Programming Atmel's AT29 Flash Family

Flash

Application Note

(AN-3)

Introduction

Atmel offers a diverse family of small sector Flash memory devices ranging in density from 256K to 4M bits. These devices read and program with a single voltage supply. The nominal supply voltage is 5V for the AT29CXXX, 3.3V for the "low voltage" AT29LVXXX, and 3V for the "Battery Voltage" AT29BVXXX Flash memory family. The entire Flash PEROM product line is designed to allow users to have one common programming algorithm for all three Flash voltage families. Therefore, upgrading from one density to another and from a higher voltage to a lower voltage device is simplified.

This application note describes the design benefits of Atmel's AT29 Flash architecture as well as how the device ID feature is used to adjust for varying densities and supply voltages. In addition, Atmel's Software Data Protection (SDP) feature, which prevents inadvertent writes, is described. An example is given to illustrate the ease with which the programming software can be written to accommodate four different 4M bit Flash devices: the AT29C040, the AT29LV040, and the newer generation Flash devices, the AT29C040A and the AT29LV040A.

Hardware and software has been developed to demonstrate the relevant design issues. The demo uses an AT89C51 Flash-based microcontroller (which has the same pinout and instruction set as an 80C51) as the host processor and a "C" language program for the software. The software automatically adjusts the amount of time required for programming the varying voltage versions of the 4M bit Flash devices in addition to accommodating for their different sector sizes.

The AT89C51, a member of Atmel's growing family of Flash microcontroller devices, features 4K bytes of in-system reprogrammable Flash memory (see Atmel application note "AT89C51 In-Circuit Programming" for additional information). Current and future versions of Atmel's microcontroller family incorporate from as little as 1K byte of Flash memory to as much as 128K bytes, providing many density options for different applications. Other versions will also include special architectures such as a combination of Flash and parallel EEPROM memory on board.

Programming Flash Devices

Unlike Atmel's Flash memories, previous generations of Flash memories had large sectors, typically 4K to 128K bytes, and required that an entire sector be erased prior to programming. Generally, the sector erase cycle time was hundreds or thousands of milliseconds and could be as long as 30 seconds for the entire memory array. In addition, a separate high voltage supply was required for a write and erase operation. Atmel's AT29 Flash memory family has simplified usage by having only one supply voltage, reducing the sector size, having the programming similar to an SRAM write operation, and decreasing significantly the total programming time.

Small sector sizes reduce the amount of system resources necessary for programming. When only a few bytes in a Flash memory need to be altered, a RAM image of the Flash sector must be created. The RAM must then be altered with the new data, and the image trans-

Rev. 0518B–10/98

ferred back into the Flash device. Because Atmel's Flash devices have small sector sizes (from 64 to 512 bytes, depending on the memory density), the RAM requirements are much less than those of large sector Flash devices. Often, the system RAM available is sufficient for Atmel's Flash, whereas large sector Flash devices usually require an additional SRAM.

A second advantage of Atmel's AT29 Flash is that an entire sector can be updated during a single program operation, instead of the byte-by-byte programming of previous generation Flash memories. This saves significant programming time when updating an entire sector, especially when comparing Atmel's small sector devices with large sector devices. In addition, Atmel's devices do not require a sector erase prior to writing, thus saving additional programming time. The maximum sector program time is 10 msec for the AT29CXXX family and 20 msec for the AT29LVXXX/AT29BVXXX families.

AT29C040 and AT29C040A Architecture

The AT29C040 provides operation similar to a byte-wide SRAM. The device has eight data lines and 19 address lines. The familiar three input control lines are also present ($\overline{CE}$, $\overline{OE}$, $\overline{WE}$). Read operations are identical to an SRAM, but write operations are somewhat different due to the write cycle time (t_{WC}) requirements of all Flash memories. Flash write operations take several milliseconds to complete, compared to the nanosecond writes of SRAM devices. It should be noted that Atmel's AT29 Flash PEROMs require only a write operation; the erase operation is automatically performed internally in the device.

Data is loaded into the AT29C040 one sector at a time, with each sector consisting of 512 bytes. The sector chosen for modification is defined by the upper order address bits (A9-A18). The entire sector must be loaded during the write operation. Any byte not loaded during the sector load will contain FF (hex) after the write operation has completed. Address lines A0 through A8 define the location of the bytes within a sector. All data must be loaded into the same sector (A9 through A18 must remain constant) and can be randomly loaded within that sector.

The AT29C040A is identical to the AT29C040 except for the sector size and the Device ID Code (the Device ID Code is described later). The AT29C040A has a 256-byte sector (instead of a 512-byte sector) which is defined by address lines A8 through A18; the bytes within the sector are determined by address lines A0 through A7.

Software Data Protection (SDP)

One concern of systems designers when using nonvolatile programmable memories is the possibility of inadvertent write operations that can be caused by noise or by power-up and power-down sequences. Atmel's Flash memories provide a feature called Software Data Protection (SDP) that addresses this issue. The user can enable SDP upon receipt of the device from Atmel, and its usage is highly recommended. Data can be written into a sector with or without SDP enabled. However, once SDP has been enabled, the device requires that all subsequent write operations perform a series of "dummy" write operations before loading the chosen sector with data. The "dummy" writes consist of loading three known data values into three predefined addresses. This 3-byte sequence preceding a write operation virtually eliminates the chance of inadvertent write operations. The sequence is described below.

1. Load Data AA (hex) into Address 05555 (hex)
2. Load Data 55 into Address 02AAA
3. Load Data A0 into Address 05555
4. Load desired sector with data
5. Pause t_{WC} (device write cycle time)
6. Continue with next operation.

If SDP is enabled, any attempt to write to the device without the 3-byte command sequence will start a write cycle. However, no data will actually be written to the device, and during this "write" cycle time (t_{WC}), valid data cannot be read from the Flash.

Product and Manufacturer ID

Atmel's Flash memory devices allow the user to access both device and manufacturer information. This feature allows a system to determine exactly which Flash memory is being used. Once this is known, the host system can choose different algorithms for write operations in order to accommodate for differences in device density, V_{CC} requirements, sector size, and required write cycle time.

Product and manufacturer ID information is determined with the Software Product Identification procedure, which is similar to the Software Data Protection sequence. The sequence is described below.

1. Load Data AA (hex) into Address 05555 (hex)
2. Load Data 55 into Address 02AAA
3. Load Data 90 into Address 05555
4. Pause t_{WC} (device write cycle time)
5. Read Address 00000
   Data read is the Manufacturer Code
6. Read Address 00001
   Data read is the Device ID Code
7. Load Data AA into Address 05555
8. Load Data 55 into Address 02AAA
9. Load Data F0 into Address 05555
10. Pause t_{WC} (device write cycle time)

11. The device is returned to standard operating mode

The following table uses the 4M bit Flash as an example to illustrate the pertinent device information that can be determined once the Device ID Code is known. Please refer to the table at the end of this application note for information on other Flash devices.

Device	ID	V_{CC}	Sector Size	t_{WC}
AT29C040	5B	5.0V ± 10%	512 bytes	10 ms
AT29C040A	A4	5.0V ± 10%	256 bytes	10 ms
AT29LV040	3B	3.3V ± 0.3V	512 bytes	20 ms
AT29LV040A	C4	3.3V ± 0.3V	256 bytes	20 ms
AT29BV040	3B	3.0V ± 10%	512 bytes	20 ms
AT29BV040A	C4	3.0V ± 10%	256 bytes	20 ms

Programming Demonstration

Hardware and software descriptions have been prepared to demonstrate how Atmel's AT29 Flash memories can be reprogrammed. The descriptions are provided in the following two sections. A circuit schematic of the demonstration hardware and a source code listing of the software are also included.

Hardware Description

The demo hardware consists of a 12 MHz AT89C51 Flash-based microcontroller with 4K bytes of on-board Flash memory. The internal AT89C51 Flash memory is used for boot code, and the external 8K x 8 SRAM and the AT29C040A are mapped as data memory. The AT29C040A is also mapped as program memory to facilitate off-chip program execution. The AT89C51 can only access a maximum of 64K bytes of data memory space, while the AT29C040A has 512K bytes of storage capacity. To solve this size mismatch, the AT29C040A is bank switched into the AT89C51 data memory map in 8K byte blocks. The bank switching is performed with six general purpose I/O port bits on the AT89C51. The system address map is shown below.

System Address Map

AT89C51 Microcontroller	0000-1FFF	Internal Program Memory
8K x 8 Static RAM	2000-3FFF	Data Memory
AT29C040A	4000-5FFF	Program and Data Memory

Software Description

The software demonstrates how the Device ID Code can be used to allow a single program to work with different Atmel Flash memories. The program uses Atmel's 4M bit Flash (AT29C040, AT29LV040, AT29C040A, and AT29LV040A) as an example, but the software can be easily adapted to accommodate other device densities.

In order to program the Flash memory, the software must first determine which Flash device is being used. This is accomplished by first putting the device into the Software Product Identification mode (described in the "Product and Manufacturer ID" section of this application note). The program subsequently reads the Device ID Code and executes the 3-byte command sequence to return the Flash to the standard operating mode. Using the Device ID Code, the program then determines the appropriate sector size and write cycle time (t_{WC}) for the particular 4M bit Flash being used.

To demonstrate a sector write, the program proceeds to load the SRAM with "dummy" data. After the data has been loaded, the program transfers the data from the SRAM to a predefined sector (within one of the mapped 8K byte blocks) of the 4M bit Flash. After pausing the required write cycle time (t_{WC}), the sector that was just written is transferred back to the SRAM buffer.

Summary

Atmel's AT29 Flash memories are designed to allow all densities and device configurations to be programmed using the same programming algorithm. The user has to simply determine the Device ID Code and set the appropriate sector size and write cycle time. This operation need only be performed once provided the sector size and write cycle information is saved. If only one density or configuration will ever be used, then reading of the Device ID Code can be eliminated, and the sector size and write cycle information can be predefined in the software. The table at the end of this application note details the device information and the Device ID Codes for Atmel's AT29XXX Series of Flash PEROMs.

As demonstrated, programming Atmel's AT29 Flash is a simple process, similar to loading an SRAM. Architectural and circuit features within the devices minimize software and system overhead while simplifying programming procedures. Atmel's AT29 Flash memories require only about one-tenth of the typical software, buffer memory, and performance overhead of previous generation Flash, thus providing substantial system cost savings.

Atmel AT29 Flash Memories

Device	Memory Size	Device ID Code	Number of Sectors	Sector Size	Write Cycle Time (t_{WC})	Comments
AT29C256	32K x 8	DC	512	64 bytes	10 ms	
AT29LV256	32K x 8	BC	512	64 bytes	20 ms	
AT29C257	32K x 8	DC	512	64 bytes	10 ms	
AT29C512	64K x 8	5D	512	128 bytes	10 ms	
AT29LV512	64K x 8	3D	512	128 bytes	20 ms	
AT29C010A	128K x 8	D5	1024	128 bytes	10 ms	
AT29LV010A	128K x 8	35	1024	128 bytes	20 ms	
AT29BV010A	128K x 8	35	1024	128 bytes	20 ms	
AT29C1024	64K x 16	25	512	128-words	10 ms	
AT29LV1024	64K x 16	26	512	128-words	20 ms	
AT29C020	256K x 8	DA	1024	256 bytes	10 ms	
AT29LV020	256K x 8	BA	1024	256 bytes	20 ms	
AT29BV020	256K x 8	BA	1024	256 bytes	20 ms	
AT29C040	512K x 8	5B	1024	512 bytes	10 ms	Use AT29C040A for new designs
AT29LV040	512K x 8	3B	1024	512 bytes	20 ms	Use AT29LV040A for new designs
AT29BV040	512K x 8	3B	1024	512 bytes	20 ms	Use AT29BV040A for new designs
AT29C040A	512K x 8	A4	2048	256 bytes	10 ms	
AT29LV040A	512K x 8	C4	2048	256 bytes	20 ms	
AT29BV040A	512K x 8	C4	2048	256 bytes	20 ms	

```
/*****************************************************************/
/* This program demonstrates how a sector in one of the 512K X 8 */
/* variants can be programmed. The program first determines     */
/* exactly which device is available by reading the device ID.   */
/* A sector is then programed with data that is copied from an  */
/* SRAM buffer. After waiting for the programming cycle to      */
/* complete the data is copied back from the 29C040 to the SRAM */
/* buffer.                                                       */
/*                                                               */
/* The sector size and programming time are determined    */
/* by examining the device ID. The different 512K X 8 devices   */
/* have either a 256- or 512-byte sector size and a 10 mS or 20 */
/* mS tWC.                                                       */
/*****************************************************************/

/***********************/
/* COMPILER DIRECTIVES */
/***********************/
```

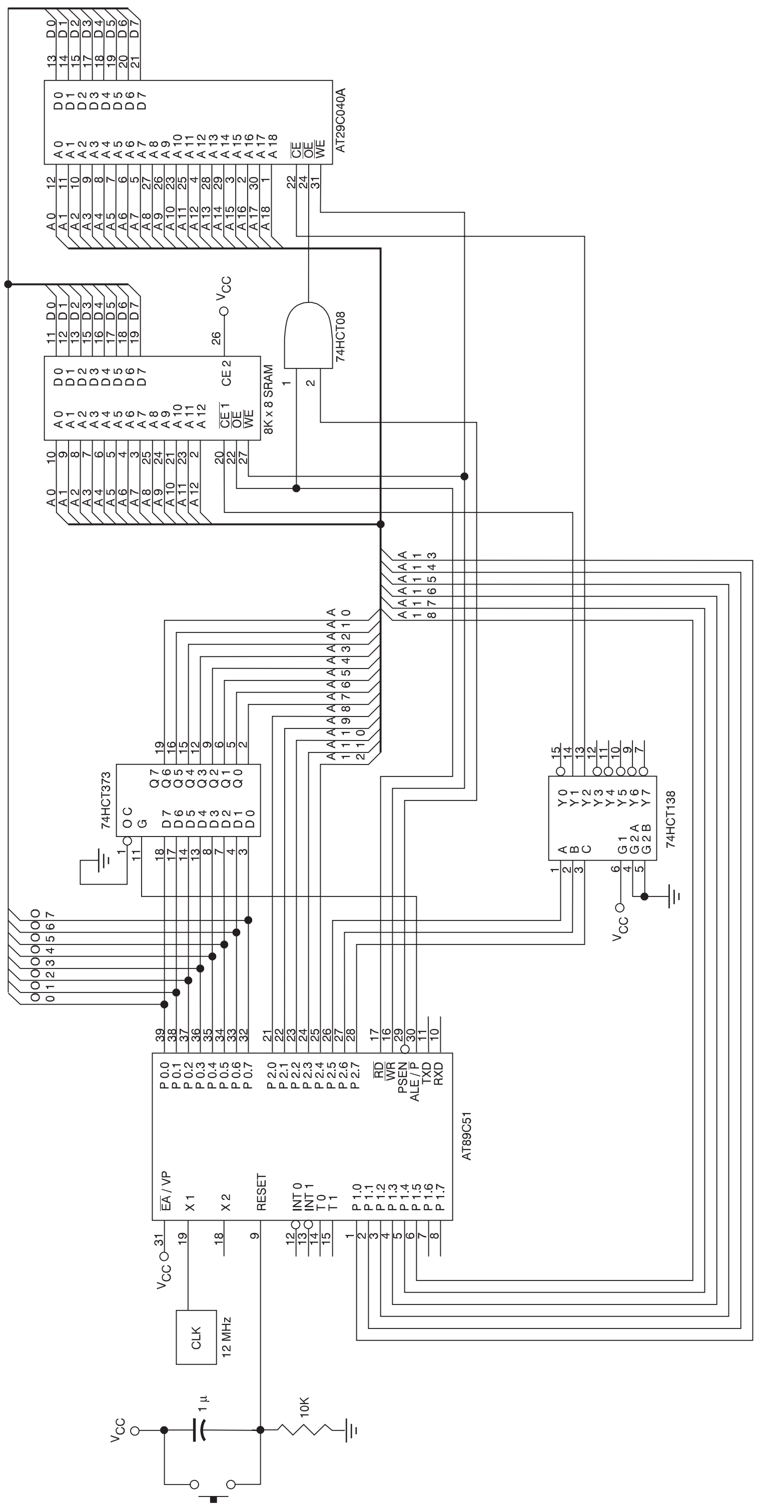
```
        .asm
        .linklist
        .symbols
        .endasm

#include        “c8051sr.h”

/********************/
/* GLOBAL VARIABLES */
/********************/

unsigned char part_id;                  /* DEVICE ID VALUE */
int sector_size;                        /* DEVICE SECTOR SIZE */
int twc;                                /* DEVICE PROGRAMMING TIME REQUIRED */
unsigned char data_buffer[512];         /* SRAM DATA BUFFER */
unsigned char block_number;             /* WHICH BLOCK TO PROGRAM */
unsigned int sector_address;            /* ADDRESS WITHIN SECTOR TO PROGRAM */
unsigned int address_pointer;           /* SCRATCH PAD ADDRESS REGISTER */
unsigned char temp_byte;                /* SCRATCH PAD DATA REGISTER */

/***********************/
/* SUPPORT SUBROUTINES */
/***********************/

/*******************************************************************/
/* DELAYMS performs a time delay.  The variable ticks indicates the */
/* length of the delay in mS.  This routine is dependant upon the   */
/* clock rate of the 89C51.  If a clock rate other than 12 MHz is   */
/* used the variable 'count' must be modified.                      */
/*******************************************************************/

void delayms(char ticks)

{
char count;

  for (ticks = ticks; ticks >= 0; ticks—)
  {
      for (count = 0; count <= 13; count++)
      {
      }
  }
}

/******************************************************************/
/* ENTER_ID_MODE is used to put the 29C040 into Software Product */
/* Identification mode.  The three step sequence is performed in */
/* assembly because of tBLC requirements of the 29C040.          */
/******************************************************************/
```

```
void enter_id_mode()

{
  .asm
  mov   a,#05h
  mov   p1,a
  mov   dptr,#4555h
  mov   a,#aah
  movx  @dptr,a                          ;write AAh to address 05555h
  mov   a,#02h
  mov   p1,a
  mov   dptr,#4aaah
  mov   a,#55h
  movx  @dptr,a                          ;write 55h to address 02AAAh
  mov   a,#05h
  mov   p1,a
  mov   dptr,#4555h
  mov   a,#90h
  movx  @dptr,a                          ;write 90h to address 05555h
  .endasm
}

/*********************************************************************/
/* LEAVE_ID_MODE is used to remove the 29C040 from Software Product */
/* Identification mode.  The three step sequence is performed in    */
/* assembly because of tBLC requirements of the 29C040.             */
/*********************************************************************/

void leave_id_mode()


{
  .asm
  mov   a,#05h
  mov   p1,a
  mov   dptr,#4555h
  mov   a,#aah
  movx  @dptr,a                          ;write AAh to address 05555h
  mov   a,#02h
  mov   p1,a
  mov   dptr,#4aaah
  mov   a,#55h
  movx  @dptr,a                          ;write 55h to address 02AAAh
  mov   a,#05h
  mov   p1,a
  mov   dptr,#4555h
  mov   a,#f0h
  movx  @dptr,a                          ;write F0h to address 05555h
  .endasm
```

```
}

/*********************************************************************/
/* GET_ID is used read the value at location 00001 of the 20C040.    */
/* The value read from the device is returned to the calling routine */
/*********************************************************************/

unsigned char get_id()

{
  P1 = 0x00;                            /* read from block 00h */
  .asm
  mov   dptr,#4001h                     ;read from address 00001h
  movx  a,@dptr                         ; (flash offset = 4000h)
  .endasm
  return(A);                            /* return data value */
}

/******************************************************************/
/* GET_PART_ID determines the device ID of the 29C040 being used. */
/* The ID value is returned to the calling routine                */
/******************************************************************/

unsigned char get_part_id()

{
unsigned char part_id;

  enter_id_mode();                      /* enter Identification mode */
  delayms(20);                          /* delay 20mS */
  part_id = get_id();                   /* read device ID from address 1 */
  leave_id_mode();                      /* exit from Identification mode */
  delayms(20);                          /* delay 20mS */
  return(part_id);                      /* return device ID value */
}

/*****************************************************************/
/* SET_PARAMETERS is used to define what sector size and write   */
/* cycle is required for the particular 29C040 being used        */
/* The sector size is stored in the global variable SECTOR_SIZE, */
/* and the programming time is stored in the global variable TWC.*/
/*****************************************************************/

void set_parameters(unsigned char part_id)

{
  switch(part_id)
  {
    case 0x5b : sector_size = 512;      /* is the device a 29C040 */
                twc = 10;
                break;
    case 0xa4 : sector_size = 256;      /* is the device a 29C040A */
                twc = 10;
                break;
    case 0x3b : sector_size = 512;      /* is the device a 29LV040 */
                twc = 20;
                break;
    case 0xc4 : sector_size = 256;      /* is the device a 29LV040A */
                twc = 20;
                break;
    default   : sector_size = 0;        /* variables default to 0 */
                twc = 0;
  }
}

/*****************************************************************/
/* DUMMY_BUFFER_LOAD simply loads the SRAM buffer with the value */
/* passed in IN_VALUE.  Although the SRAM buffer has 512 bytes,  */
/* only the number of bytes required to fill a sector are loaded.*/
/*****************************************************************/

void dummy_buffer_load(char in_value)

{
int count;

  for (count = 0; count <= sector_size; count++)
  {
    data_buffer[count] = in_value;
  }
}

/*****************************************************************/
/* WRITE_SECTOR copies data from the SRAM buffer into the sector */
/* specified in the 29C040.  After loading the sector the routine*/
/* paused the required tWC for the programming cycle to complete.*/
/*****************************************************************/

void write_sector()

{
  .asm
        mov     a,#05h                  ;perform 3 step SDP sequence
        mov     p1,a
        mov     dptr,#4555h
        mov     a,#aah
        movx    @dptr,a                 ;write AAh to address 05555h
        mov     a,#02h
        mov     p1,a
        mov     dptr,#4aaah
        mov     a,#55h
        movx    @dptr,a                 ;write 55h to address 02AAAh
        mov     a,#05h
        mov     p1,a
        mov     dptr,#4555h
        mov     a,#A0h
        movx    @dptr,a                 ;write A0h to address 05555h
        mov     dptr,#_block_number
        movx    a,@dptr
        mov     P1,a                    ;set up block address
        mov     dptr,#_sector_size      ;load sector size
        movx    a,@dptr
        mov     r0,a
        inc     dptr
        movx    a,@dptr
        mov     r1,a
        mov     dptr,#_sector_address   ;load first sector address to write
        movx    a,@dptr
        add     a,#40h
        mov     r3,a
        inc     dptr
        movx    a,@dptr
        mov     r2,a
        mov     dptr,#_data_buffer      ;load pointer to data_buffer
nextwr: movx    a,@dptr                 ;load data to write to 29C040
        inc     dptr                    ;increment data_buffer pointer
        push    dpl
        push    dph
        mov     dpl,r2
        mov     dph,r3
        movx    @dptr,a                 ;write data to 29C040
        inc     dptr                    ;increment flash address pointer
        mov     r2,dpl
        mov     r3,dph
        pop     dph
        pop     dpl
        djnz    r1,nextwr               ;decrement byte counter
        djnz    r0,nextwr               ; loop until sector has been loaded
  .endasm
  delayms(twc);                         /* delay for the programming cycle */
}

/*********************************************************************/
/* READ_SECTOR copies a sector from the 29C040 into the SRAM buffer */
/* Either 256 or 512 bytes are transfered depending on the size of  */
/* the sector.                                                       */
/*********************************************************************/

void read_sector()
```

```
{
unsigned int count;

  P1 = block_number;                               /* initial block # */
  address_pointer = sector_address + 0x4000;       /*create address pointer*/
  for (count = 0; count < sector_size; count++)    /*transfer sector to SRAM*/
  {
    .asm
    mov         dptr,#_address_pointer  ;load address pointer's address
    movx        a,@dptr                 ;load address pointer high byte
    mov         b,a
    inc         dptr
    movx        a,@dptr                 ;load address pointer low byte
    mov         dpl,a
    mov         dph,b
    movx        a,@dptr                 ;read data from 29C040
    mov         dptr,#_temp_byte
    movx        @dptr,a                 ;store data from 29C040 into temp_byte
    .endasm
    data_buffer[count] = temp_byte;             /*place data into SRAM*/
    address_pointer = address_pointer + 1;      /*increment address pointer*/
  }
}

/************/
/* MAINLINE */
/************/

main()

{
  part_id = get_part_id();              /* GET PART ID */
  set_parameters(part_id);              /* DETERMINE WRITE PARAMETERS */
  dummy_buffer_load(0x55);              /* LOAD SRAM BUFFER WITH DUMMY DATA */
  block_number = 0x1f;                  /* SPECIFY BLOCK NUMBER TO WRITE */
  sector_address = 0x0400;              /* SPECIFY ADDRESS WITHIN BLOCK */
  write_sector();                       /* COPY SRAM BUFFER TO 29C040 */
  read_sector();                        /* COPY 29C040 SECTOR TO SRAM */
}
```

Atmel AT29C040A Demo Circuit



Note: If the Flash is to be used as external program memory, then pin 31 (EA/ Vpp) of the AT89C51 cannot be connected to V_{CC}.