What is MPS E2?

by Application Staff, November 1999

Introduction

The X84256 MPS™ E^2 (Microprocessor Compatible Serial E2PROM) is a 256K-bit nonvolatile serial memory organized as 32K x 8. However, due to the sequential nature of the device, the use of consecutive reads or writes to the device also allows it to be used in 16-bit or 32-bit environments. This device supports an 64-byte page with a typical nonvolatile write cycle time of 5ms. Additionally, there is a write protect pin (WP) to disable all nonvolatile writes to the device.

The unique feature of this device is its' ability to directly interface with a parallel data bus without glue logic. It's small size, low power, and low cost make it an ideal alternative to parallel access nonvolatile memory devices. System designers currently using parallel access devices to hold configuration data (e.g. motherboards, add-on cards, PCMCIA cards, Plug and Play cards, data recording systems, etc...) will especially benefit from the features of the X84256.

This CMOS device utilizes Xicor's proprietary Direct Write™ E^2PROM cell technology which allows for an endurance of at least 1,000,000 write cycles per byte and a minimum data retention of 100 years.

X84256 Data Bus Cycles

An understanding of how the MPS E^2 serial interface works can be gained from the included bus cycle diagrams. The system designer accesses the X84256 using standard "read" or "write" activity on a parallel data bus. The interface is accomplished by directly connecting a single I/O pin from the data bus, as well as the $\overline{WE}$ and $\overline{OE}$ signals generated by the system processor and an address decoded $\overline{CE}$. When the X84256 is selected, the device will interpret each "read" or "write" on the data bus as a single bit in the serial interface protocol. Since the device is in standby mode when $\overline{CE}$ is HIGH, the serial interface scheme is fully static and can be interrupted at any time to perform other bus activities. At a later time, communications with the device can be resumed from the point when the communications were interrupted.

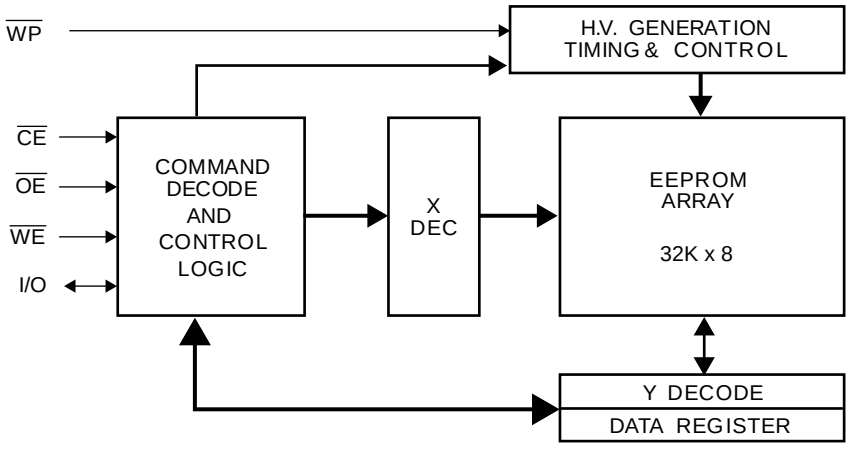
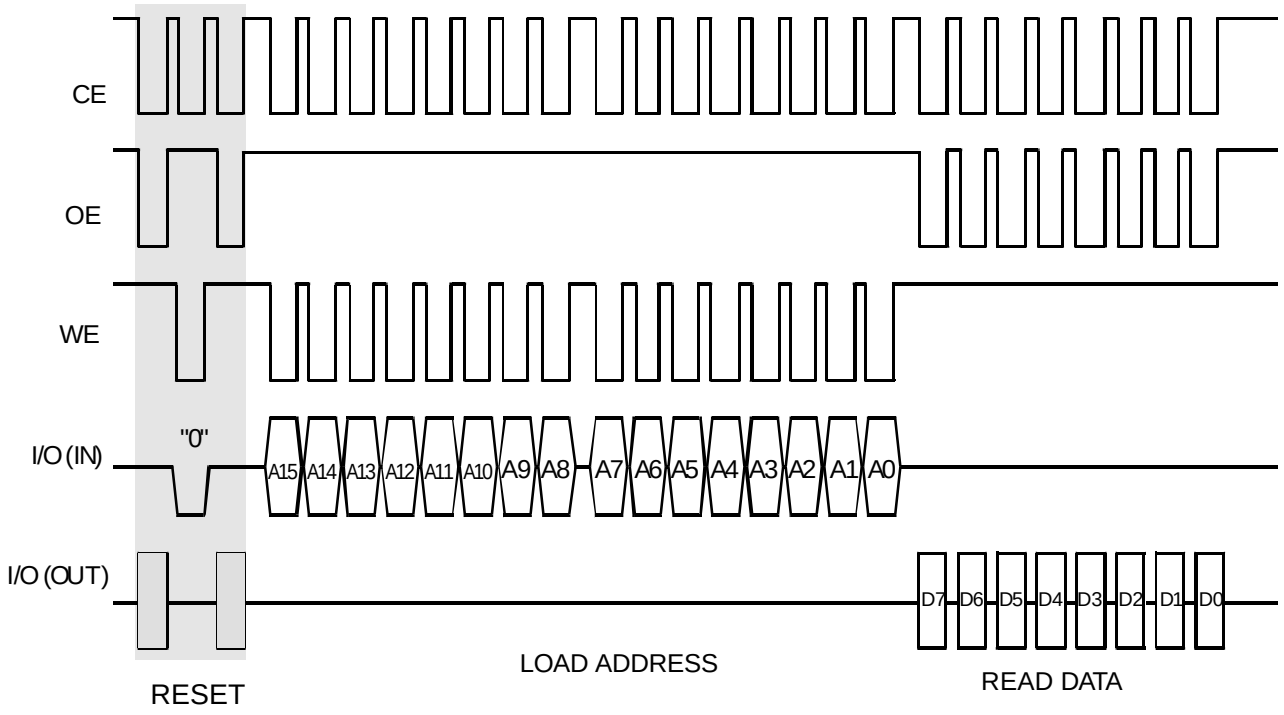


Figure 1. X84256 MPS™ E2 Block Diagram

"Direct Write" and "MPS" are trademarks of Xicor, Inc.

In order to complete the interface, there are two additional unique sequences to reset the device and to initiate a nonvolatile write cycle. A reset condition is recognized by the X84256 whenever the system processor issues the following sequence: “read”, “write a zero”, and “read”. This 3-bus cycle sequence is used to initialize the part before a read or write sequence or to interrupt a sequence already in progress. An initiate nonvolatile write cycle condition is recognized by the X84256 whenever the system processor issues the following sequence: “read”, “write a one”, and “read”. This sequence is used to begin an internal write cycle after data has been loaded into the device. An additional feature of the device is the ability to determine the early completion of an internal nonvolatile write cycle. This can be accomplished by “polling” the level of the I/O pin during a nonvolatile write cycle. When the I/O pin can be read as a logic HIGH, then the cycle is complete and the part is again able to interact within the system.

The X84256 read sequences should usually be initiated with a reset sequence, followed by 16 consecutive “writes” to the device without a “read”. These 16-bits are for the initial memory address to be accesssed (MSB first).



When Accessing: X84256 Array: A15=0

Figure 2. X84256 Read Cycle

At this point, the system processor can sequentially “read” any number of bits from the device beginning at the initial address.

Write sequences follow the same format. Typically, the X84256 can be completely rewritten using page writes in less than 325ms when the technique of I/O polling for the early completion of each nonvolatile write cycle is used.

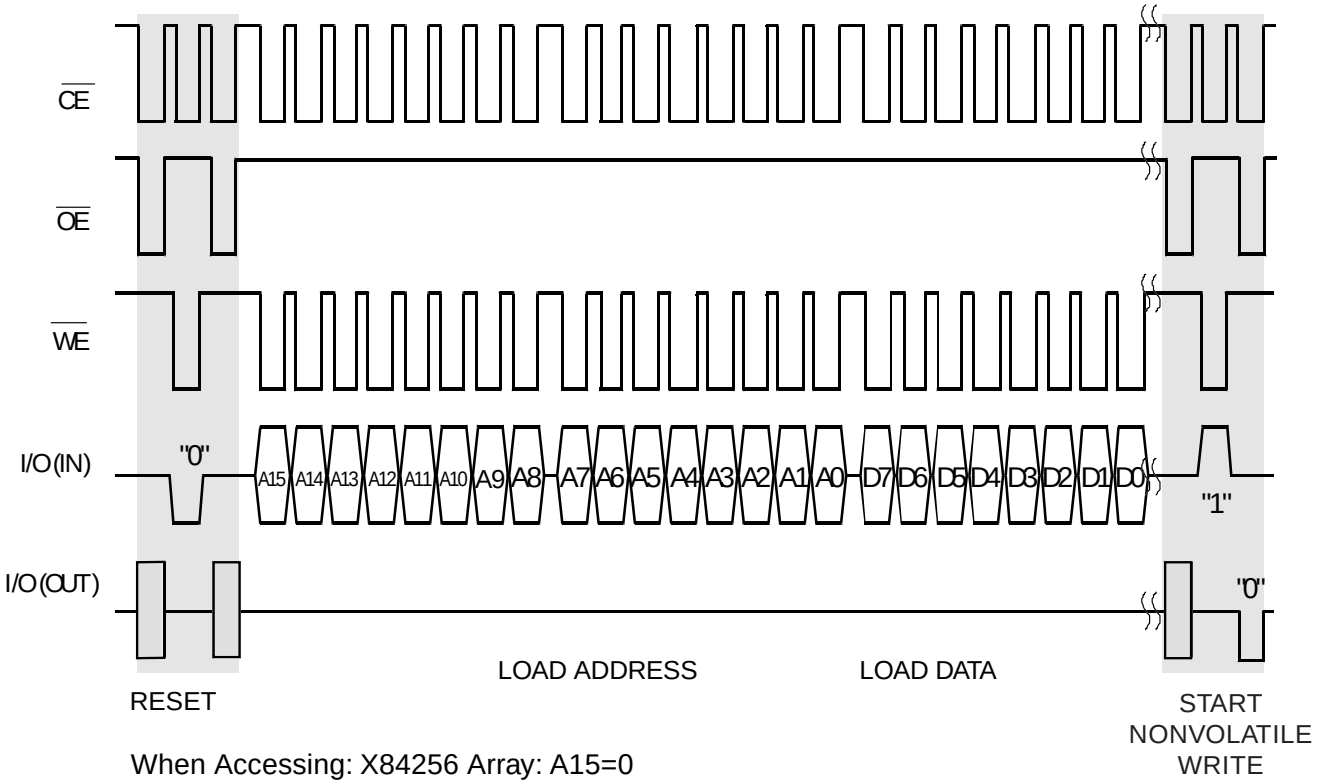


Figure 3. X84256 Write Cycle

Interfacing to the X84256

The X84256 is designed so that it is compatible with systems using either $\overline{CE}$ controlled write cycles (e.g. Motorola processors) or $\overline{WE}$ controlled write cycles (e.g. Intel processors). The following circuits show how simply the interfacing can be accomplished for different environments, such as the 68000, 80x86, 8051 microcontrollers, and TMS320 DSP processors.

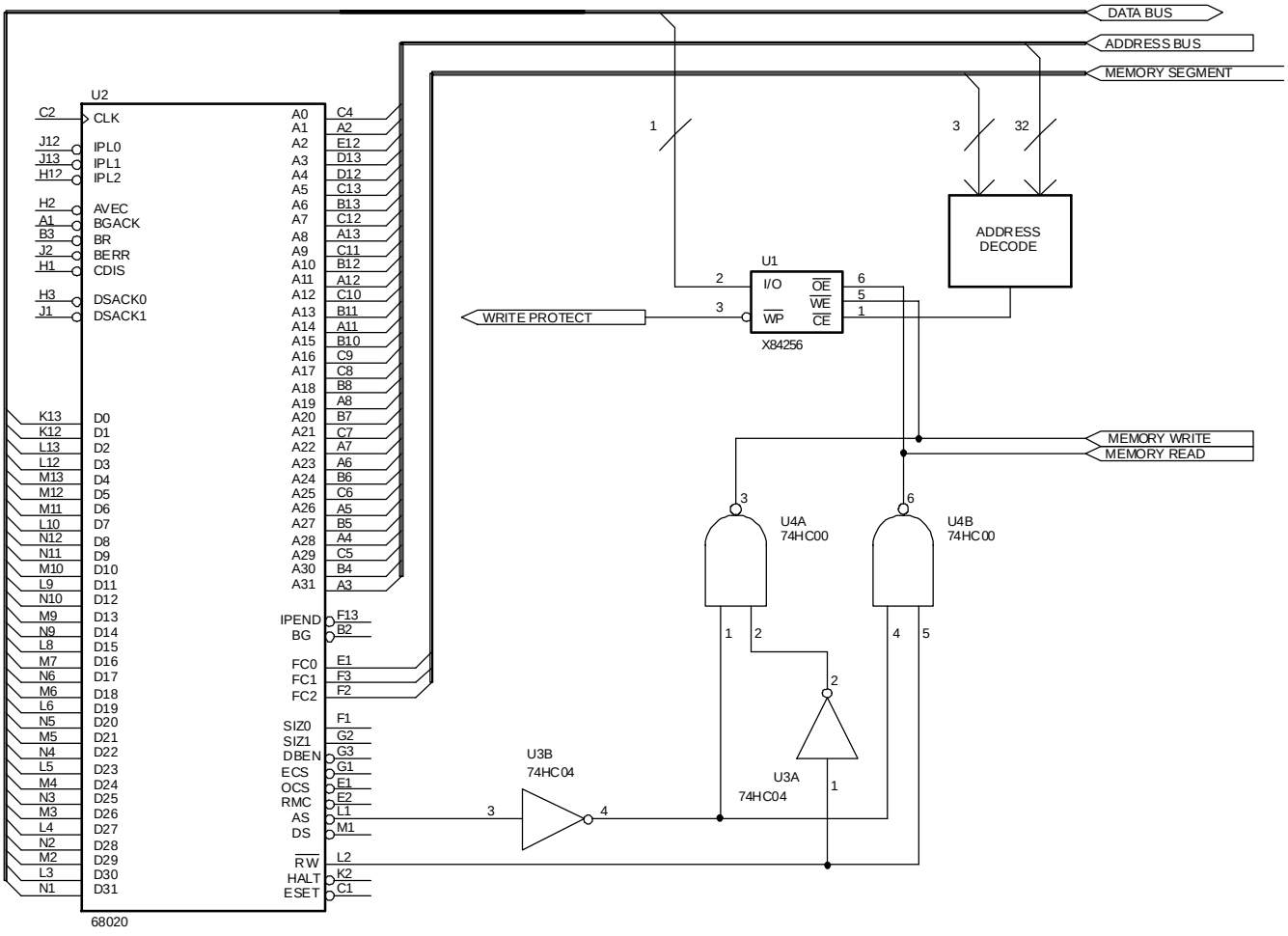


Figure 4. $\overline{CE}$ Controlled Write Cycle Environment (68000)

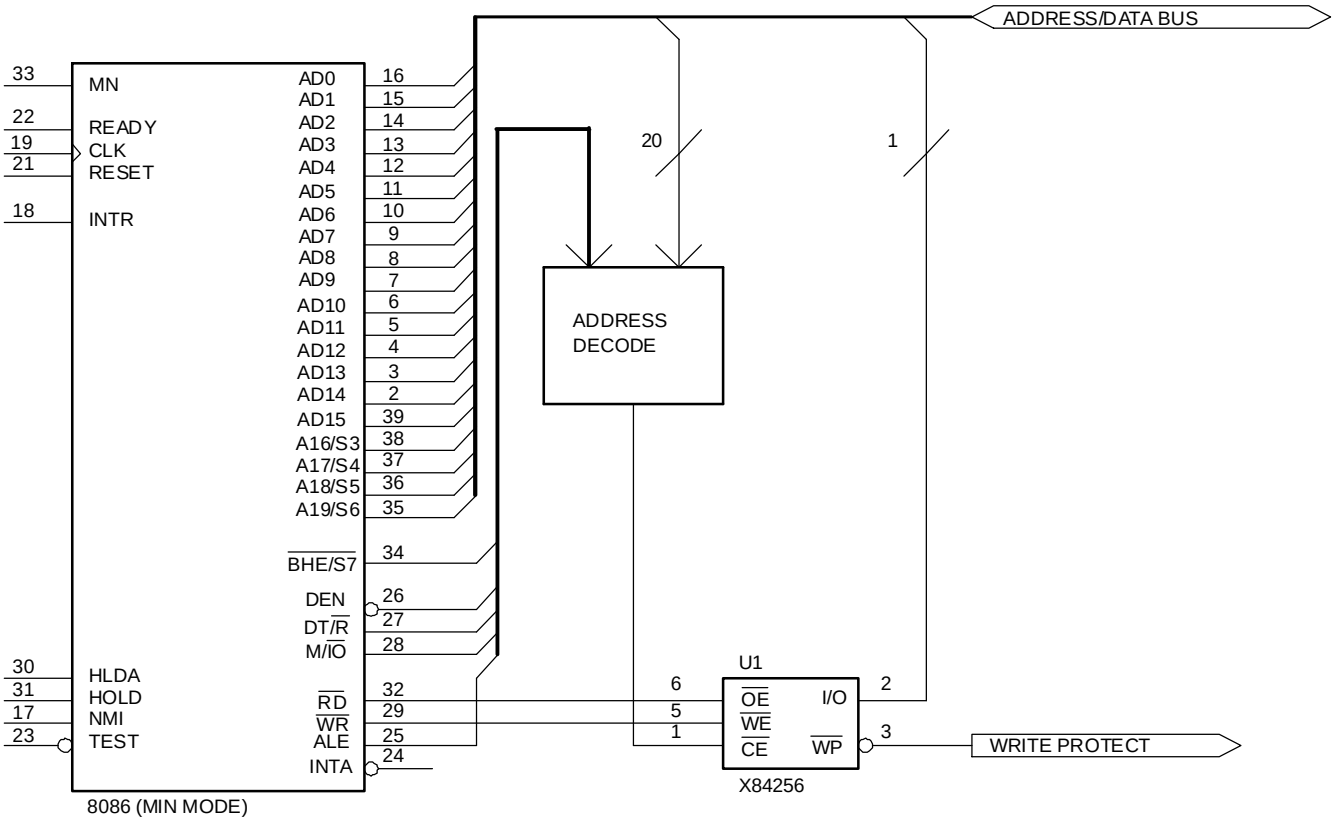


Figure 5. $\overline{WE}$ Controlled Write Cycle Environment (80x86)

Although the device is specified as requiring 100ns read cycle (tRC) and write cycle (tWC) times, there should be no problem interfacing to typical high speed busses. These cycle times are sums of the LOW pulse ($\overline{OE}$ or $\overline{WE}$) and the HIGH phase required before the next LOW pulse. As long as the LOW pulse minimum lengths are satisfied according to the bus timings of the X84256, then in software, a designer can accomodate the minimum cycle length requirement by issuing NOP instructions (or any instructions that will take sufficient time to execute).

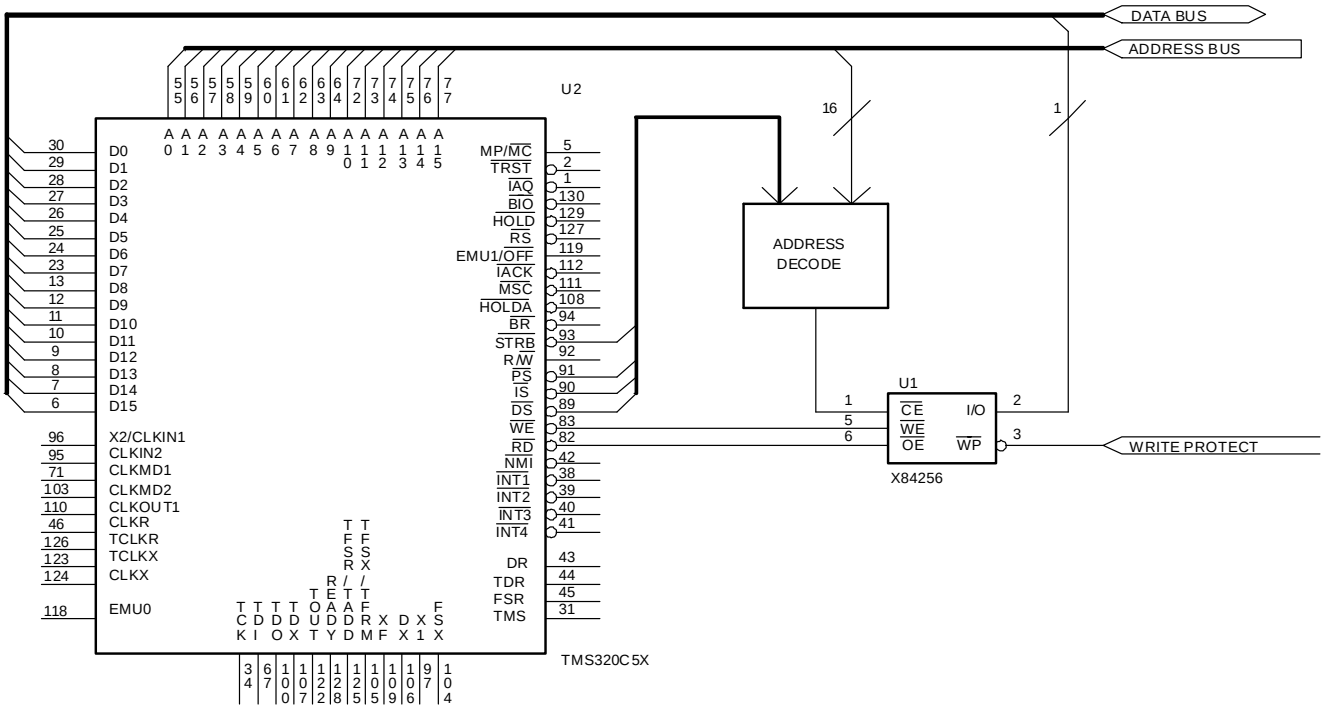


Figure 6. DSP systems (TMS320)

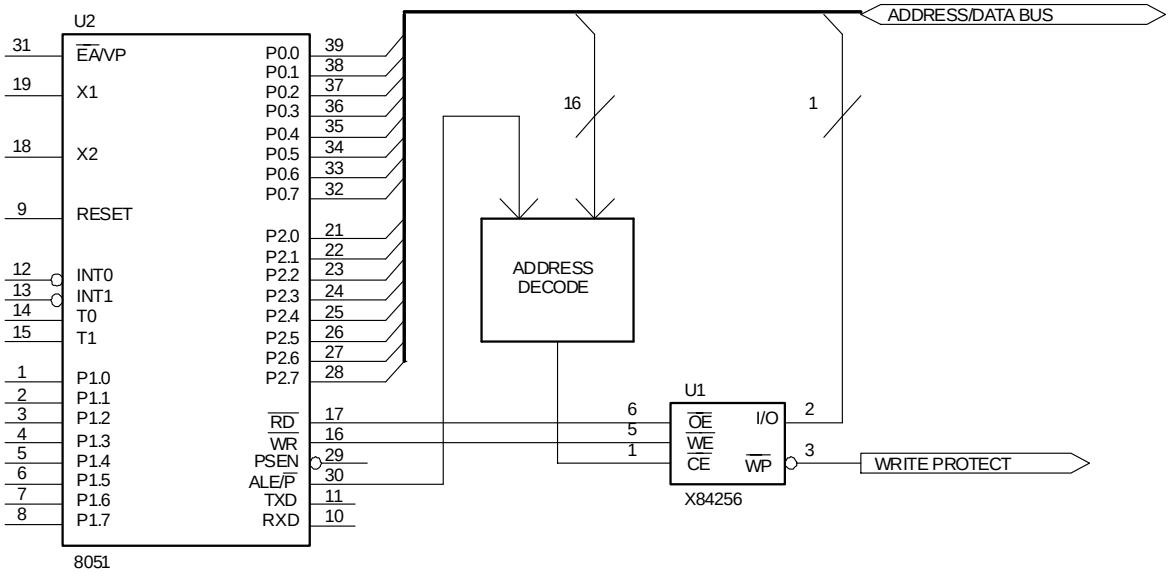


Figure 7. Microcontroller Systems (8051)

The X84256 can also be an effective alternative for systems that otherwise have serial data port capabilities. For many microcontrollers, a designer is limited to only a few bit, addressable I/O port pins. Instead of using one of these pins strictly for E^2PROM communications, a designer can use the X84256. A commonly encountered circuit uses 2 I/O port pins of an 8051 microcontroller to interface to an I^2C E^2PROM. One previous circuit example shows how the X84256 frees up these port pins.

```
*****************************************************************************************************/
/*
/*                                Xicor X84256 MPS E2
/*
/* The following C subroutines are provided to interface with an X84256 that is memory mapped  into the
/* I/O space of an IBM PC or compatible environment.  With these subroutines, a designer can read multiple
/* data bytes from the X84256 by using the simple function call:
/*
/*             read_X84256(int no_bytes,int addr,int io_port,unsigned char *bytes)
/*
/* Any number of bytes (no_bytes) can be read sequentially from the starting address (addr) of the X84256
/* mapped into a given I/O address (io_port).  The results are passed back sequentially to a series of
/* system locations pointed to by (bytes).  Care should be taken to provide enough locations for each data
/* byte read.  Additionally, either a byte write or page write to the X84256 can be initiated with the
/* following call:
/*
/*             write_X84256(int no_bytes,int addr,int io_port,unsigned char *bytes)
/*
/* Up to 8 bytes (as specified by no_bytes) can be written to the X84256 at an I/O address (io_port)
/* beginning at a memory address (addr).  Sending more than 8 bytes will cause a "wrap-around" of data on
/* the chosen page.  The data to be written is contained in an array (pointed to by *bytes).  These rou
/* tines can be appended to other ANSI C code with the following (or similar) compiler directive:
/*
/*             #include "c:\xicor.c                                                    "
/*
/* Note that the source code should have an #include <dos.h> statement since the code (xicor.c) needs
/* access to the inportb() and outportb() functions.
/*                                                                                          GHC IV
/*****************************************************************************************************/

/*****************************************************************************/
/* Sends RESET sequence to the X84256 at io_port
/*****************************************************************************/
void x84reset(int io_port) {
       inportb(io_port);                        /* read bit from io_port */
       outportb(io_port,0x00);                  /* write a LOW bit to io_port */
       inportb(io_port);                        /* read bit from io_port */
}

/*****************************************************************************/
/* Writes a one to the X84256 at io_port
/*****************************************************************************/
void x84write_one(int io_port) {
       outportb(io_port,0xff);                  /* write a HIGH bit to io_port */
}

 /*****************************************************************************/
/* Writes a zero to the X84256 at io_port
/*****************************************************************************/
void x84write_zero(int io_port) {
       outportb(io_port,0x00);                  /* write a LOW bit to io_port*/
}
/*****************************************************************************/
/* Reads a bit from the X84256 at io_port
/*****************************************************************************/
int x84read(int io_port) {
int bit_val;
       bit_val = inportb(io_port) & 1;          /* read bit D0 from io_port */
       return(bit_val);                         /* return bit value to calling routine */
}

/*****************************************************************************/
/* Polls for the early completion of a nonvolatile write cycle in the X84256 at
/* io_port
/*****************************************************************************/
void x84poll(int io_port) {
int bit_temp;
       do {
              bit_temp=x84read(io_port);        /* continuously read bits until the bit is HIGH */
       } while (bit_temp == 0);
```

```
}

/*****************************************************************************************/
/* Initiates and completes a nonvolatile write cycle in the X84256 at io_port
/*****************************************************************************************/
void x84write_start(int io_port) {
        x84read(io_port);                           /* read bit from io_port */
        x84write_one(io_port);                      /* write a LOW bit to io_port */
        x84read(io_port);                           /* read bit from io_port */
        x84poll(io_port);                           /* poll the I/O pin for completion of write cycle */
}

/*****************************************************************************************/
/* Sends all 16 required address bits, including don't cares, to the X84256 at
/* io_port
/*****************************************************************************************/
void x84addr_send(int dont_care,int io_port,int addr) {
int addr_bit,mask_addr,addr_temp;
        for (addr_bit = 0; addr_bit < dont_care; addr_bit++) {
                x84write_zero(io_port);             /* loop and send don't cares */
        }
        mask_addr = 256;                            /* bit mask to isolate address MSB */
        for (addr_bit = 0; addr_bit < (16-dont_care); addr_bit++) {
                                                    /* loop through all 9 address bits */
                addr_temp = addr & mask_addr;       /* mask to determine next required address bit */
                if (addr_temp == 0)                 /* if address bit is LOW, then ... */
                      x84write_zero(io_port);       /* write a LOW bit to io_port */
                else
                      x84write_one(io_port);        /* otherwise, write a HIGH bit to io_port */
                mask_addr = mask_addr >> 1;         /* shift bit mask right to get next bit */
        }
}

 /*****************************************************************************************/
/* Sends all 8 data bits to the X84256 at io_port
/*****************************************************************************************/
void x84data_send(int io_port,int data) {
int mask_data,data_temp,data_bit;
        mask_data = 128;                            /* bit mask to isolate data MSB */
        for (data_bit = 1; data_bit < 9; data_bit++) {/* loop through all 8 data bits */
                data_temp = data & mask_data;       /* mask to determine next required data bit */
                if (data_temp == 0)                 /* if data bit is LOW, then ... */
                      x84write_zero(io_port);       /* write a LOW bit to io_port */
                else
                      x84write_one(io_port);        /* otherwise, write a HIGH bit to io_port */
                mask_data = mask_data >> 1;         /* shift bit mask right to get next bit */
        }
}

/*****************************************************************************************/
/* Reads 8 data bits from the X84256 at io_port and reconstructs the databyte
/*****************************************************************************************/
int x84data_get(int io_port) {
int n,bit_temp[9];
bit_temp[0]=0;                                  /* clear array position 0, resetting previous sum */
        for (n=1;n<9;n++) {                     /* loop through 8 data bits */
                bit_temp[n] = x84read(io_port)+2*bit_temp[n-1];
                                                /* at the current position, shift the previous sum of */
                                                /* data bits left, read the next bit, add it to previous */
        }                                       /* sum, and store the result */
        return(bit_temp[8]);                    /* return the data byte value to the calling routine */
}

/*****************************************************************************************/
/* General READ master routine that is called by the user in order to read a
/* number of bytes from the X84256 at io_port
/*****************************************************************************************/
void read_X84256(int no_bytes,int addr,int io_port,unsigned char *bytes) {
int n;
        x84reset(io_port);                          /* reset the X84256 */
        x84addr_send(7,io_port,addr);               /* send the address bits (don't cares & addr) */
        for (n=0;n<no_bytes;n++) {                  /* loop through all bytes to be read (no_bytes) */
                bytes[n]=x84data_get(io_port);      /* receive each byte and store sequentially */
        }
}

/*****************************************************************************************/
/* General WRITE master routine that is called by the user in order to write a
```

```
/* number of bytes to the X84256 at io_port
/*************************************************************************************/
void write_X84256(int no_bytes,int addr,int io_port,unsigned char *bytes) {
int n;
        x84reset(io_port);                              /* reset the X84256 */
        x84addr_send(7,io_port,addr);                   /* send the address bits (7 don't cares & addr) */
        for(n=0;n<no_bytes;n++) {                       /* loop through all bytes to be written (no_bytes) */
                x84data_send(io_port,*(bytes+n));       /* sequentially send each byte */
        }       x84write_start(io_port);                /* initiate nonvolatile write cycle */
}

 /*************************************************************************************/
/* Main listing example to utilize these functions
/*
/* When used in a larger program, the following include directives must be
/* included prior to compilation.  The example shows a page write of length
/* 4 bytes being initiated at address 0x000 of an X84256 memory mapped to
/* address 0x303. Data to be written is stored sequentially in send_buffer.
/* Similarly, 22 consecutive bytes are read from this X84256 starting at address
/* 0x01F and stored to receive_buffer.
/*************************************************************************************/

#include <c:\xicor.c>
#include <dos.h>
unsigned char send_buffer[8], receive_buffer[512];   /* maximum buffer sizes needed for X84256 protocol */
.
.
.

main () {
        write_X84256(4,0,0x303,&send_buffer);
        read_X84256(22,0x1F,0x303,&receive_buffer);
}
```

littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.