Interfacing to Xicor's X84641/129 MPS™ EEPROMs

by Applications Staff

The following C routines can be used for interfacing to an MPS EEPROM and can be easily adapted for many different hardware platforms. Figure 1 shows a typical ISA interface circuit. Depending on the position of jumpers J1 and J2, an MPS EEPROM is memory-mapped into 1 of 4 I/O address banks (as determined by jumper J2). Jumper J1 determines the exact address within the bank. This allows for any address to be chosen in the following ranges: 301h-307h, 309h-30Fh, 311h-317h, or 319h-31Fh. The single unused address in the selected bank (i.e. 300h, 308h, 310h, or 318h) is reserved for a D flip-flop used to toggle the $\overline{WP}$ pin. This glue logic is only necessary to provide a generic ISA bus interface for an MPS EEPROM. For such applications, much of this logic may already exist on the target PC card or can be eliminated with a user supplied CE strobe (i.e. jumper J3).

This code can be obtained from www.xicor.com.

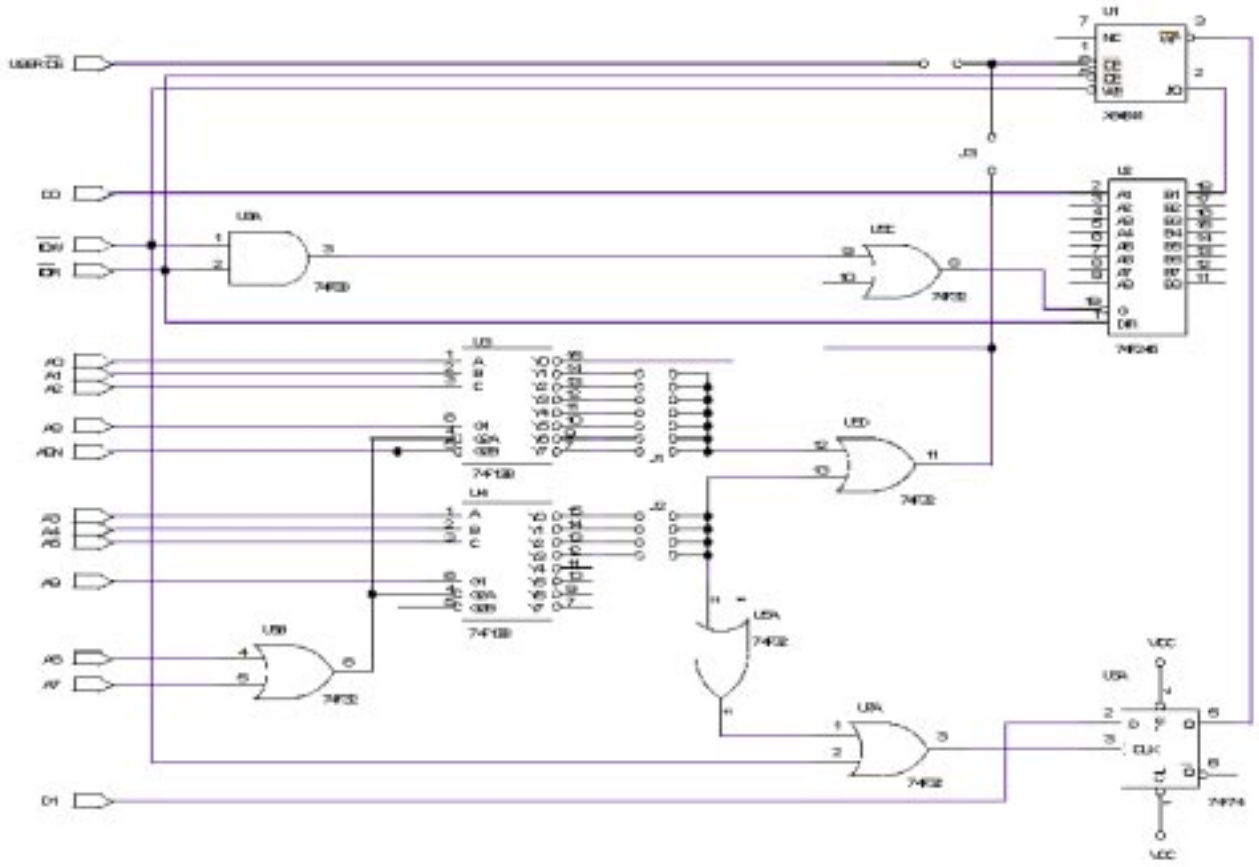

Figure 1. Circuitry used to interface with X84641 or X84129.

```
/*************************************************************************/
/*                                                                       */
/* C Code for Xicor's X84641/129 family of MPS EEPROMs                   */
/* (for example, save these routines as a file called "xicor.c")         */
/*                                                                       */
/* The following C subroutines are provided to interface with Xicor's    */
/* family of high density MPS EEPROMs. This code is written to perform   */
/* with the X84641 (64K bits) and X84129 (128K bits)  */
/* devices. This code should also be compatible with future devices in   */
/* this family with only trivial changes.                                */
/*                                                                       */
/* This code was tested using the XK84 development system hardware,      */
/* which consists of a standard ISA interface card with I/O address      */
/* decoding logic (as shown in Figure 1). This software interfaces to a  */
/* device that has been memory mapped into this IBM PC cOmpatible I/O    */
/* space.                                                                */
/*                                                                       */
/* With these subroutines, a designer can read multiple data bytes from  */
/* the X84641/129 by using the simple function call:                     */
/*                                                                       */
/* read_X84MPS(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 X84641/129 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 X84641/129     */
/* can be initiated with the following call:                             */
/*                                                                       */
/* write_X84MPS(int no_bytes,int addr,int io_port,unsigned char *bytes)  */
/*                                                                       */
/* Up to 32 bytes (as specified by no_bytes) can be written to the       */
/* X84641/129 at an I/O address (io_port) beginning at a memory          */
/* address (addr).  Sending more than 32 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 routines 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 X84641/129 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 X84641/129 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 X84641/129 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 X84641/129 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 X84641/129 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 X84641/129 at io_port
/*************************************************************/
void x84write_start(int io_port) {
 x84read(io_port);/* read bit from io_port */
 x84write_one(io_port);/* write a HIGH 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 unused
/* upper address bits, to the X84641/129 at io_port
/*************************************************************/
void x84addr_send(int unused_bits,int io_port,int addr) {
int addr_bit,mask_addr,addr_temp;
for (addr_bit = 0; addr_bit < unused_bits; addr_bit++) {
```

```
x84write_zero(io_port);/* loop and send unused bits as LOW */
}
mask_addr = 4096;/* bit mask to isolate address MSB */
/* mask_addr = 4096 for X84641 */
/* mask_addr = 8192 for X84129 */
                          /* mask_addr = 1024 for X84161 */

for (addr_bit = 0; addr_bit < (16-unused_bits); addr_bit++) {   /* loop through all */
      /*  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 X84641/129 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 X84641/129 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 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 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 X84641/129
/* at io_port
/*************************************************************/
void read_X84MPS(int no_bytes,int addr,int io_port,unsigned char *bytes) {
int n;
int density;
density = 3;/* density = 3 for X84641*/
/* density = 2 for X84129 */
/* density = 5 for X84161 */
x84reset(io_port);/* reset the X84161/641/129 */
x84addr_send(density,io_port,addr);/* send the address bits */
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 X84641/129
/* at io_port
/*************************************************************/
void write_X84MPS(int no_bytes,int addr,int io_port,unsigned char *bytes) {
int n;
int density;
density = 3;/* density = 3 for X84641 */
/* density = 2 for X84129 */
/* density = 5 for X84161 */
x84reset(io_port);/* reset the X84161/641/129 */
x84addr_send(density,io_port,addr);/* send the address bits */
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 18 bytes being
/* initiated at address 0x021 of an X84641 memory mapped to
/* address 0x303. Data to be written is stored sequentially
/* in send_buffer. Similarly, 42 consecutive bytes are read
/* from this X84641 starting at address 0x01F and stored to
/* receive_buffer.
/*
/* The loading of send_buffer[] is not explicitly shown.
```

```
/*********************************************************/

#include <dos.h>
#include <c:\xicor.c>

unsigned char send_buffer[32];/* worst case buffer sizes needed */
unsigned char receive_buffer[8192];/*  for X84641/129 protocol */
/* receive_buffer[8192] for X84 */
/* receive_buffer[16384] for X84129 */
/* receive_buffer[2048] for X84 */
main () {
outportb(0x300,0xff);/* only used for setting /WP HIGH on the XK84
write_X84MPS(18,0x21,0x303,&send_buffer);
read_X84MPS(42,0x1F,0x303,&receive_buffer);
}
```