APPLICATION NOTE

Using Flash memory with the XA

AN97019

Philips
Semiconductors

PHILIPS

Using Flash memory with the XA

Application Note AN97019

Abstract

This note examines the various options involved in connecting Flash memories to the XA. Special attention is payed to the possibility of dynamic software updating (re-loading the Flash).

APPLICATION NOTE

Using Flash memory with the XA

AN97019

Author:

Paul Seerden
Systems Laboratory Eindhoven,
The Netherlands

Keywords

Microcontrollers 16 bit, XA, Flash Memory

Number of pages: 24

Date: 97-03-14

Using Flash memory with the XA

Application Note AN97019

Summary

This report is a guide for designers who consider an XA design with external Flash memory. Most of the problems / aspects that they meet are brought to light. Pro's and cont's of different design choises are dealt with. Hardware interface examples as well as software routines are illustrated. The focus is on applications with the possibility of dynamicly updating the software (re-loading the flash).

CONTENTS

1. INTRODUCTION

The Philips Semiconductors “P51XA” (Extended Architecture) is a 16-bit microcontroller family designed for applications needing high performance and high integration. A typical XA system can be built using external Flash memory for program storage. Using Flash memory provides an advantage over traditional non-volatile memories. Unlike EPROMs, Flash devices can be programmed in-system ! However, connection of a Flash device to a microcontroller like the XA is not straightforward. This application note will discuss how to achieve an efficient XA - Flash memory interface.

Chapter 2 describes hardware interfacing examples of XA systems without external data memory (so, they use only the internal XA on-chip data memory).

Chapter 3 describes an application example of an XA system with additional external data memory.

Chapter 4 describes a software driver to (re)load the Flash memory contents. The driver is written in C and for loading the Flash memory the Intel-hex file format is used. Serial downloading of the hex file is done using UART0 of the XA-G3.

1.1 What is Flash memory again ?

If the write-enable input of Flash parts is left out of consideration, then this type of memory is in fact the same as a normal EPROM. The real difference is the simplicity of data storage and erasure of the memory. Unlike EPROM’s , Flash parts can be erased and programmed in-system. Flash devices are able to receive commands, like chip erase and program, by an exact defined sequence of instructions. That makes accidentally erasing the Flash almost impossible. The different instruction command codes are shown below in table 1.

TABLE 1 Flash instructions (AMD 29F010)

Command	First WR Cycle		Second WR cycle		Third WR cycle		Fourth RD/ WR cycle		Fifth WR cycle		Sixth WR cycle	
	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Read / Reset	5555	AA	2AAA	55	5555	F0	RA	RD				
Autoselect	5555	AA	2AAA	55	5555	90						
Byte Program	5555	AA	2AAA	55	5555	A0	PA	PD				
Chip Erase	5555	AA	2AAA	55	5555	80	5555	AA	2AAA	55	5555	10
Sector Erase	5555	AA	2AAA	55	5555	80	5555	AA	2AAA	55	SA	30

Notes:

Addressbits A15...A18 are don’t care for all address commands except for the Program Address (PA) and Sector Address (SA)

RA = Address of the memory location to read

PA = Address of the memory location to program (at falling edge on WE input)

SA = Address of the sector to erase

RD = Contents of memory location RA

PD = Contents of memory location PA (at falling edge on WE input)

Read / Reset : This command puts the Flash in the read/reset state, and enables the Flash for data reads.

Autoselect : Used to get the device type and manufacturer code.

Byte Program : Used to write bytes to the flash.

Chip Erase : Will program and verify the entire memory for an all zero pattern.

Sector Erase : Flash memory is divided into sectors. This command erases multiple sectors concurrently.

1.2 To take into consideration

- Please realize that it is impossible to execute code from a normal Flash memory part while programming it. So, the program code that reloads the Flash needs to be located in a separate memory area/component.
- Flash interface timing versus XA frequency. In the design examples of chapters 2 and 3 a possible need for wait states is not taken into account. The software example is tested on a 20Mhz XA-G3 system (slowest WR cycle is 100ns) with the use of 90ns Flash parts.
- The software (chapter 4) to serially download (UART0 used) the Intel-hex file does NOT use flowcontrol (like RTS/CTS or xon/xoff). So, the host (sending the Intel-hex file) needs to send the data at a rate the XA software can handle.
- All given design examples in this report are with an XA-G3 8-bit wide databus mode. Please realize that for 16-bit systems the A0 line becomes the WRH signal. This means that the addresses, used by the driver software, to which the commands are written to, must be changed (see chapter 4). Also, in this case, words in stead of bytes must be written to both (or all) the Flash parts.
- If the Flash memory is mapped at address 0 in data memory space (to enable writing it) realize that the instruction MOVX is needed to write (and read) the lower 512 bytes of the flash. If MOVX is not used the internal RAM of the XA is addressed.
- All the examples use 128 kBytes Flash parts (29F010), so only address lines 0 to 16 are connected. If a system needs to be prepared for larger flash memory parts then address lines A17 and A18 can be connected to pin 1 and pin 30 of the 29F010 part.

This application note (with C source files) is available for downloading from the Philips Bulletin Board Systems and from the world wide web. It is packed in the self extracting PC DOS file: XAFLASH.EXE.

- North American Bulletin Board, telephone number: (800) 451 6644 (in the US) or (408) 991 2406
- European Bulletin Board, telephone number: +31 40 272 1102
- Philips Semiconductors WWW: http://www.semiconductors.philips.com

2. SYSTEM WITHOUT XRAM

Applications not requiring additional external data memory (only use the internal on-chip RAM) have three options for locating the Flash loader software part. These three options are explained in detail in the next paragraphs.

- Loader on-chip (EPROM or OTP XA) and boot from on-chip ROM
- Loader on-chip (EPROM or OTP XA), but boot from external Flash memory
- Romless XA, loader in separate external ROM

2.1 Boot from on-chip ROM

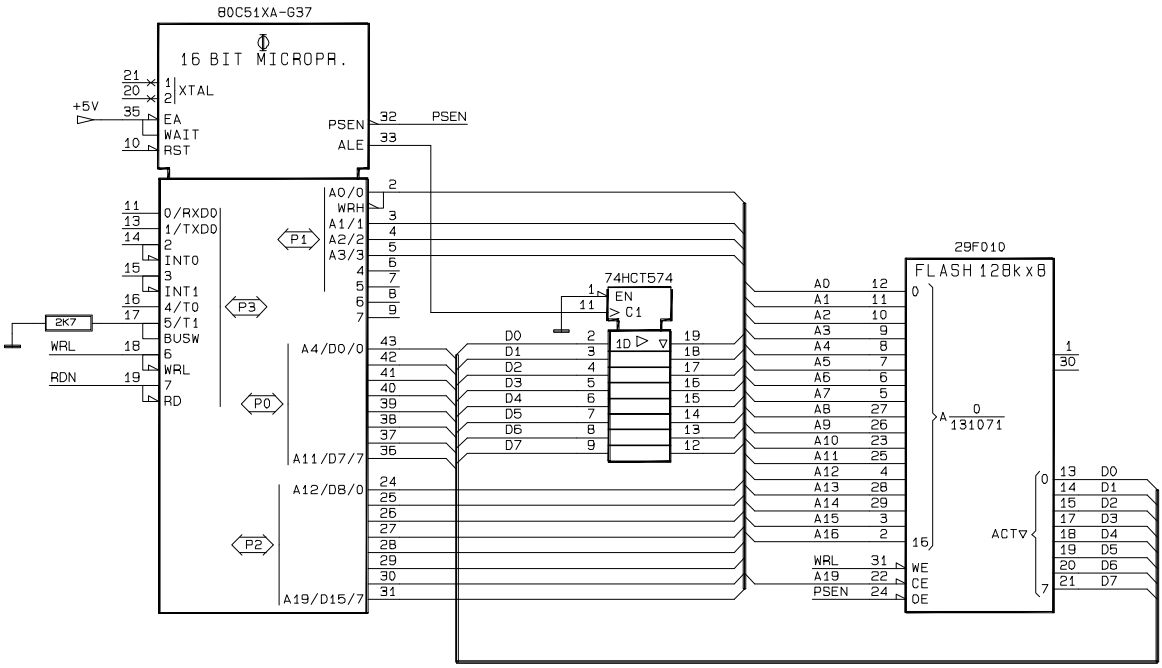

Figure 1. Boot from on-chip XA ROM

The system shown in figure 1 always boots from internal XA ROM (EA pin high). The internal XA ROM contains the start-up code, the vector table and the Flash loader program.

After start-up initialization the application program inside the external Flash memory is called. However, before the XA executes an external bus cycle the WAITDisable bit in the BCR register must be set. This causes the XA to ignore the status of WAIT input pin and allows tying the WAIT input high for applications that use internal code and do not need the WAIT function.

Besides start-up / initialization code the internal ROM of the XA also contains the Flash (re)loader program. After receiving a request to reload the Flash (for example via the UART) the application calls the loader software inside the internal ROM (see chapter 4). After updating the Flash memory contents the 'new' application program is called (or a software reset is generated).

The XA interrupt vectors are located at fixed addresses from address 0 to 120h and thus also part of the on-chip ROM. Interrupt vectors are pointers to interrupt routines. These routines are part of the application program and located inside the Flash memory. Therefore, the vectors should in some way be re-directed to fixed address pointers inside the Flash memory, or all interrupt routines must be at fixed addresses.

Advantages:

- Simple / cheap design (no extra external hardware needed)
- Start-up with empty Flash is easy to test. This is convenient for production and diagnostic purposes.

Disadvantages:

- Not flexible (fixed addresses of routines and vectors and pointers)
- Start-address of application must be fixed
- Price of OTP XA against Romless XA
- Need of a kind of 'secondary vector table' with fixed addresses
- When mapped at address 0 the first 32 Kbyte of the Flash memory is useless, because of overlap with on-chip ROM
- No easy software emulation, because it is not located at address 0

2.2 Boot from external Flash

The external Flash memory of the system shown in figure 2 contains the start-up code, the vector table and the application program. The XA on-chip ROM contains the Flash reloader software.

The intention of the system is to boot / power-up always from code inside external Flash memory. This means that the EA pin needs to be 'low' during power-up, when it is sampled by the XA. To take care of this the output of the flip-flop is put in 'reset' state (by the resistor and capacitor).

However, the Flash reloader software is inside the on-chip XA ROM. If a Flash reload is requested a re-start, but now from internal ROM (EA high), is needed. To achieve this, an external data memory write cycle to the flip-flop (at address $80000) with D0 (databus line 0) high is performed. After that, the system is re-started using an external reset circuit (MAX311). Now the internal on-chip code is running and the Flash memory can be re-loaded.

Again, if running internally (EA/WAIT pin high), before an external bus cycle is performed the WAITDisable bit must be set in the BCR register (see previous paragraph).

The address ($80000) used to set the flip-flop must be outside the Flash memory range, because otherwise a write operation into the Flash is executed. To generate an external system reset a '0' is written to port pin P3.3 (manual reset input of the MAX811).

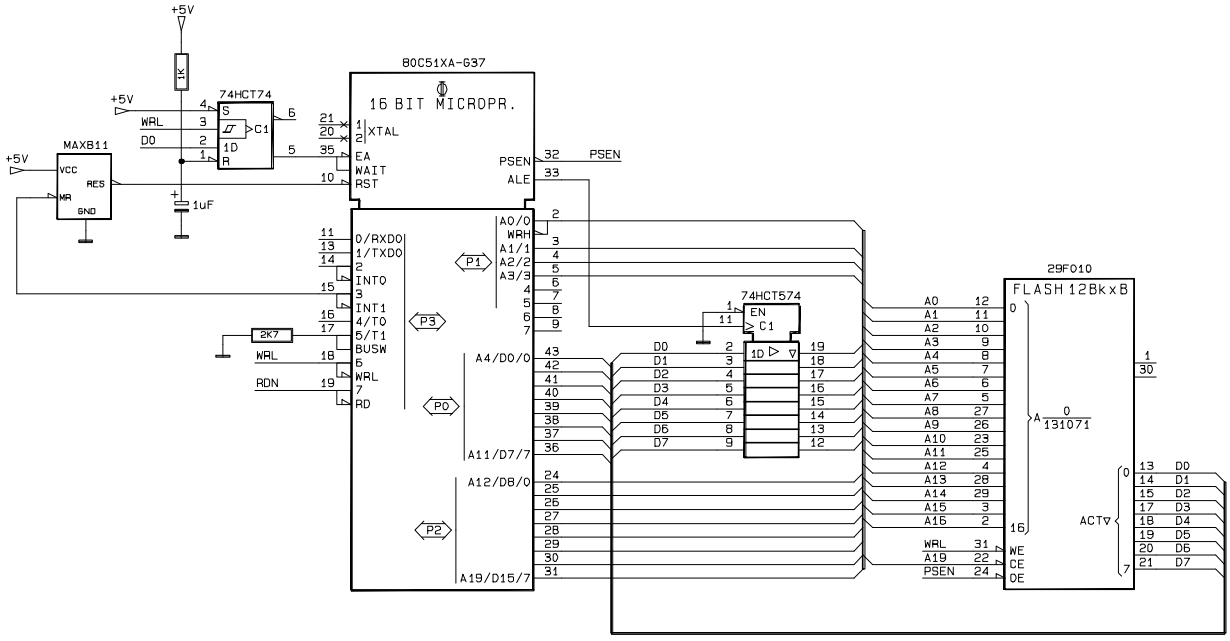

Figure 2. XA with on-chip code, but always boot from external flash

After updating the Flash contents again the system needs to be re-started, but now from the newly loaded code in external Flash memory. This means that now a ‘zero’ needs to be written to the flip-flop (EA low), and then again an external system ‘reset’ must be performed (write ‘0’ to P3.3).

The external reset circuitry is used because the XA doesn’t (re)sample the EA and BUSW pins after an internal reset (software RESET instruction or watchdog). If the use of port pin P3.3 (or other) is a problem, one could consider to connect the RDN line to the manual reset (MR) input of the MAX811. In that case, to generate a hardware reset, an external data memory read needs to be performed.

Advantages:

- Flexible design
- Interrupt vector table inside Flash
- Easy emulation of code, because Flash is mapped at address 0
- Linking / loading at address 0

Disadvantages:

- Extra glue logic needed (reset circuitry, flip-flop)
- Price of OTP XA against Romless XA
- Extra measures to take at first time power-up with empty flash (force EA high !)

2.3 Boot from external ROM

The system in figure 3 shows a Romless XA system with external Flash memory, containing the application program, at address $0 and a separate Flash loader PROM mapped above the Flash memory space ($20000). At power-up or after reset the system boots the application from the Flash memory (EA pin is low).

If a request to reload the Flash is received (for example via the UART) the application ‘jumps’ to the reload routine (see chapter 4) in external PROM. After reloading the Flash a software reset must be generated to restart the application.

For the example in figure 3 a 4 Kbyte PROM is selected. This is because of chip size and price reasons. Of course it is also possible to use a ‘normal’ EPROM type 27C...

When more Flash memory is needed, both chip enable inputs of Flash and PROM can be connected to address line A19 instead of A17.

Writing into the Flash at address $0 up to the first 512 bytes (1Kbyte for the XA-S3) is only possible using the MOVX instruction.

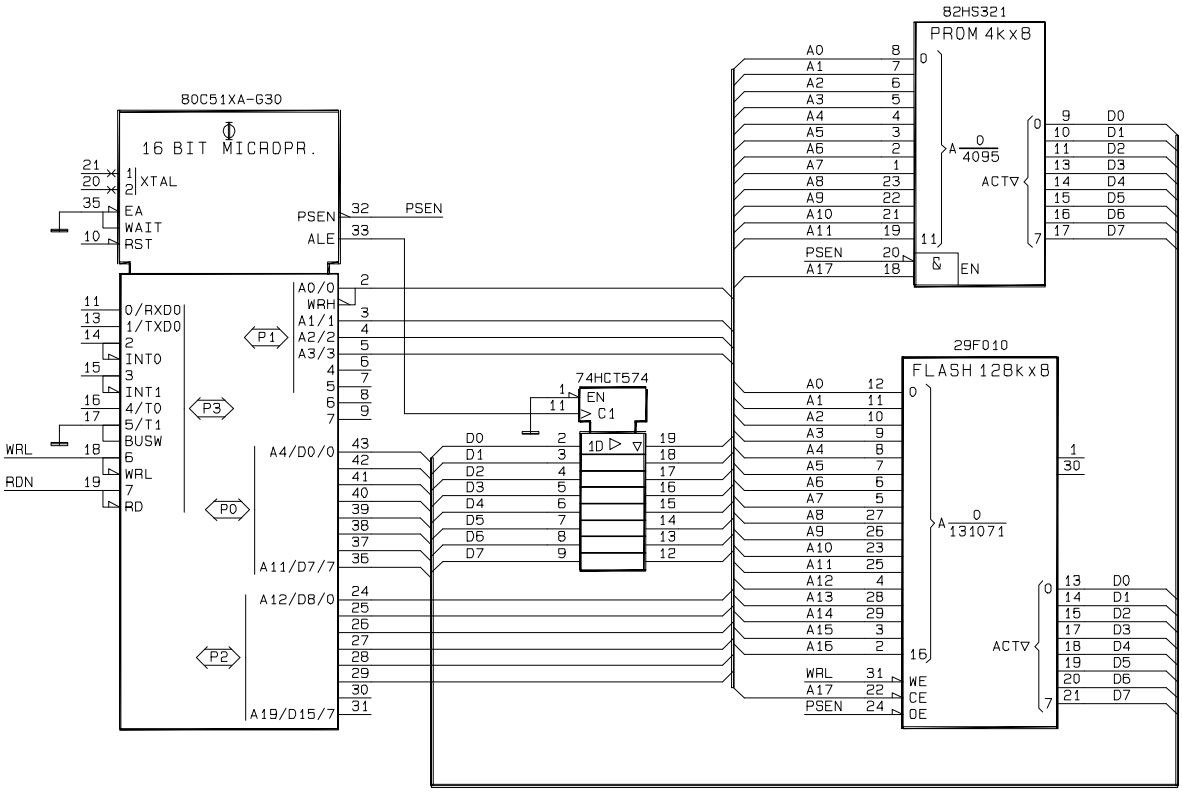

Figure 3. Romless XA, external ROM and Flash

Advantages:

- Flexible design
- Interrupt vector table inside Flash
- Easy emulation of code
- Linking / loading at address 0
- ROMless XA + external PROM is cheaper than on-chip ROM/OTP XA

Disadvantages:

- Extra external (E)PROM needed
- More PCB space needed
- Extra measures to take at first time power-up with empty flash (force boot from PROM)

3. SYSTEM WITH XRAM

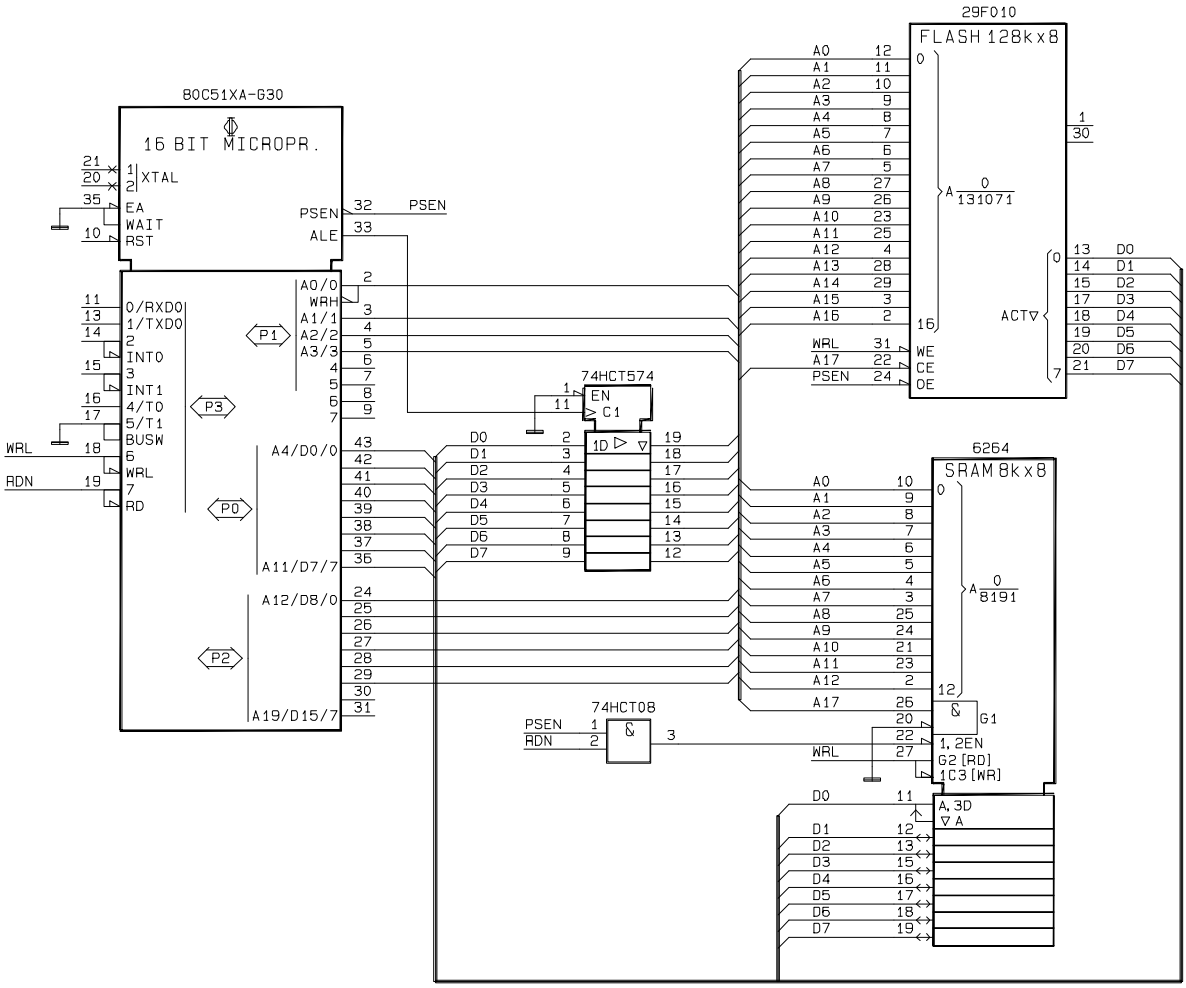

Figure 4. Romless XA, external SRAM and Flash

The system shown in figure 4 has 8 Kbytes (or if necessary more) of external data memory available. In this case an OTP/Eprom XA is superfluous, therefore a ROMless XA is selected.

At power-up the system always boots from code inside external Flash memory (EA is low). The Flash device also contains the reloader software. The SRAM device is mapped in both data and program memory space at address $20000. If a Flash reload is requested the re-loader software part is copied from flash into the static RAM and then executed.

After updating the Flash contents the system needs to re-boot, but now from the new re-loaded code in external Flash. This is simply done by executing a RESET instruction.

If a larger amount of Flash memory is needed, both chip enable inputs of Flash and SRAM can be connected to address line A19 instead of A17 (SRAM space is moved to $80000).

Writing into the Flash at address $0 up to the first 512 bytes (1Kbyte for the XA-S3) is only possible when using the MOVX instruction.

Advantages:

- Simple design
- Interrupt vector table inside Flash
- Easy emulation of code
- Linking / loading at address 0
- Lower cost (Romless XA, but SRAM device needed)

Disadvantages:

- RAM is not mapped at 0, so not usable for system stack
- PCB space (extra external SRAM)
- Extra measures to take at first time power-up with empty flash (insert a programmed Flash !)

4. FLASH LOADER SOFTWARE

This chapter describes tested driver software for an 8-bit XA-G3 system with an external AMD 29F010 (128 Kbytes) Flash part, but without external data memory (SRAM). The driver software provides the source code necessary to erase and re-program the 29F010 Flash part. The code (contents) for Flash memory re-loading is obtained from an Intel-hex file that is serially downloaded using UART0.

In the example application the base address of the Flash memory is mapped at $40000h in data memory space. To change this adjust the include file flash.h and recompile all modules.

The compiler used: XAC compiler V1.1 from TASKING.

Overview of all modules:

BOOT.C	Contains the main routine called after power-up initialization. First the flash memory is erased. Next, an Intel-hex file is uploaded and the flash is re-programmed. And, finally a RESET instruction is executed to re-boot the system.
LOADER.C	Handles reading the Intel-hex file. Checksum of each record is checked and if needed an error message is generated. Data bytes are passed to the embedded program function to write into the flash device.
FLASH.C	Contains the 29F010 Flash embedded program and chip-erase functions: *EmbeddedProgram()* Performs the actual byte program on the Flash device. After variable setup, the unlock sequences are sent, with the final unlock cycle being the byte to program at the appropriate address. *ChipErase()* Performs a chip erase on the Flash device, clearing all data and returning all locations to a value of FF hexadecimal. Protected sectors are not checked. On return a status is flagged (un / successful chip erase).
SERIAL.C	Handles the serial communication part using UART0. Contains hardware initialization, send byte, receive byte and send ASCII string routines.
FLASH.H	Include file that contains local definitions for Flash driver software package.

Note

For systems using 16-bit databus width the module flash.c needs to be adjusted. All addresses used in the unlock sequences need to be incremented by 1. This is because in a 16-bit system address line A0 is used as the WRH signal and A1 to A16 is connected to the Flash device.

5. REFERENCES AND TOOLS

Used references and development tools:	**how to get**
• AMD Flash development kit	see http://www.amd.com
• AMD Flash memory products 1994/1995 data book	see http://www.amd.com
• MAXIM Integrated products data book	see http://www.maxim.com
• HI-Tech XA C compiler (version 7.60)	see http://www.htsoft.com
• Tasking XA C compiler (version 1.1)	see http://www.tasking.com
• FDI XTEND board (rev. A1)	XTEND-G3
• Philips 16-bit 80C51XA data handbook IC25	9397 750 00733

APPENDIX 1. BOOT.C

```
/*************************************************************************/
/*  Name of module    : BOOT.C                                           */
/*  Creation date     : 1997-04-21                                       */
/*  Program language  : C                                                */
/*  Name              : P.H. Seerden                                     */
/*                                                                       */
/*          (C) Copyright 1997 Philips Semiconductors B.V.               */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  History:                                                             */
/*                                                                       */
/*  97-04-21    P.H. Seerden    Initial version                          */
/*                                                                       */
/*************************************************************************/


#include "flash.h"


void main(void)
{
    ua_init();           /* initialise UART0 for console-in/console-out    */

    PrintString("\n\n\n29F010 Flash loader Program V1.0\n\n"
                "Step 1 - Erase Flash memory\n"
                "Step 2 - Upload Intel-hex file\n"
                "Step 3 - Execute software RESET\n\n"
                "Hit any key to continue !!");

    ci();                            /* wait for any character            */

    PrintString("\nErasing chip .");
    if (ChipErase())
        PrintString("\nChip erase error !!\n");
    else
        PrintString("\nSuccessfully erased 29F010 !\n");

    PrintString("\n * downloading *\n");
    switch (Download())
    {
        case 0 : PrintString("\nready and ok\n");         break;
        case 1 : PrintString("\nChecksum Error\n");       break;
        case 2 : PrintString("\nBad record type\n");      break;
        case 3 : PrintString("\nFlash program Error\n");  break;
        default: PrintString("\nUnknown Error\n");        break;
    }

#pragma asm
    RESET               ; execute software reset instruction
#pragma endasm
}
```

APPENDIX 2. LOADER.C

```
/***************************************************************************/
/*  Name of module    : LOADER.C                                           */
/*  Creation date     : 1997-04-21                                         */
/*  Program language  : C                                                  */
/*  Name              : P.H. Seerden                                       */
/*  Description       : Handles downloading Intel Hex-32 files.            */
/*                                                                         */
/*          (C) Copyright 1997 Philips Semiconductors B.V.                 */
/*                                                                         */
/***************************************************************************/

#include "flash.h"


_rom char ascii[] = "0123456789ABCDEF";
static BYTE checkSum;


/***************************************************************************/
/*  Load2Hex()                                                             */
/*                                                                         */
/*  purpose:   This routine reads two hex bytes and returns their          */
/*             binary byte value. To read the two hex bytes the routine    */
/*             ci() (console in) is called.                                */
/*                                                                         */
/***************************************************************************/
static BYTE Load2Hex(void)
{
  BYTE  t,i,s;

    i = ci();                    /* read one hex byte from the UART       */
    for (t = 0; t < 16; t++)     /* convert hex byte to binary low nibble */
        if (ascii[t] == i)
            break;

    i = ci();                    /* read second hex byte                  */
    for (s = 0; s < 16; s++)     /* convert to binary high nibble         */
        if (ascii[s] == i)
            break;

    s = (t<<4) + s;              /* merge low and high nibble             */
    checkSum = checkSum + s;     /* update checksum of this record        */
    return s;
}


/***************************************************************************/
/*  Load4Hex()                                                             */
/*                                                                         */
/*  purpose:    This routine reads four hex bytes and returns their        */
/*              binary word value.                                         */
/*                                                                         */
/***************************************************************************/
static WORD Load4Hex(void)
{
  WORD t;

    t = Load2Hex();                  /* first do the low byte             */
    return (t<<8) + Load2Hex();      /* now the high byte and merge       */
}
```

```
/*************************************************************************/
/*  Download()                                                           */
/*                                                                       */
/*  purpose:    Download Intel-hex file. Use EmbeddedProgram() to write  */
/*              bytes to flash.                                          */
/*                                                                       */
/*************************************************************************/
BYTE Download(void)
{
  BYTE length;
  LONG addr,segm,address;

    segm = 0;
    while (1)                       /* start downloading Intel Hex file    */
    {
        while (ci() != ':')         /* wait for : (start record)           */
            ;

        checkSum = 0;
        length   = Load2Hex();                    /* read record length   */
        addr     = Load4Hex();                    /* read address         */

        switch (Load2Hex())                       /* read record type     */
        {
          case 0:                                 /* read data record     */
                address = base + segm + addr;
                break;
          case 1:                                 /* read end record      */
                Load2Hex();                       /* fetch last checksum  */
                if (checkSum != 0)
                    return 1;                     /* checksum error       */
                return 0;                         /* download OK          */
          case 2:                                 /* read esar record     */
                segm = Load4Hex();
                segm = segm<<4;
                length -= 2;
                break;
          case 3:                                 /* read start address   */
                break;
          case 4:                                 /* read elar record     */
                segm = Load4Hex();
                segm = segm<<16;
                length -= 2;
                break;
          default:
                return 2;                         /* bad record type      */
        }
        while (length != 0)                       /* read data bytes      */
        {
            if (EmbeddedProgram(address,Load2Hex()))
                return 3;                         /* byte program error   */
            address ++;
            length --;
        }
        Load2Hex();                               /* fetch checksum       */
        if (checkSum != 0)
            return 1;                             /* checksum error       */
    }
}
```

APPENDIX 3. FLASH.C

```
/*****************************************************************************/
/*  Name of module    : FLASH.C                                                */
/*  Creation date     : 1997-04-21                                             */
/*  Program language  : C                                                      */
/*  Name              : P.H. Seerden                                           */
/*                                                                             */
/*          (C) Copyright 1997 Philips Semiconductors B.V.                     */
/*                                                                             */
/*****************************************************************************/
/*                                                                             */
/*  Description:                                                               */
/*                                                                             */
/*   Source code for programming 29F010 Flash components.                      */
/*                                                                             */
/*****************************************************************************/

#include "flash.h"


/*****************************************************************************/
/*  EmbeddedProgram()                                                          */
/*                                                                             */
/*  purpose:    Performs the AMD Embedded Programming algorithm.               */
/*              The device is a 29F010, it is not checked for protected        */
/*              sectors. A polling address is setup, the address is            */
/*              started, a 4-cycle unlock sequence is initiated the            */
/*              device is polled using dq7 poll for programming.               */
/*                                                                             */
/*****************************************************************************/
/*  parameters:     LONG address = address to program                          */
/*                  BYTE val     = byte to program                             */
/*****************************************************************************/
/*  return value:   0 : successful                                             */
/*                  1 : unsuccessful                                           */
/*****************************************************************************/
BYTE EmbeddedProgram(LONG address, BYTE val)
{
  _far BYTE *p;
       BYTE i;

    p  = (_far BYTE *)(base + 0x5555);
    *p = 0xAA;                          /* first WR cycle                */

    p  = (_far BYTE *)(base + 0x2AAA);
    *p = 0x55;                          /* second WR cycle               */

    p  = (_far BYTE *)(base + 0x5555);
    *p = 0xA0;                          /* third WR cycle                */

    p  = (_far BYTE *)address;
    *p = val;                           /* output the byte to the device  */

    for (;;)
    {
        i = *p;                            /* read data                  */
        if ((val & 0x80) == (i & 0x80))    /* DQ7 = true data bit ?      */
            return 0;                      /* Embedded program ok !      */
```

```
        if (i & 0x20)                       /* is DQ5 = 1 ?                */
        {
            i = *(_far BYTE *)address;       /* DQ6 + DQ5 both changed ?    */
            if ((i & 0x80) == (val & 0x80))
                return 0;                   /* device passed DATA POLL     */
            else
                return 1;                   /* DQ7 not true, while DQ5=1   */
        }
    }
}


/*************************************************************************/
/*  ChipErase()                                                          */
/*                                                                       */
/*  purpose:    Performs the AMD Embedded Erase algorithm.               */
/*              The device is a 29F010 5v device, it is not checked for  */
/*              protected sectors. The 6-cycle unlock sequence is        */
/*              initiated, and the device is polled using dq7 poll.      */
/*                                                                       */
/*************************************************************************/
/* return value:   0 : successful                                        */
/*                 1 : unsuccessful                                      */
/*************************************************************************/
BYTE ChipErase(void)
{
  _far BYTE *address;
       BYTE temp;

    address = (_far BYTE *)(base + 0x5555);
    *address = 0xAA;                        /* first WR cycle            */

    address = (_far BYTE *)(base + 0x2AAA);
    *address = 0x55;                        /* second WR cycle           */

    address = (_far BYTE *)(base + 0x5555);
    *address = 0x80;                        /* third WR cycle            */

    address = (_far BYTE *)(base + 0x5555);
    *address = 0xAA;                        /* fourth WR cycle           */

    address = (_far BYTE *)(base + 0x2AAA);
    *address = 0x55;                        /* fifth WR cycle            */

    address = (_far BYTE *)(base + 0x5555);
    *address = 0x10;                        /* sixth WR cycle            */

    for (;;)                        /* polling algorithm                 */
    {
        temp = *address;            /* read data from unprotected address */
        if (temp & 0x80)            /* is DQ7 = 1 ?                      */
            return 0;               /* We're done, DATA POLL ok          */

        if (temp & 0x20)            /* is DQ5 = 1 ?                      */
        {
            temp = *address;        /* DQ6 + DQ5 changed simultaneously ? */
            if (temp & 0x80)
                return 0;           /* device passed DATA POLL algorithm  */
            else
                return 1;           /* DQ7 not true, even after DQ5 = 1   */
        }
    }
}
```

APPENDIX 4. SERIAL.C

```
/*************************************************************************/
/*  Name of module    : SERIAL.C                                         */
/*  Creation date     : 1997-04-21                                       */
/*  Program language  : C                                                */
/*  Name              : P.H. Seerden                                     */
/*  Description       : Console I/O for P51XA UART 0.                    */
/*                                                                       */
/*          (C) Copyright 1997 Philips Semiconductors B.V.               */
/*                                                                       */
/*************************************************************************/

#include <regxag3.sfr>

#define BAUD_RATE      19200
#define OSC            20000000L              /* Xtal frequency          */
#define DIVIDER        (OSC/(64L*BAUD_RATE))

void ua_init(void)
{
    TL1 = RTL1 = -DIVIDER;
    TH1 = RTH1 = -DIVIDER >> 8;
    TR1 = 1;                            /* enable timer 1               */
    S0CON = 0x52;                       /* mode 1, receiver enable      */
}

void co(char c)
{
    if (c == '\n')
    {
        while (!TI_0)   ;
        S0BUF = '\r';
        TI_0 = 0;
    }
    while (!TI_0)   ;
    S0BUF = c;
    TI_0 = 0;
}

char ci(void)
{
  char    c;

    while (!RI_0)   ;
    c = S0BUF & 0x7F;
    RI_0 = 0;
    return c;
}

void PrintString(const char *p)
{
  _rom char *s;

    s = (_rom char *)p;
    while (*s != '\0')
    {
        if (*s == '\n')
            co('\r');                   /* output a '\r' first          */
        co(*s);
        s++;
    }
}
```

APPENDIX 5. FLASH.H

```
/*****************************************************************************/
/*  Name of module    : FLASH.H                                                */
/*  Creation date     : 1997-04-21                                             */
/*  Program language  : C                                                      */
/*  Name              : P.H. Seerden                                           */
/*                                                                             */
/*          (C) Copyright 1997 Philips Semiconductors B.V.                     */
/*                                                                             */
/*****************************************************************************/
/*                                                                             */
/*  Description:                                                               */
/*                                                                             */
/*   Local declarations for the XA flash driver package.                       */
/*                                                                             */
/*****************************************************************************/
/*                                                                             */
/*  History:                                                                   */
/*                                                                             */
/*  97-04-21    P.H. Seerden    Initial version                                */
/*                                                                             */
/*****************************************************************************/

typedef unsigned char    BYTE;
typedef unsigned short   WORD;
typedef unsigned long    LONG;


#define base      0x40000           /* base address of flash memory        */


extern void co(char ch);
extern char ci(void);
extern void ua_init(void);
extern void PrintString(const char *s);
extern BYTE Download(void);
extern BYTE EmbeddedProgram(LONG address, BYTE val);
extern BYTE ChipErase(void);
```