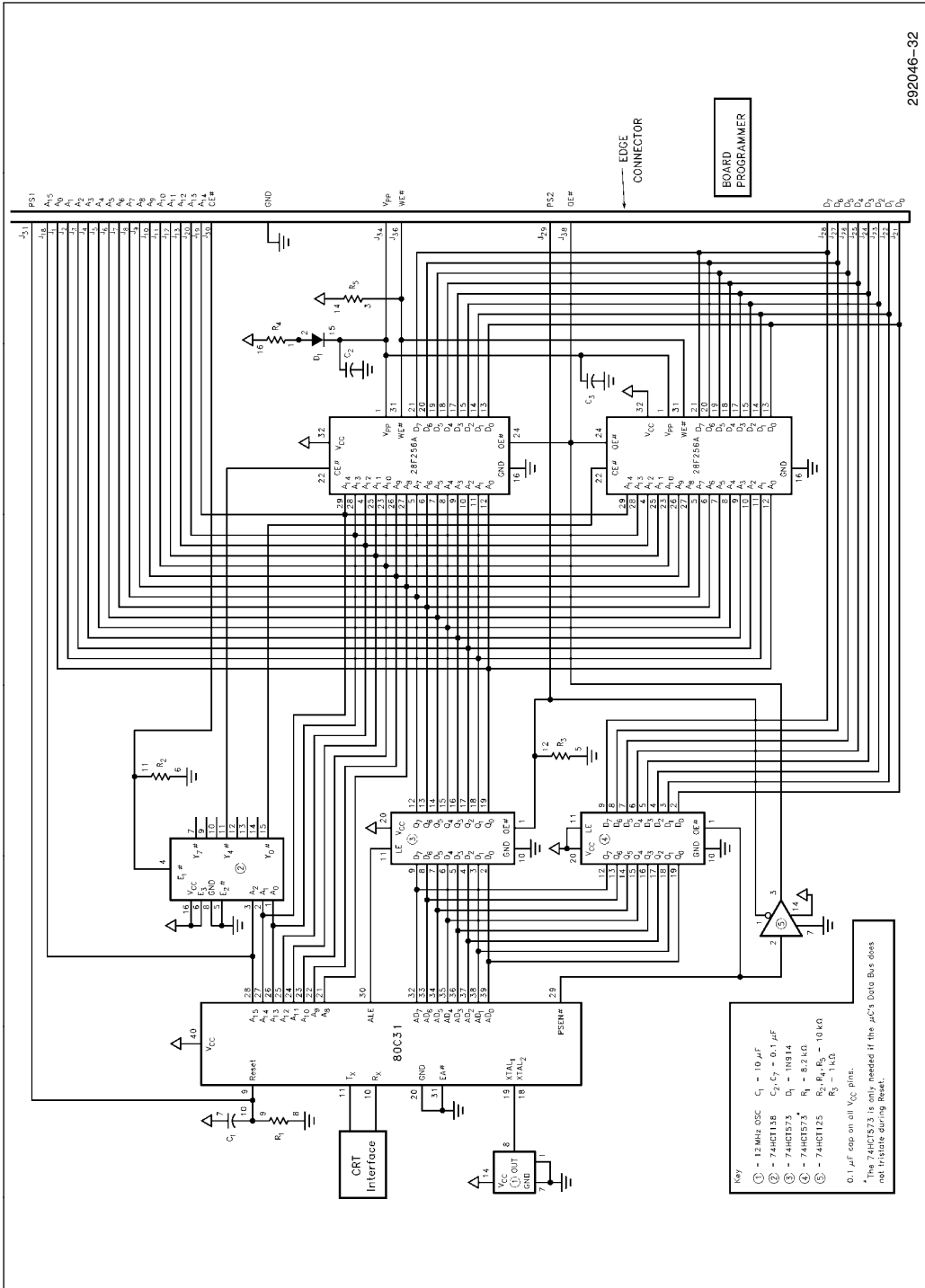
The designer has two options for erasing and programming the high and low bytes of the flash memory array independently.

1) The designer can route two WE# lines to the programmer connector—BYTE HIGH WE# and BYTE LOW WE#.

2) The reprogramming software can follow the masking procedure shown in section 4.4. This method allows a common WE# line for the high and low bytes.

8. Note the lack of isolation buffers between the 80C31's high order addresses (Port 2) and the board-programmer interface, compared to the latch separating the low order addresses (Port 0) and the interface. In this design example, we make use of the 80C31's ability to tristate these ports, so no isolation is needed for any of the addresses. The latch on Port 0 is for the time-multiplexed address/data architecture of this microcontroller, and not specifically for isolation.

9. MEMEN = memory enable, active low.



292046-32

Figure 4. Detailed 8-Bit Bus Design Schematic



OBP EQUIPMENT AND VENDORS

If you are considering OBP for your next design, and have not used on-board programming before, you will need to choose a board-programmer vendor. Various suppliers offer OBP systems; therefore, it is well worth it to send out requests for programming support bids. If your production volume justifies the purchase of more than one board-programmer, you may want to negotiate a non-recurring engineering charge for development cost, followed by variable costs for additional units.

Most vendors offer a variety of basic systems, designed to easily adapt to your needs. Systems can be purchased that program either single boards serially, or a number of boards in parallel. Light-weight OBP equipment designed for field reprogramming can also be obtained from some of the vendors.

Most companies will work directly with you at the beginning of your design phase to ensure OBP compatibility. If your design is beyond the definition stage, the programmer manufacturer will request a copy of your schematics or block diagrams under non-disclosure. The vendor has an OBP design specialist that will check the design for OBP compatibility. Any potential problems will be located and corrected at this early stage.

Every board's architecture is different (i.e., based on different central processing units (CPU), decoding schemes, buffering methodologies, interface connectors, and types and densities of memories). Vendors write custom software modules for each application. Also, the vendor or the board designer typically builds an interface jig to connect the board's edge connector to the programmer. This choice is often left as a decision for the designer.

Partial List* of Companies Selling Board-Programmers

Following are a few of the companies who offer on-board programming solutions today:

Data I/O Corp.
Digelec
Elan Digital Systems
Oliver Advanced Engineering, Inc.
Stag Microsystems, Inc.

*This list is intended for example only, and in no way represents all companies that support on-board programming. Intel Corporation assumes no responsibility for circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

SUMMARY

- On-board programming (OBP) has been around since 1981.
- Designing a board for OBP can be easily done by working with a board-programmer vendor's OBP-design-specialist during the initial design phase.
- In-circuit alterable code storage can be easily implemented by using flash memory and its features.
- Time and money savings can be realized in a number of ways by taking advantage of flash memory OBP:
 - <> Decreased board costs and improved reliability from elimination of EPROM sockets;
 - <> Decreased manufacturing costs from elimination of board eraser depreciation costs, recurring U.V. light bulb and energy expenses;
 - <> Decreased inventory expense from simplified test and rework methods (one-step diagnostics, erasure, and board configuration);
 - <> Decreased product costs based on decreased board-handling loss;
 - <> Improved board diagnostics and testability leading to higher quality and decreased customer returns; and
 - <> **Quicker, more reliable field code updates.**



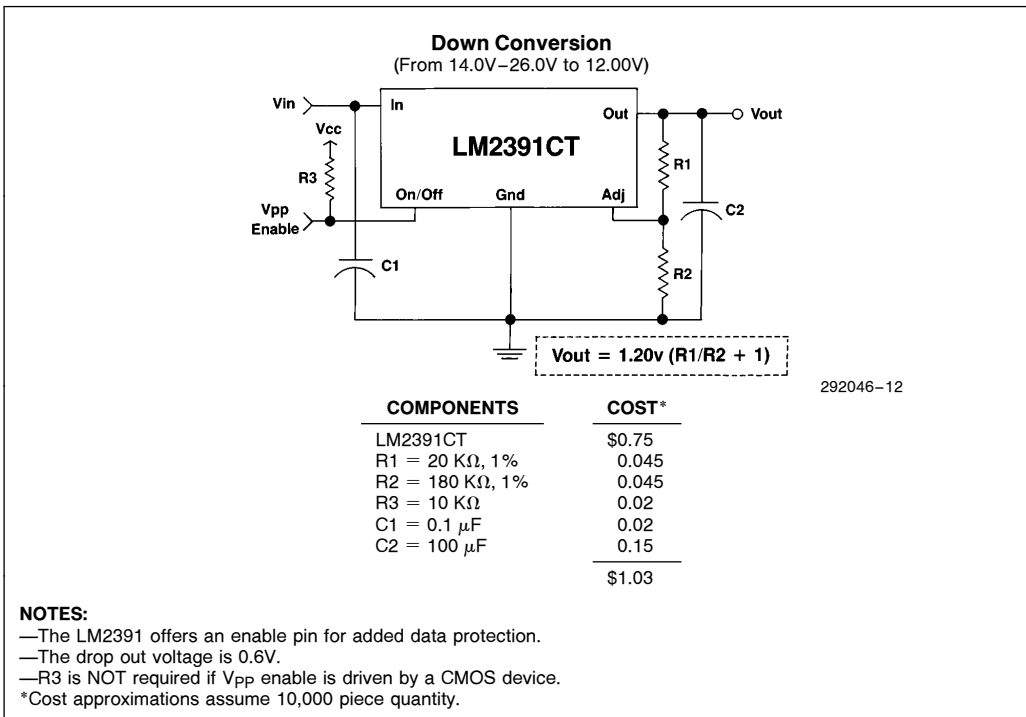
APPENDIX B

V_{PP} GENERATION CIRCUITS

- Circuit #1—Regulation from a higher voltage
- Circuit #2—Regulation from a higher voltage
- Circuit #3—Regulation from a higher voltage
- Circuit #4—5V to 12V Boost
- Circuit #5—5V to 12V Boost
- Circuit #6—Monolithic DC/DC Convertor

For more detailed information on V_{PP} generation circuits, see AP-357 titled Power Supply Solutions for Flash Memory (Order Number 292092).

Circuit # 1



Circuit # 2

Down Conversion
(From 16.00V–26.00V to 12.00V)

Vout = 1.25v (R2/R1 + 1)

COMPONENTS	COST*
LM-317	0.40
R1 = 124Ω, 1%	0.045
R2 = 1070Ω, 1%	0.045
C1 = 0.1 μF	0.02
C2 = 100 μF	0.15
	<u>\$0.66</u>

292046-13

NOTES:
LM-317 requires a minimum $V_{IN}-V_{OUT} = 3.0V$
*Cost approximations assume 10,000 piece quantity.

Circuit # 3

Down Conversion
(From 15.0V–40.0V to 12.00V)

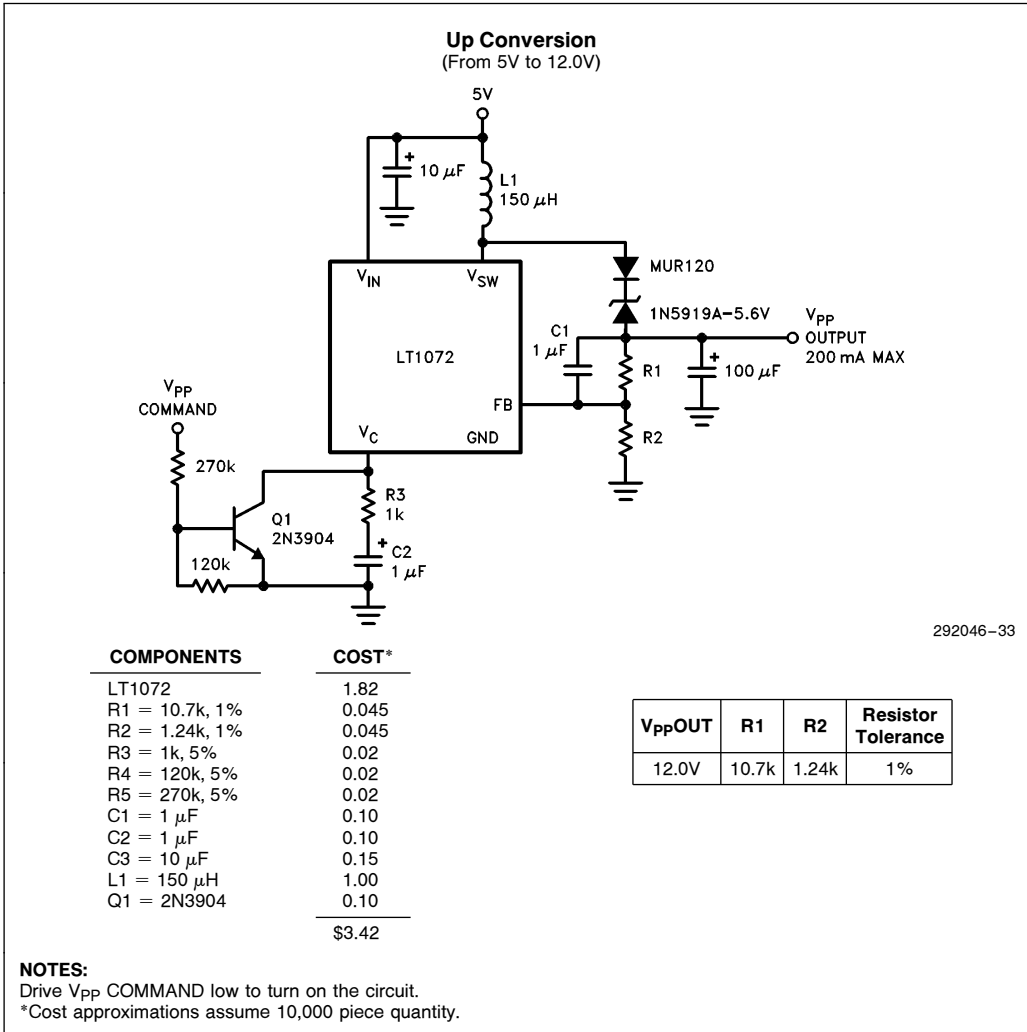
Vout = 1.25v (R1/R2 + 1)

COMPONENTS	COST*
LT-1085	2.50
R1 = 124Ω, 1%	0.045
R2 = 1070Ω, 1%	0.045
C1 = 10 μF	0.10
C2 = 10 μF	0.10
	<u>\$2.79</u>

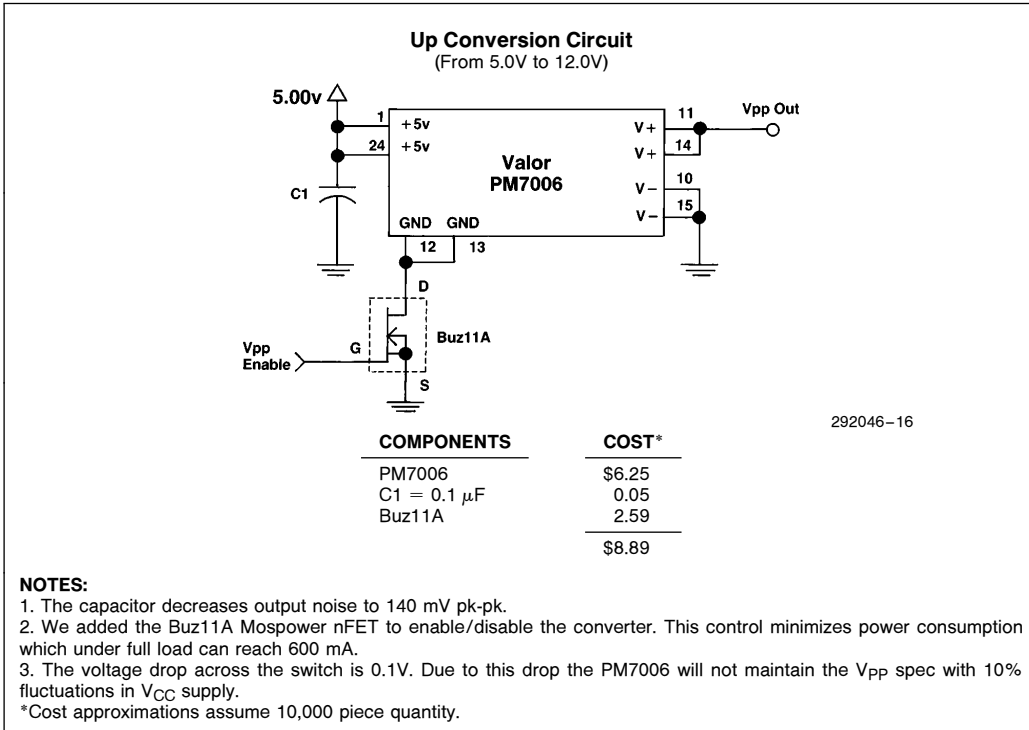
292046-14

NOTES:
LT-1085 requires a minimum $V_{IN}-V_{OUT} = 1.5V$
*Cost approximations assume 10,000 piece quantity.

Circuit # 4



Circuit # 5



APPENDIX C

LIST* OF DC-DC CONVERTER COMPANIES

AT&T MICROELECTRONICS†
3000 Skyline Drive
Mesquite, TX 75149
Tel: (800) 526-7819
Fax: (214) 284-2317

BURR-BROWN CORP.†
P.O. Box 11400
Tucson, AZ 85734
Tel: (800) 548-6132
Fax: (602) 741-3895

LINEAR TECHNOLOGY CORP.Δ
1630 McCarthy Blvd.
Milpitas, CA 95035
Tel: (408) 432-1900
Fax: (408) 434-0507

MAXIM INTEGRATED PRODUCTSΔ
120 San Gabriel Drive
Sunnyvale, CA 94086
Tel: (408) 737-7600
Fax: (408) 737-7194

MOTOROLA INC.Δ
2100 E. Elliot Rd.
Tempe, AZ 85284
Tel: (800) 845-6686

NATIONAL SEMICONDUCTOR CORP.Δ
Mt. Prospect, IL 60056
Tel: (800) 628-7364
Fax: (800) 888-5113

SHINDENGEN AMERICA, INC.†
2649 Townsgate Rd., Suite 200
Westlake Village, CA 91361
Tel: (800) 634-3654
Fax: (805) 373-3710

SILICONIX INC.Δ
2201 Laurelwood Rd.
Santa Clara, CA 95056
Tel: (800) 554-5565
Fax: (408) 727-5414

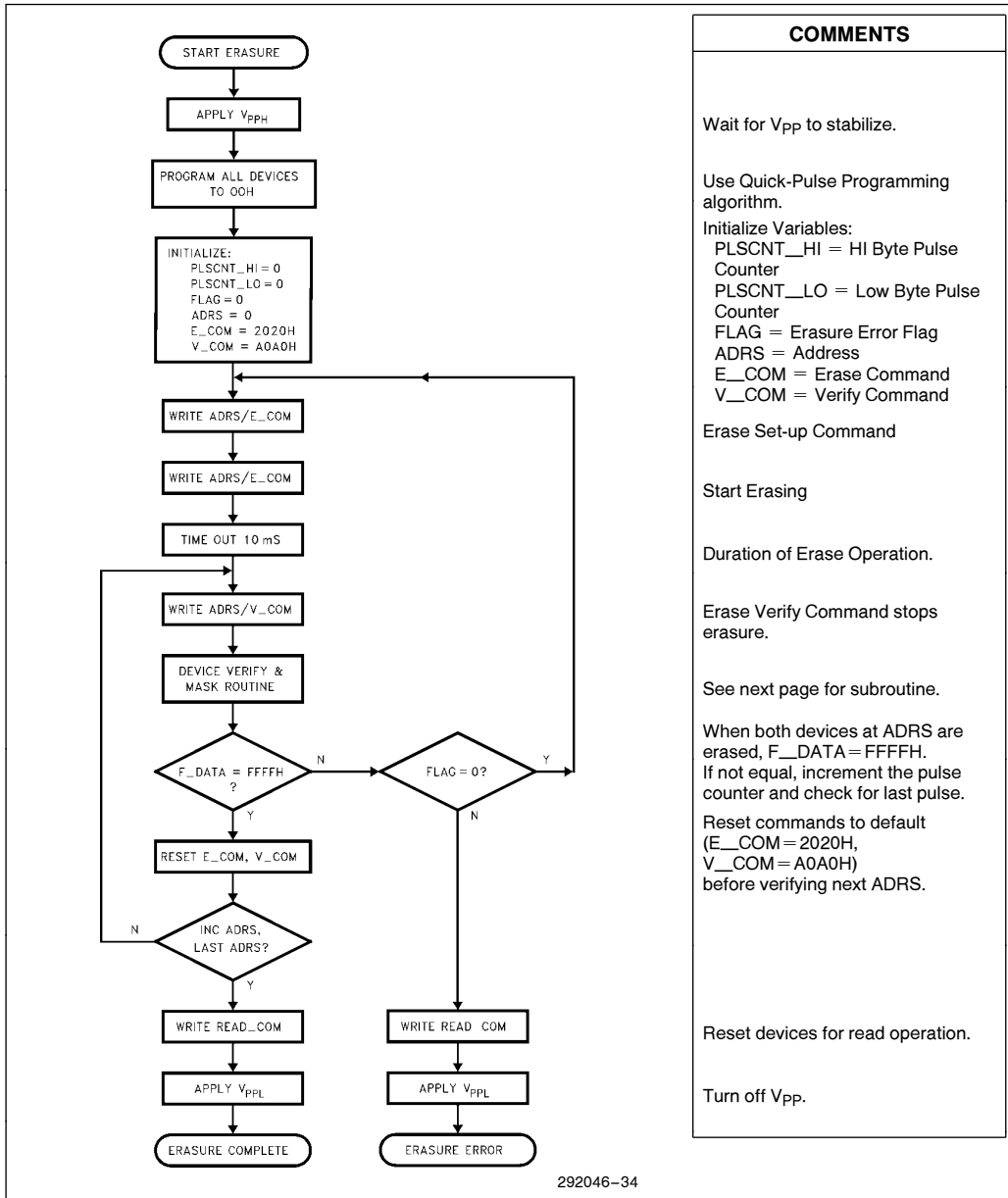
TOKO AMERICA, INC.†
1250 Feehanville Drive
Mount Prospect, IL 60056
Tel: (708) 297-0070
Fax: (708) 699-7864

VALOR ELECTRONICS†
6275 Nancy Ridge Dr.
San Diego, CA 92121
Tel: (619) 458-1471

*This list is intended for reference only, and in no way represents all companies that support power conversion products. Since this industry develops many new solutions each year, Intel recommends that the designer contacts the vendors for the latest products. Intel will continue to work with the industry to develop optimum solutions for power conversion. Intel Corporation assumes no responsibilities for circuitry other than circuitry embodied in Intel products. No other circuit patent licenses are implied.

†Monolithic Solutions
ΔDiscrete DC to DC Converter Solutions

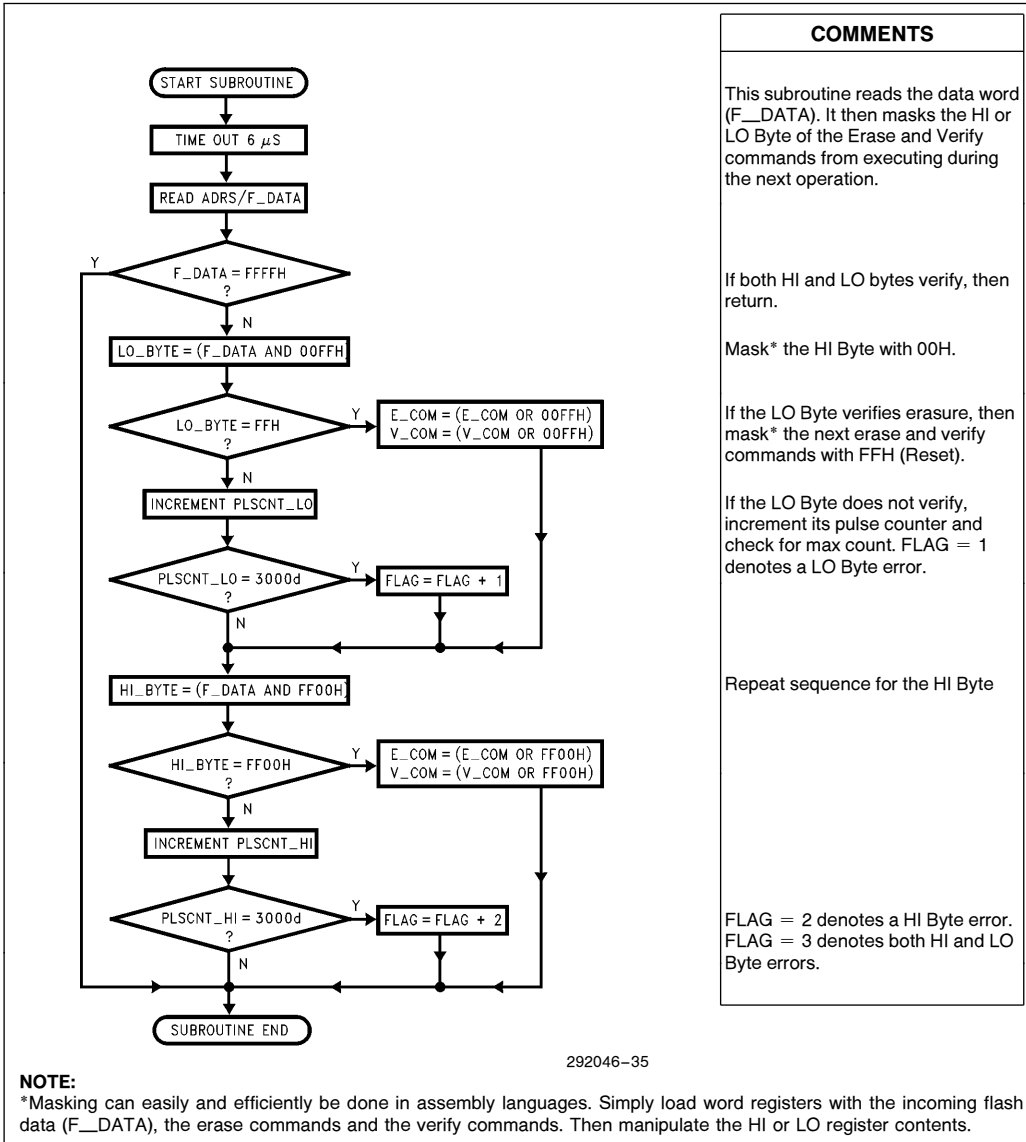
APPENDIX D PARALLEL ERASE FLOW CHART



COMMENTS
Wait for Vpp to stabilize.
Use Quick-Pulse Programming algorithm.
Initialize Variables: PLSCNT_HI = HI Byte Pulse Counter PLSCNT_LO = Low Byte Pulse Counter FLAG = Erasure Error Flag ADRS = Address E_COM = Erase Command V_COM = Verify Command
Erase Set-up Command
Start Erasing
Duration of Erase Operation.
Erase Verify Command stops erasure.
See next page for subroutine.
When both devices at ADRS are erased, F_DATA = FFFFH. If not equal, increment the pulse counter and check for last pulse.
Reset commands to default (E_COM = 2020H, V_COM = A0A0H) before verifying next ADRS.
Reset devices for read operation.
Turn off Vpp.

292046-34

Device Erase Verify and Mask Subroutine



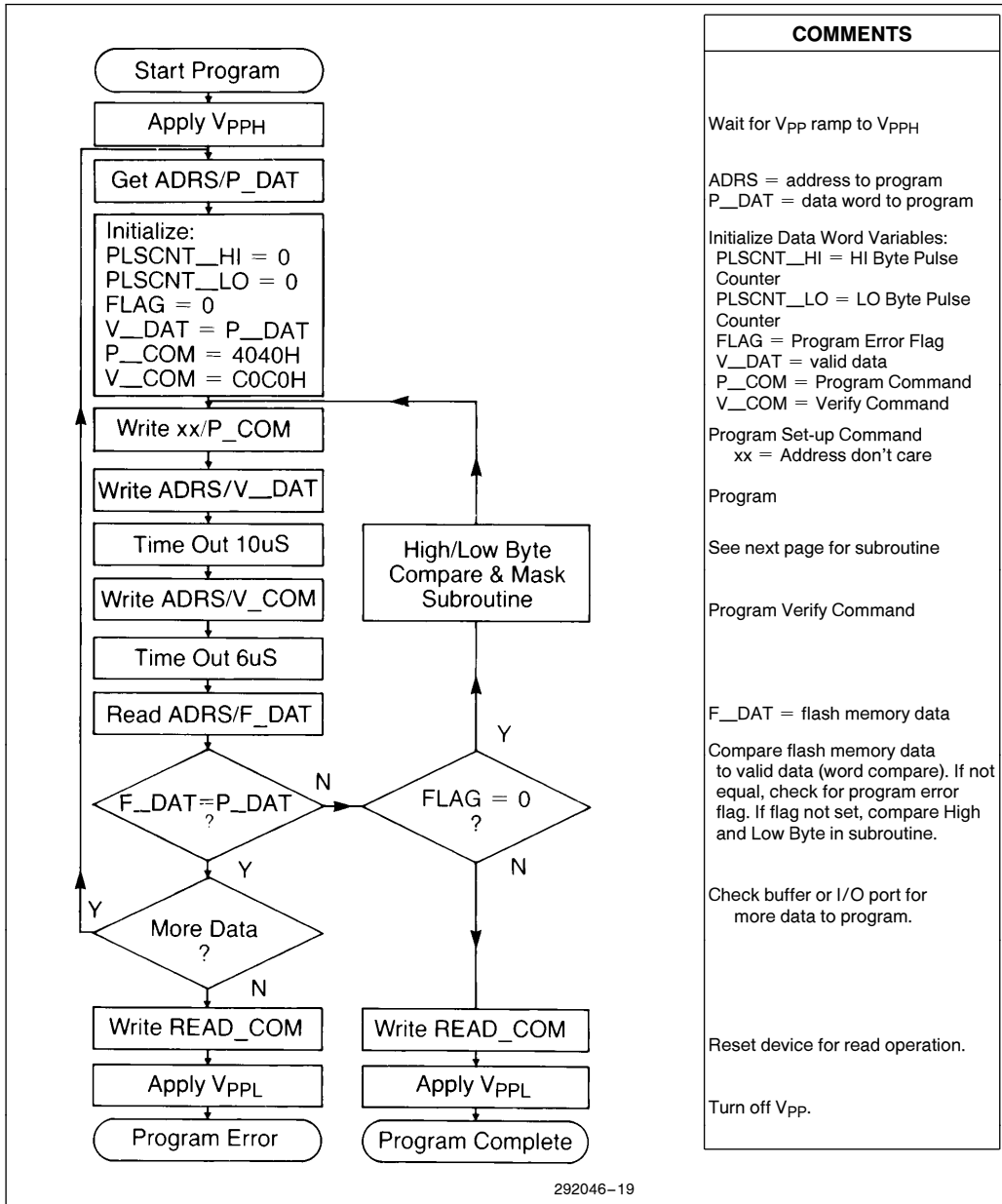
COMMENTS
This subroutine reads the data word (F_DATA). It then masks the HI or LO Byte of the Erase and Verify commands from executing during the next operation.
If both HI and LO bytes verify, then return.
Mask* the HI Byte with 00H.
If the LO Byte verifies erasure, then mask* the next erase and verify commands with FFH (Reset).
If the LO Byte does not verify, increment its pulse counter and check for max count. FLAG = 1 denotes a LO Byte error.
Repeat sequence for the HI Byte
FLAG = 2 denotes a HI Byte error. FLAG = 3 denotes both HI and LO Byte errors.

NOTE:

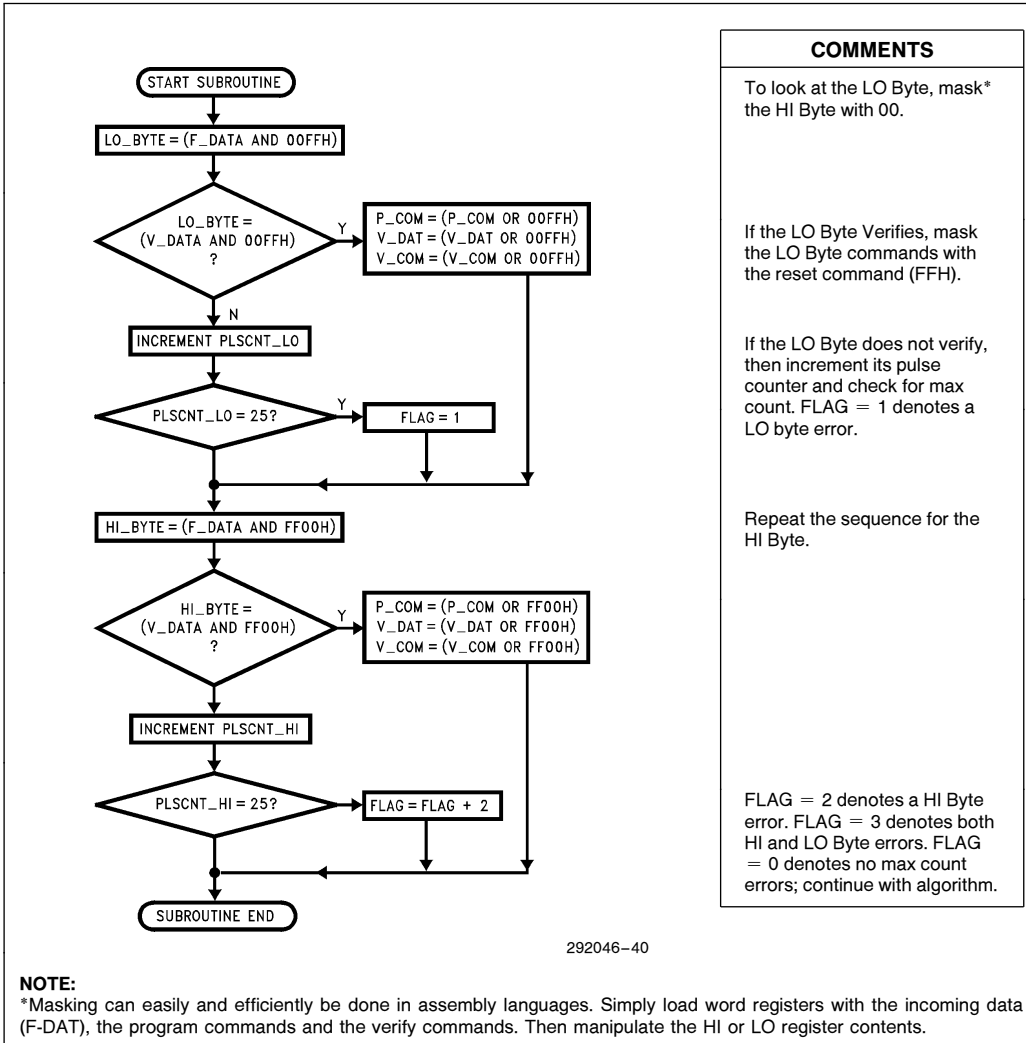
*Masking can easily and efficiently be done in assembly languages. Simply load word registers with the incoming flash data (F_DATA), the erase commands and the verify commands. Then manipulate the HI or LO register contents.

292046-35

APPENDIX E PARALLEL PROGRAMMING FLOW CHART



Program Verify and Mask Subroutine



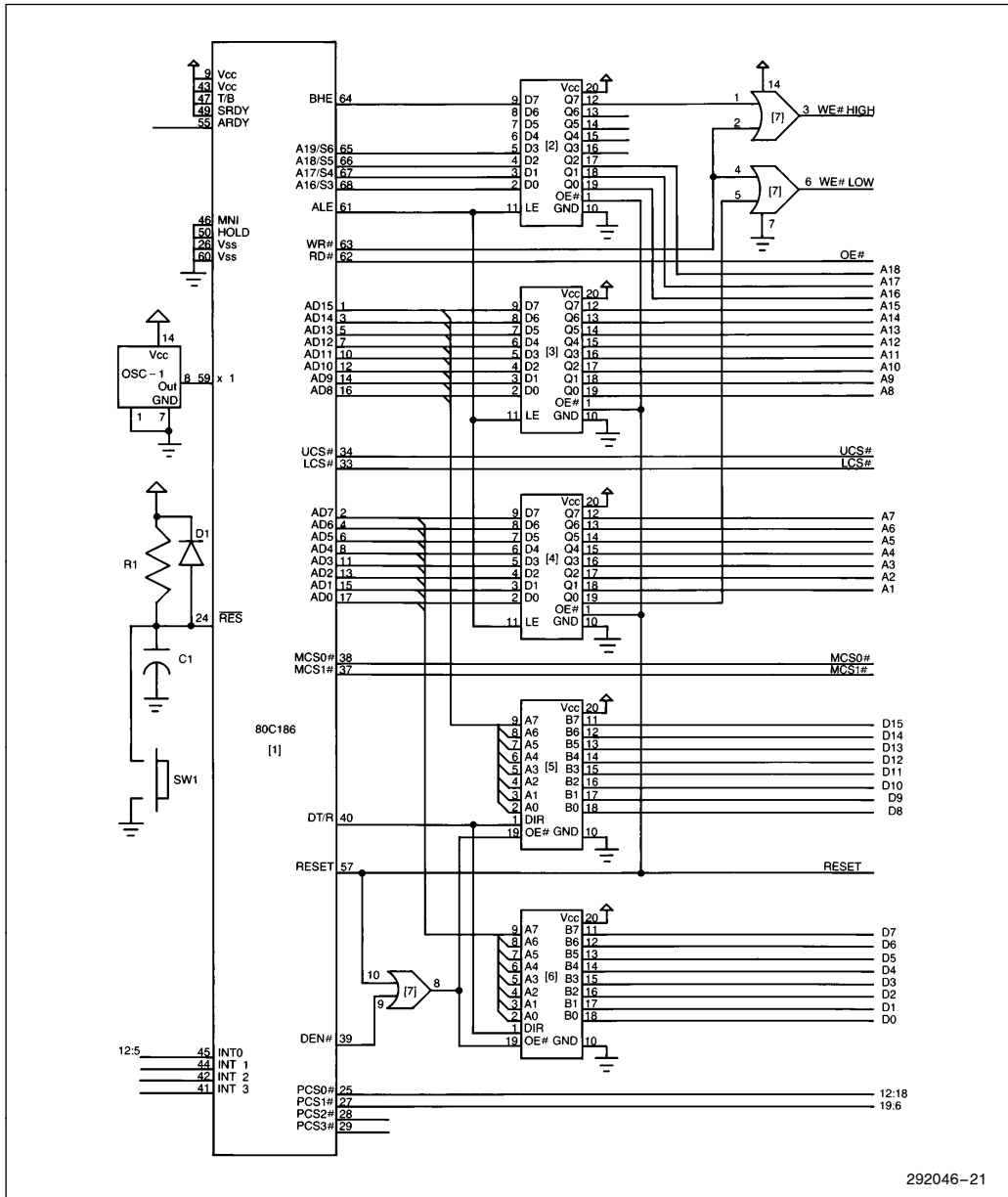
COMMENTS
To look at the LO Byte, mask* the HI Byte with 00.
If the LO Byte Verifies, mask the LO Byte commands with the reset command (FFH).
If the LO Byte does not verify, then increment its pulse counter and check for max count. FLAG = 1 denotes a LO byte error.
Repeat the sequence for the HI Byte.
FLAG = 2 denotes a HI Byte error. FLAG = 3 denotes both HI and LO Byte errors. FLAG = 0 denotes no max count errors; continue with algorithm.

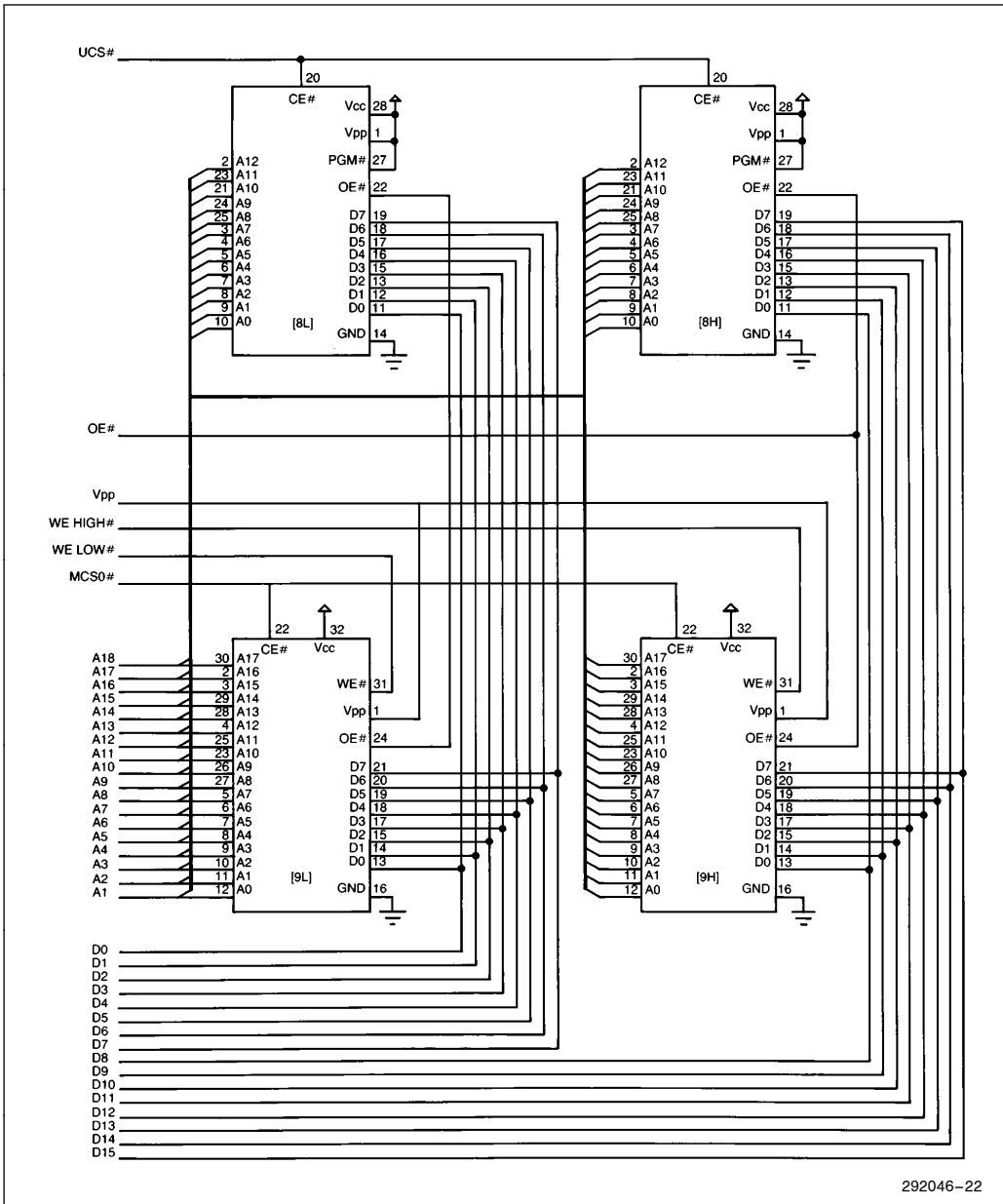
292046-40

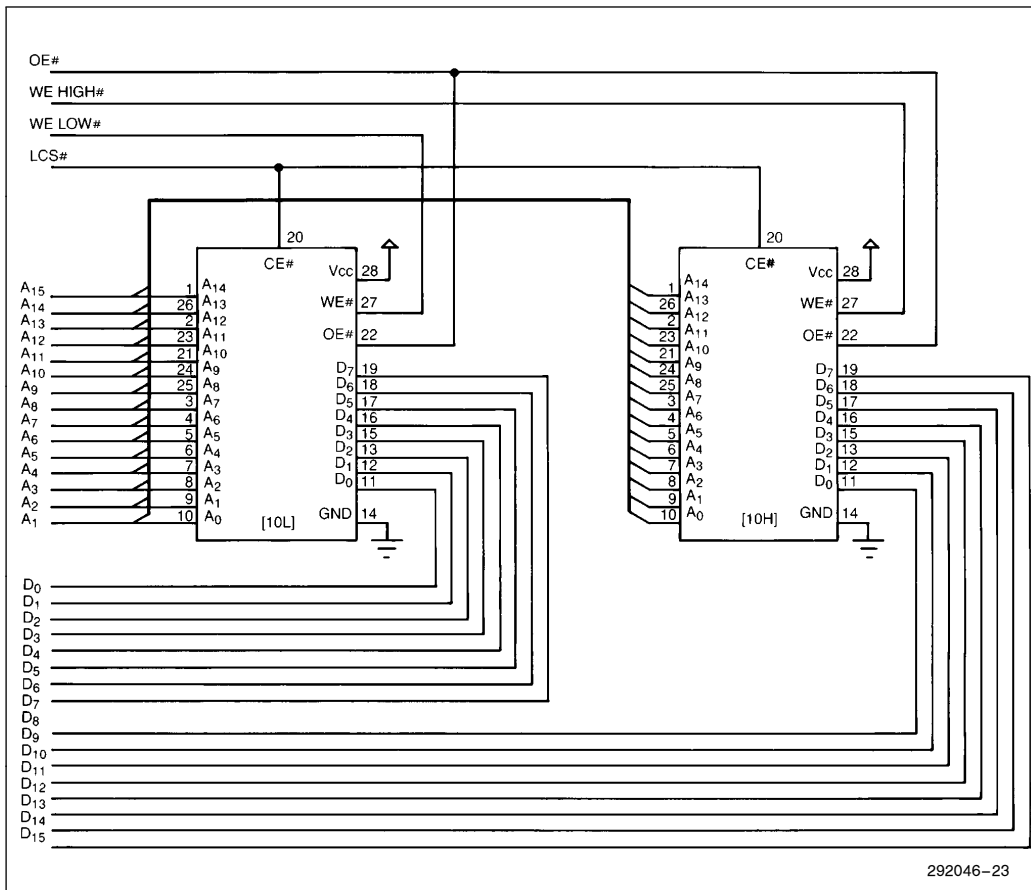
NOTE:

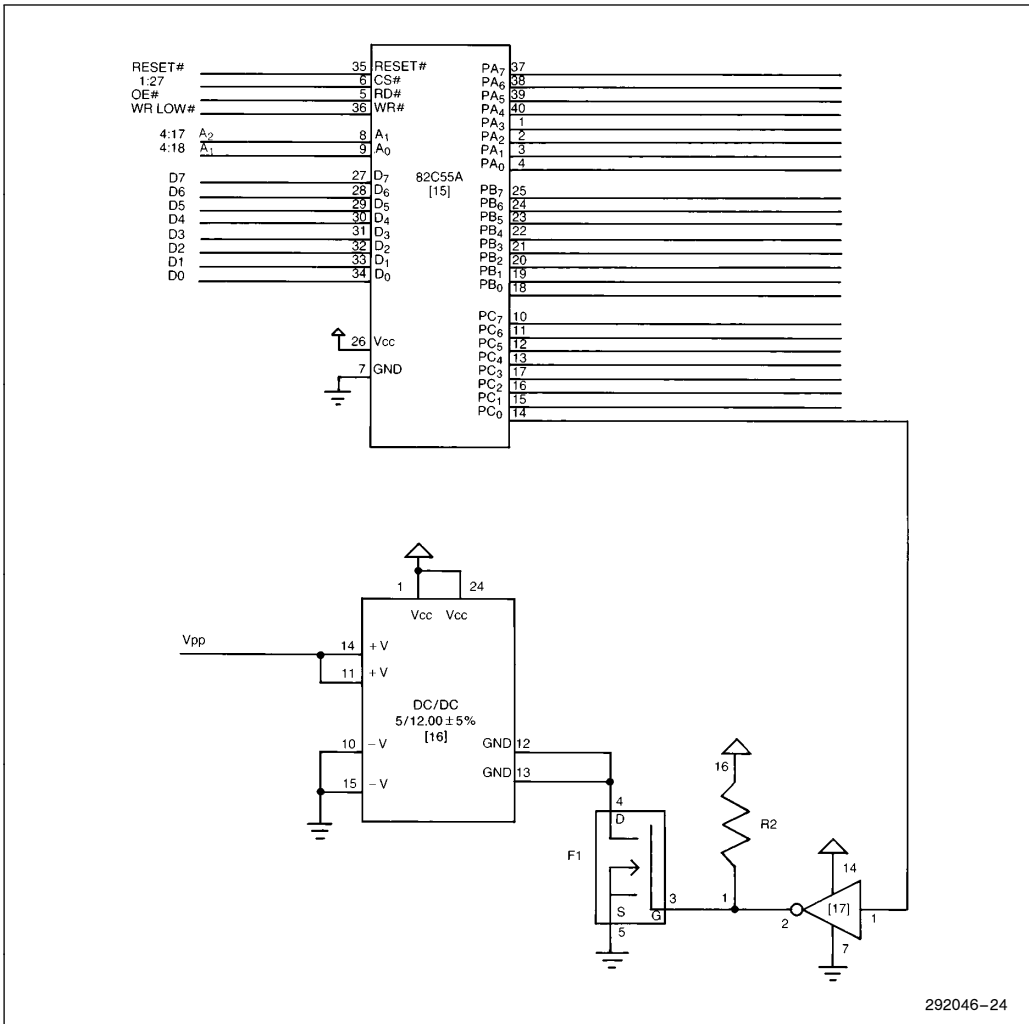
*Masking can easily and efficiently be done in assembly languages. Simply load word registers with the incoming data (F-DAT), the program commands and the verify commands. Then manipulate the HI or LO register contents.

APPENDIX F DETAILED SYSTEM SCHEMATICS

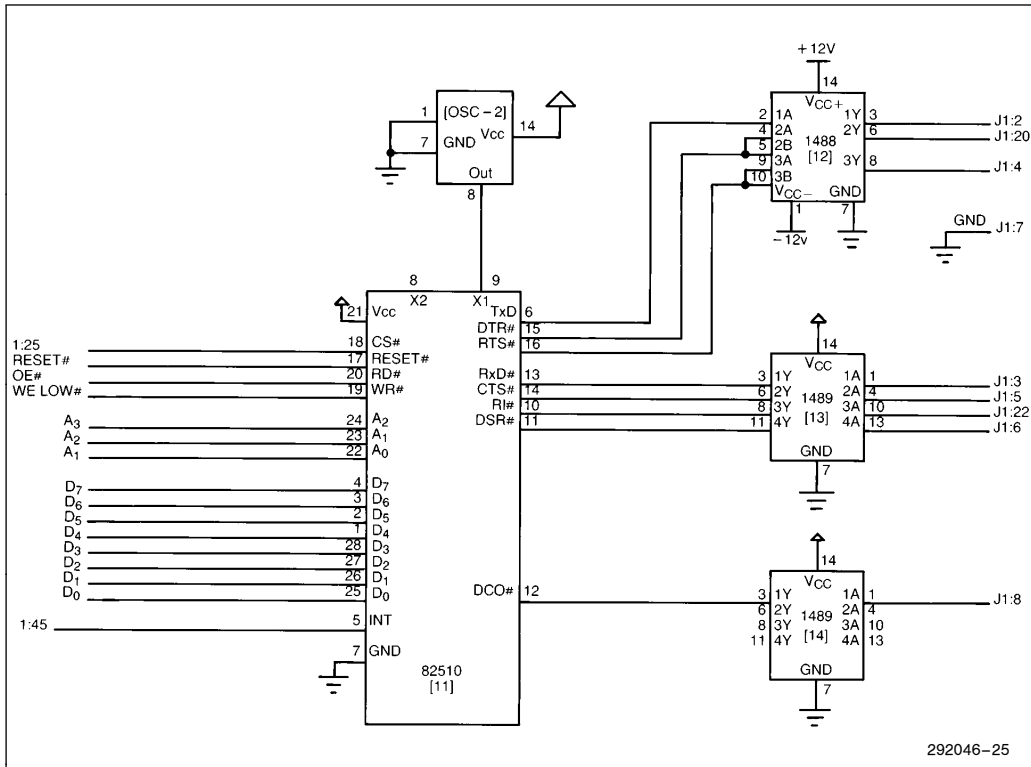








292046-24



256K FLASH MEMORY DEMO PARTS LIST

Device	Component	Pins	Description
[1]	80C186	68	16-bit high integration CPU
[2,3,4]	74HC573	20	Latch
[5,6]	74HC245	20	Transceiver
[7]	74HC32	14	OR gate
[8L,8H]	27C64	28	16 Kbyte EPROM
[9L,9H]	28F256A	32	64 Kbyte flash memory
[10L,10H]	32K x 8 SRAM	28	64 Kbyte SRAM
[11]	82510	28	Asynchronous Serial Controller
[12]	14C88	14	RS-232 Line Driver
[13,14]	14C89	14	RS-232 Line Receiver
[15]	82C55A	40	Programmable Peripheral Controller
[16]	PM7006	24	DC/DC Converter (5V–12.00V)
[17]	7406	14	Inverter—Open Collector (O.C.)
C1	20 μ F	2	Capacitor for CPU reset
D1	1N914	2	Diode for CPU reset
F1	BUZ11A	3	MOSPOWER nFET
J1	DB-25	25	Connector (male)
OSC-1	20 MHz	14	CPU Oscillator
OSC-2	18.432 MHz	14	Serial Controller Oscillator
R1	10 K Ω	2	$\frac{1}{4}$ W, 10% Resistor for CPU reset
R2	1 K Ω	2	$\frac{1}{4}$ W, 10% Resistor for O.C. pull-up
SW1		3	Momentary Push Button for CPU reset

NOTES:

- Place a 0.1 μ F bypass capacitor at the V_{CC} input of each IC.
- Place a 0.1 μ F bypass capacitor on the V_{PP} input of each 28F256 flash memory.

28F512 UPGRADE FOR THE 80C186/FLASH MEMORY DESIGN

To upgrade the 80C186/Flash memory design to handle 28F512's, the range of the CE# signal has to be increased. There are a number of ways to generate a CE# signal that will span the 128 Kbyte address range of two 28F512 devices.

- AND two of the current MCS lines together (defined for 64 Kbytes each); or

- Change the MCS individual block-select size from 64 Kbytes:

```
MMCS__VALUE = 41F8H,
MPCS__VALUE = 0A0B8H
```

to 128 Kbytes:

```
MMCS__VALUE = 01FEH,
MPCS__VALUE = 0C0BEH
```

Also, cut the CE# trace to the RAM sockets. Then wire MCS0# to the RAM CE#. This eliminates the MCS0# and LMCS# range overlap caused by increasing the MCS range to 128 Kbytes. See 80C186 Data Sheet page 21 and 22 (Order # 270354).

28F010 UPGRADE TO THE 80C186/FLASH MEMORY DESIGN

To upgrade the 80C186/flash memory design to handle 28F010's, a CE# signal has to be generated. There are a number of ways to generate a CE# signal that will span the 256 Kbyte address range of two 28F010 devices.

1. AND two of the MCS lines together (defined for 128 Kbytes each as noted in the 28F512 modifications):

Cut the LMCS trace to the RAM sockets. Connect MCS0# to CE# on the RAM sockets (U10L,UH).

Cut the MCS2# trace to the flash memory. Add an AND gate. Connect MCS2# (cut trace) and MCS3# to the inputs of the AND gate. Then wire the AND gate output to the CE# of the flash memories.

Also, change the onboard memory MCS register to:
 MMCS_VALUE = 01FEH, MPCS__
 VALUE = 0C0BEH [128K blocks],

and delete:

LMCS_REG and LMCS_Value.

2. Add a decoder;

Add a decoder (74HC138). Connect address lines A18 and A19 to the B and C inputs of the decoder. Tie the A input of the decoder low, and enable all the enables. By using outputs Y0, Y2, Y4, and Y6, you have four CE# lines decoding 256 Kbyte blocks each.

Cut the MCS2# trace to the flash memories. Connect the Y2 output from the decoder to the CE# input of the flash memory.

3. Replace the address latch (U2) with a PLD that latches and decodes.

Program a 5C032 as an integrated latch and decoder. Replace the upper address latch [U2] with the Intel 5C032 EPLD. Cut the CE# trace to the flash memories. Connect the flash memories' CE# to the 5C032 pin 12. This maps the address space 40000H to 7FFFFH. See Figures 1 and 2 for a comparison of the 74HC573 (U2) and programmed 5C032 pin outs. Figure 3 is the source code for the EPLD.

Also, change the value of the MMCS and MPCS registers to 64 Kbyte blocks so that the MCS0# range does not overlap the LMCS range.

MMCS_VALUE = 41F8H, MPCS__
 VALUE = 0A0B8H.

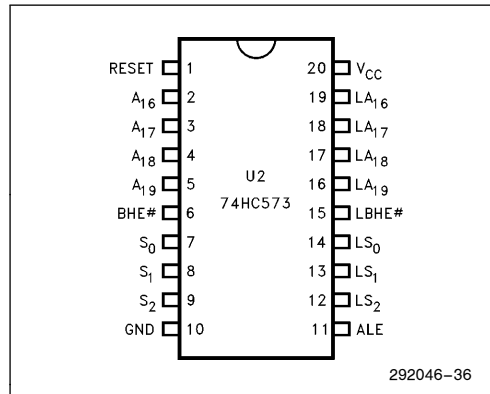


Figure 1. Latch Pinout

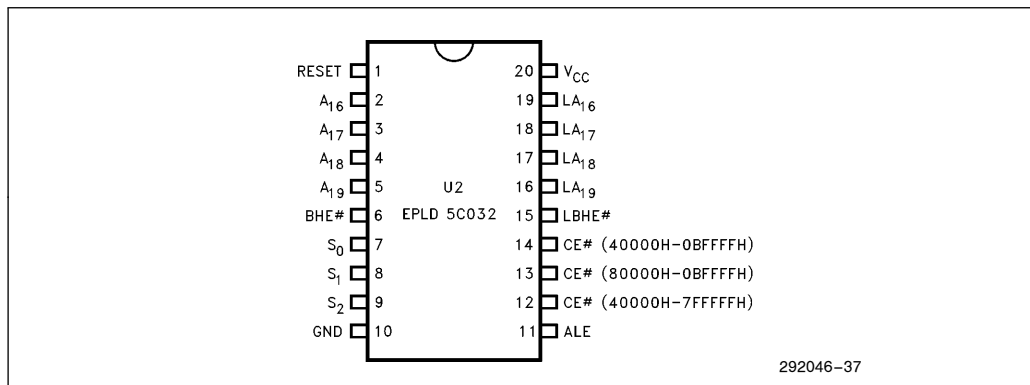


Figure 2. Integrated Latch and Decoder

```

Thom Bowns - PLFG Applications
Intel
January 13, 1989
EPLD HOTLINE: 1-800-323-EPLD
002
5C032
Custom Latched Decoder
OPTIONS: TURBO=ON
PART: 5C032

INPUTS: ALE@11, RESET@1, A19@5, A18@4, A17@3, A16@2, nBHE@6

OUTPUTS: LA18@17, LA17@18, LA16@19, LnBHE@15, nCE3@14, LA19@16,
nCE2@13, nCE1@12

NETWORK:
ALE = IN (ALE)
RESET = INP (RESET)
nRESET = NOT (RESET)
A19 = INP (A19)
A18 = INP (A18)
A17 = INP (A17)
A16 = INP (A16)
nBHE = INP (nBHE)
LA19, LA19 = COIF (LA19d, nRESET)
LA18, LA18 = COIF (LA18d, nRESET)
LA17, LA17 = COIF (LA17d, nRESET)
LA16, LA16 = COIF (LA16d, nRESET)
LnBHE, LnBHE = COIF (LnBHE, nRESET)
nCE3, nCE3 = COIF (nCE3, nRESET)
nCE2, nCE2 = COIF (nCE2, nRESET)
nCE1, nCE1 = COIF (nCE1, nRESET)

EQUATIONS:
LA19d = A19 * ALE + LA19 * !ALE;
LA18d = A18 * ALE + LA18 * !ALE;
LA17d = A17 * ALE + LA17 * !ALE;
LA16d = A16 * ALE + LA16 * !ALE;
LnBHEd = nBHE * ALE + LnBHE * !ALE;
nCE3d = nCE3EQN * ALE + nCE3 * !ALE;
nCE2d = nCE2EQN * ALE + nCE2 * !ALE;
nCE1d = nCE1EQN * ALE + nCE1 * !ALE;
nCE2EQN = !(A19 * !A18);
nCE1EQN = !(LA19 * A18);
nCE3EQN = !(LA19 * A18 + A19 * !A18);

END$

```

Figure 3. Source Code for the Integrated Latch and Decoder