Keywords: network microcontroller, flash memory

APPLICATION NOTE 3478

Flash Memory Selection

Mar 14, 2005

Abstract: This application note describes those requirements and gives examples of memory chips that are compatible with the DS80C400/DS80C410/DS80C411. Software code is presented that can be used to erase and program flash memory that is electrically compatible, but not ROM-loader software compatible.

Note: Maxim Integrated does not have information regarding the availability of devices mentioned in this application note or the availability of alternate devices. The user assumes full responsibility for certifying the suitability, including electrical specifications and availability, of a particular device in their application.

Requirements

Electrical

The DS80C400/DS80C410/DS80C411 require 3.3V byte-wide (x8) flash memory.

Access Time

The DS80C400/DS80C410/DS80C411 execution speed (CPU speed) is the crystal speed, times the clock multiplier. (The clock multiplier is not changed by the ROM loader, but can be set by application software.) The clock period t_{clcl} is calculated as 1/(CPU speed).

Execution out of flash is only possible when the flash memory meets the DS80C400/DS80C410/DS80C411 timing requirements, as listed in the respective data sheets.

Meeting the following two timing requirements, for example, is an easy way to limit the field of possible parts (Before committing to a design, all other parameters should also be verified.):

- "PSEN low to valid instruction in" (t_{pliv} = 2 t_{clcl} - 25ns): The flash *output enable to output delay time*, t_{glqv}, must be faster than the DS80C400/DS80C410/DS80C411 t_{pliv}. For example, AM29LV081B-70 has a t_{glqv} of 30ns and a 30MHz DS80C400 (t_{clcl} = 33.3ns) has a t_{pliv} of 41.6ns.
- "Address to valid data in" (t_{aviv1} = 3 t_{clcl} - 19ns): The flash *access time*, t_{avav}, must be faster than this number. For example, when running at 25MHz, t_{clcl} is 40ns and t_{avav} must therefore be smaller than 101ns.

Table 1 shows examples of execution speeds for the DS80C400 and the AM29LV081 flash.

Table 1. CPU Speed Limits Determined by Flash Access Time, t_{avav}	
Rated Flash Speed	**Max CPU Speed**
55ns	40MHz
70ns	33MHz
90ns	27.5MHz
120ns	21MHz
150ns	17MHz

The CPU crystal should be a good serial baud-rate generator. Also, operation on a 100Mb network is only possible when CPU speed is higher than approximately 27MHz.

ROM Loader

The DS80C400/DS80C410/DS80C411 provide a built-in ROM that supports loading SRAM and flash through the serial port and through the network. Note that the maximum loader serial-port baud rate is limited by the crystal speed, e.g., 115200bps requires at least 20MHz.

Loader Algorithm

The ROM loader has built-in algorithms for flash sector erase and flash byte programming. The loader assumes a uniform sector size of 64KB and uses the programming algorithms shown in **Table 2**.

Table 2. ROM Loader Flash Algorithms												
Operation	Cycle 1		Cycle 2		Cycle 3		Cycle 4		Cycle 5		Cycle 6	
	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Program	0555	AA	02AA	55	0555	A0	PA	PD				
Sector Erase	0555	AA	02AA	55	0555	80	0555	AA	02AA	55	0000	30

Note: Table 2 lists DS80C4XX address lines A15–A0 only. A2–A16 are always the sector address. All numbers in hexadecimal.

Flash Address Lines

Most flash memories ignore the high-order address bits (DS80C400/DS80C410/DS80C411 address bits A12 and higher) when matching the "magic" address. To initiate a programming sequence, for example, it does not matter whether the first address is 5555h or 555h.

Some flash memories also ignore the lowest address bit (called A-1 on word/byte selectable memories, else A0). For example, it does not matter whether the address is 555h or 554h.

This means that, while some memories appear incompatible at first glance, full compatibility with the DS80C400/DS80C410/DS80C411 loader can be achieved by connecting the flash address lines, as shown in **Table 3**, columns (B) or (C). All other flash memories (without address line A-1, and where not listed otherwise in **Table 4**) should be connected as shown in column (A).

Table 3. Address Line Connections			
DS80C4xx Address Line	(A) Standard	(B) With A-1	(C) See Text
A0	A0	A0	A1
A1	A1	A1	A2
A2	A2	A2	A3
A3	A3	A3	A4
A4	A4	A4	A5
A5	A5	A5	A6
A6	A6	A6	A7
A7	A7	A7	A8
A8	A8	A8	A9
A9	A9	A9	A10
A10	A10	A10	A11
A11	A11	A-1	A0
A12	A12	A11	A12
A13	A13	A12	A13
A14	A14	A13	A14
...	...	...	...
A(n)	A(n)	A(n-1)	A(n)

Loader Support Levels

A *fully supported* flash memory works "out of the box" with the ROM loader.

A *partially supported* flash memory needs application software help to erase flash sectors, but is programmable by the ROM. Most flash memories in this class have a boot sector that can be programmed, but not fully erased by the ROM loader.

Table 4 shows a selection of flash memories and their respective ROM loader compatibilities. This table may be of assistance in determining the suitability of alternate devices.

Table 4. Selection of 3V Flash Memories (x8)					
Vendor (ID)	Device (ID)	Size (Byte)	Blocks	ROM Support	Notes
Atmel (1F)	AT49BV001A (4,5)	128K	B	Prog	ROM can program, but not erase part of the boot sector.
Macronix (C2)	MX29LV081 (38)	1M	U	Full	Data sheet shows incompatibility for sector erase, but sector erase does work in practice.
	MX29LV017B (C8)	2M	U	Full	—
	MX29LV033A (A3)	4M	U	Full	—
Spansion (01) (AMD/Fujitsu)	AM29LV200B (3B,BF)	256K	B	Prog	Requires address line connection (B). ROM can program, but not erase part of the boot sector.
	AM29LV004B (B5,B6)	512K	B	Prog	ROM can program, but not erase part of the boot sector.
	AM29LV040B (4F)	512K	U	Full	—
	AM29LV400B (B9,BA)	512K	B	Prog	Requires address line connection (B). ROM can program, but not erase part of the boot sector.
	AM29LV008B (37,3E)	1M	B	Prog	ROM can program, but not erase part of the boot sector.
	AM29LV081B (38)	1M	U	Full	—
	AM29LV800 (5B,DA)	1M	B	Prog	Requires address line connection (B). ROM can program, but not erase part of the boot sector.
	AM29LV017D (4F)	2M	U	Full	—
	AM29LV116D (4C,C7)	2M	B	Prog	ROM can program, but not erase part of the boot sector.
	AM29LV160B/D or S29AL016D (49,C4)	2M	B	Prog	Requires address line connection (B). ROM can program, but not erase part of the boot sector.
	AM29LV033C or S29AL032D Model 0 (A3)	4M	U	Full	—
	AM29LV320D or S29AL032D Model 3/4 (F6,F9)	4M	B	Prog	Requires address line connection (B). ROM can program, but not erase part of the boot sector.
SST (BF)	SST39VF1681/2 (C8)	2M	U	Full	Requires address line connection (C).
ST Micro-electronics (20)	M29W004B (EA,EB)	512K	B	Prog	ROM can program, but not erase part of the boot sector.
	M29W040B (E3)	512K	U	Prog	Should be fully supported according to data sheet, but sector erase algorithm does not work in practice.
	M29W008D (D2,DC)	1M	B	Prog	ROM can program, but not erase part of the boot sector.

Legend:
Blocks Column: B = Device has boot block, U = Uniform 64KB block size, O = Uniform block size other than 64KB Vendor and Device IDs in hexadecimal.
ROM Support Column: Full Support, ROM can Program, or No Support

Flash Software

If a particular memory is not fully supported by the ROM loader, it can be erased/programmed by software loaded into SRAM. **Listing 1** shows software that erases a flash chip. **Listing 2** shows how to program a flash sector by copying 64KB from SRAM into flash. These example programs typically have to be modified to match the programming algorithm of a particular flash.

The example programs load into the first 64KB of SRAM ("bank 0"). When using the Microcontroller Tool Kit (MTK), turn off the "Clear Heap" option. When using the ROM loader to *program* partially supported flash memories (after running the chip erase program), turn off the "AutoZap" option in the Java Development Kit or the "Erase Flash" option in MTK.

Listing 1. Erase Flash Chip (chiperase.asm)

```
;*****************************************************************************
;* Copyright (C) 2013 Maxim Integrated, All Rights Reserved.
;*
;* Permission is hereby granted, free of charge, to any person obtaining a
;* copy of this software and associated documentation files (the "Software"),
;* to deal in the Software without restriction, including without limitation
;* the rights to use, copy, modify, merge, publish, distribute, sublicense,
;* and/or sell copies of the Software, and to permit persons to whom the
;* Software is furnished to do so, subject to the following conditions:
;*
;* The above copyright notice and this permission notice shall be included
;* in all copies or substantial portions of the Software.
;*
;* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
;* OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
;* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
;* IN NO EVENT SHALL MAXIM INTEGRATED BE LIABLE FOR ANY CLAIM, DAMAGES
;* OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
;* ARISING FROM, OUT OF, OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
;* OTHER DEALINGS IN THE SOFTWARE.
;*
;* Except as contained in this notice, the name of Maxim Integrated
;* shall not be used except as stated in the Maxim Integrated
;* Branding Policy.
;*****************************************************************************
;
; This program demonstrates how to erase a flash memory chip that is not
; supported by the ROM loader of the networked microcontrollers.
;
; To build this program, run
;    macro chiperase.asm
;    a390 -d -l chiperase.mpp
;
; To execute, load into SRAM and use loader commands 'B0', 'X0'.
;
; Note: When using MTK to load this code, be sure to disable the "Clear Heap"
; option.
;

; The flash memory we want to erase is connected to CE2\, or address 400000h,
; since the ROM uses 2MB per chip enable.

FLASH_ADDRESS EQU 400000h

; Different flash memories use different addresses to "tickle" flash programming/
; erase operation.

; Erase ST M29W040
; FLASH_TICKLE0 EQU (FLASH_ADDRESS or 555h)
; FLASH_TICKLE1 EQU (FLASH_ADDRESS or 2aah)

; Erase AM29LV200BT
FLASH_TICKLE0 EQU (FLASH_ADDRESS or 0aaah)
```

```
FLASH_TICKLE1 EQU (FLASH_ADDRESS or 555h)


        org 000000h

start:
        ; The ROM enables 24-bit mode and disables interrupts.
        ; No other initialization is necessary.

        ; Start flash chip erase

        ; 1st Cycle
        mov dptr, #FLASH_TICKLE0
        mov a, #0aah
        movx @dptr, a

        ; 2nd Cycle
        mov dptr, #FLASH_TICKLE1
        mov a, #55h
        movx @dptr, a

        ; 3rd Cycle
        mov dptr, #FLASH_TICKLE0
        mov a, #80h
        movx @dptr, a

        ; 4th Cycle
        mov dptr, #FLASH_TICKLE0
        mov a, #0aah
        movx @dptr, a

        ; 5th Cycle
        mov dptr, #FLASH_TICKLE1
        mov a, #55h
        movx @dptr, a

        ; 6th Cycle
        mov dptr, #FLASH_TICKLE0
        mov a, #10h
        movx @dptr, a

        ; Wait for operation to complete
        mov dptr, #FLASH_ADDRESS
wait:
        movx a, @dptr
        cjne a, #0ffh, wait

        ; Reset
        mov dptr, #FLASH_ADDRESS
        mov a, #0f0h
        movx @dptr, a

        ; Print success message
        mov a, #'D'
        acall printchar
        mov a, #'O'
        acall printchar
        mov a, #'N'
        acall printchar
        mov a, #'E'
        acall printchar

        ; Done!
        sjmp $
```

```
        ; Serial port 0 is initialized by the loader. Printing
        ; a character is therefore trivial.
tix bit scon.1
printchar:
        jnb  tix, $
        clr tix
        mov sbuf, a
        jnb  tix, $
        ret

        end
```

Listing 2. Programming Unsupported Flash (flashprogram.asm)

```
;******************************************************************************
;* Copyright (C) 2013 Maxim Integrated, All Rights Reserved.
;*
;* Permission is hereby granted, free of charge, to any person obtaining a
;* copy of this software and associated documentation files (the "Software"),
;* to deal in the Software without restriction, including without limitation
;* the rights to use, copy, modify, merge, publish, distribute, sublicense,
;* and/or sell copies of the Software, and to permit persons to whom the
;* Software is furnished to do so, subject to the following conditions:
;*
;* The above copyright notice and this permission notice shall be included
;* in all copies or substantial portions of the Software.
;*
;* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
;* OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
;* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
;* IN NO EVENT SHALL MAXIM INTEGRATED BE LIABLE FOR ANY CLAIM, DAMAGES
;* OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
;* ARISING FROM, OUT OF, OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
;* OTHER DEALINGS IN THE SOFTWARE.
;*
;* Except as contained in this notice, the name of Maxim Integrated
;* shall not be used except as stated in the Maxim Integrated
;* Branding Policy.
;******************************************************************************
;
; This program demonstrates how to erase a flash memory chip that is not
; supported by the ROM loader of the networked microcontrollers.
;
; To build this program, run
;    macro flashprogram.asm
;    a390 -d -l flashprogram.mpp
;
; To execute, first load the data you wish to program into SRAM memory in
; bank 1 (010000h - 01ffffh). Then, load this code into SRAM and use
; the loader commands 'B0', 'X100'.
;
; Note: When using MTK to load this code, be sure to disable the "Clear Heap"
; option.
;

; The flash memory we want to program is connected to CE2\, or address 400000h,
; since the ROM uses 2MB per chip enable.

FLASH_ADDRESS EQU 400000h

; Different flash memories use different addresses to "tickle" flash programming/
; erase operation.

; Byte Program AM29LV200BT
```

```
FLASH_TICKLE0 EQU (FLASH_ADDRESS or 0aaah)
FLASH_TICKLE1 EQU (FLASH_ADDRESS or 555h)

; Address of programming buffer in SRAM
PROG_BUFFER EQU 010000h

; We want to program a whole 64KB block
PROG_SIZE EQU 10000h

; This example program uses three datapointers
dps equ 086h

        org 000100h

start:
        ; The ROM enables 24-bit mode and disables interrupts.
        ; No other initialization is necessary.

        ; Make sure r2 is in register bank 0
        mov psw, #0

        ; Number of bytes to program
        mov r1, #high(PROG_SIZE)
        mov r0, #low(PROG_SIZE)

        ; Source
        mov dps, #1
        mov dptr, #PROG_BUFFER

        ; Destination
        mov dps, #8
        mov dptr, #FLASH_ADDRESS

loop:
        ; Start flash chip program for this byte

        ; Get source byte
        mov dps, #1
        movx a, @dptr
        inc dptr

        ; Save byte we wish to program
        mov r2, a

        ; No need to write the same byte again
        ; (also prevents writing of 0ffh)
        mov dps, #8
        movx a, @dptr
        xrl a, r2
        jz next

        ; Select dptr0
        mov dps, #0

        ; 1st Cycle
        mov dptr, #FLASH_TICKLE0
        mov a, #0aah
        movx @dptr, a

        ; 2nd Cycle
        mov dptr, #FLASH_TICKLE1
        mov a, #55h
        movx @dptr, a

        ; 3rd Cycle
```

```
        mov dptr, #FLASH_TICKLE0
        mov a, #0a0h
        movx @dptr, a

        ; 4th Cycle: Put destination byte
        mov dps, #8
        mov a, r2
        movx @dptr, a

        ; Wait for operation to complete
wait:
        movx a, @dptr
        cjne a, 2, wait

next:
        mov dps, #8
        inc dptr

        djnz r0, loop

        ; Display progress indicator
        mov a, #'.'
        acall printchar

        djnz r1, loop

        ; Reset
        mov dptr, #FLASH_ADDRESS
        mov a, #0f0h
        movx @dptr, a

        ; Print success message
        mov a, #13
        acall printchar
        mov a, #10
        acall printchar
        mov a, #'D'
        acall printchar
        mov a, #'O'
        acall printchar
        mov a, #'N'
        acall printchar
        mov a, #'E'
        acall printchar

        ; Done!
        sjmp $


        ; Serial port 0 is initialized by the loader. Printing
        ; a character is therefore trivial.
tix bit scon.1
printchar:
        jnb  tix, $
        clr tix
        mov sbuf, a
        jnb  tix, $
        ret

        end
```

Flash Identification

Listing 3 is a C program that decodes the vendor and device for a number of flash memory chips using their Autoselect capabilities.

Listing 3. Flash Identification (identify.c)

```c
/* -------------------------------------------------------------------------
 *  Copyright (C) 2013 Maxim Integrated, All Rights Reserved.
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a
 *  copy of this software and associated documentation files (the "Software"),
 *  to deal in the Software without restriction, including without limitation
 *  the rights to use, copy, modify, merge, publish, distribute, sublicense,
 *  and/or sell copies of the Software, and to permit persons to whom the
 *  Software is furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included
 *  in all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 *  OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 *  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 *  IN NO EVENT SHALL MAXIM INTEGRATED BE LIABLE FOR ANY CLAIM, DAMAGES
 *  OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
 *  ARISING FROM, OUT OF, OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 *  OTHER DEALINGS IN THE SOFTWARE.
 *
 *  Except as contained in this notice, the name of Maxim Integrated
 *  shall not be used except as stated in the Maxim Integrated
 *  Branding Policy.
 * -------------------------------------------------------------------------
 */

/*
 * Identify - Tries to Identify Flash Memory Make/Model.
 * Run from bank 20 (using loader commands B20, X)
 */

#include <stdio.h>
#include <reg400.h>

void main()
{
  unsigned char vendor, device, ce;

  puts("DS80C400/DS80C410/DS80C411 Flash Memory Identification");

  do {

    printf("\nIdentify flash at which chip enable? ");
    do {
      putchar(ce = _getkey());
      if ((ce < '0') || (ce > '7'))
        printf(" 0-7> ");
    } while ((ce < '0') || (ce > '7'));
    puts("\n");

    ce -= '0';
    ce <<= 5; // 2MB per chip enable

    AP = ce;
#pragma asm
    /* Tickle Flash Memory */
    mov dptr, #0x5555
    mov dpx, ap
    mov a, #0xaa
    movx @dptr, a

    mov dptr, #0xaaaa
```

```
    mov dpx, ap
    mov a, #0x55
    movx @dptr, a

    /* Read ID Command */
    mov dptr, #0x5555
    mov dpx, ap
    mov a, #0x90
    movx @dptr, a

    /* Read Manufacturer ID */
    mov dptr, #0
    mov dpx, ap
    movx a, @dptr
#pragma endasm
    vendor = ACC;

#pragma asm
    /* Reset Flash */
    mov dptr, #0
    mov dpx, ap
    mov a, #0xf0
    movx @dptr, a
    mov a, #0xff
    movx @dptr, a
#pragma endasm

#pragma asm
    /* Tickle Flash Memory */
    mov dptr, #0x5555
    mov dpx, ap
    mov a, #0xaa
    movx @dptr, a

    mov dptr, #0xaaaa
    mov dpx, ap
    mov a, #0x55
    movx @dptr, a

    /* Read ID Command */
    mov dptr, #0x5555
    mov dpx, ap
    mov a, #0x90
    movx @dptr, a

    /* Read Manufacturer ID */
    mov dptr, #0x01
    mov dpx, ap
    movx a, @dptr
#pragma endasm
    device = ACC;

#pragma asm
    /* Reset Flash */
    mov dptr, #0
    mov dpx, ap
    mov a, #0xf0
    movx @dptr, a
    mov a, #0xff
    movx @dptr, a
#pragma endasm

    printf("Flash memory at CE%bu: Vendor ID %02bX, Device ID %02bX.\n --> ", ce >> 5, vendor, device);

    switch (vendor) {
```

```c
case 0x01: printf("Spansion AM");
  switch (device) {
    case 0x37: puts("29LV008 Top Boot"); break;
    case 0x38: puts("29LV081"); break;
    case 0x3b: puts("29LV200 Top Boot"); break;
    case 0x3e: puts("29LV008 Bottom Boot"); break;
    case 0x49: puts("29LV160 Bottom Boot"); break;
    case 0x4c: puts("29LV116 Bottom Boot"); break;
    case 0x4f: puts("29LV040"); break;
    case 0x5b: puts("29LV800 Bottom Boot"); break;
    case 0xa3: puts("29LV033"); break;
    case 0xb5: puts("29LV004 Top Boot"); break;
    case 0xb6: puts("29LV004 Bottom Boot"); break;
    case 0xb9: puts("29LV400 Top Boot"); break;
    case 0xba: puts("29LV400 Bottom Boot"); break;
    case 0xbf: puts("29LV200 Bottom Boot"); break;
    case 0xc4: puts("29LV160 Top Boot"); break;
    case 0xc7: puts("29LV116 Top Boot"); break;
    case 0xc8: puts("29LV017"); break;
    case 0xda: puts("29LV800 Top Boot"); break;
    case 0xf6: puts("29LV320 Top Boot"); break;
    case 0xf9: puts("29LV320 Bottom Boot"); break;
    default:   puts(" ????"); break;
  }
break;
case 0x1f: printf("Atmel AT");
  switch (device) {
    case 0x21: puts("49BV/LV008 T"); break;
    case 0x22: puts("49BV/LV008"); break;
    case 0xeb: puts("49LL080"); break;
    default:   puts(" ????"); break;
  }
break;
case 0x20: printf("ST M");
  switch (device) {
    case 0x5b: puts("29W800 Bottom Boot"); break;
    case 0xd2: puts("29W008 Top Boot"); break;
    case 0xd7: puts("29W800 Top Boot"); break;
    case 0xdc: puts("29W008 Bottom Boot"); break;
    case 0xe3: puts("29W040"); break;
    case 0xea: puts("29W004 Top Boot"); break;
    case 0xeb: puts("29W004 Bottom Boot"); break;
    default:   puts(" ????"); break;
  }
break;
case 0x89: printf("Intel or Sharp LH or "); // fall through
case 0x2c: printf("Micron MT");
  switch (device) {
    case 0x16: puts("28F320J3"); break;
    case 0x70: puts("28F004B3/28F400B3 Top Boot"); break;
    case 0x71: puts("28F004B3/28F400B3 Bottom Boot"); break;
    case 0x9c: puts("28F008B3/28F800B3 Top Boot"); break;
    case 0x9d: puts("28F008B3/28F800B3 Bottom Boot"); break;
    case 0xaa: puts("28F016SC"); break;
    default:   puts(" ????"); break;
  }
break;
case 0xbf: printf("SST SST");
  switch (device) {
    case 0xc8: puts("39VF1681"); break;
    case 0xc9: puts("39VF1682"); break;
    case 0xd4: puts("39LF/VF512"); break;
    case 0xd5: puts("39LF/VF010"); break;
    case 0xd6: puts("39LF/VF020"); break;
    case 0xd7: puts("39LF/VF040"); break;
```

```
        case 0xd8: puts("39LF/VF080"); break;
        default:   puts(" ????"); break;
      }
      break;
    case 0x02:
    case 0xc2: printf("Macronix MX");
      switch (device) {
        case 0x38: puts("29LV081"); break;
        case 0xa3: puts("29LV033"); break;
        case 0xc8: puts("29LV017"); break;
        default:   puts(" ????"); break;
      }
      break;
    case 0xda: printf("Winbond W");
      switch (device) {
        case 0xd6: puts("39L040"); break;
        default:   puts(" ????"); break;
      }
      break;
    default:   puts("Unknown vendor/unknown device");
      break;
    }
  } while (1);
}
```

https://support.maximintegrated.com/micro

Related Parts		
DS80C400	Network Microcontroller	Free Samples
DS80C410	Network Microcontrollers with Ethernet and CAN	Free Samples
DS80C411	Network Microcontrollers with Ethernet and CAN	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact