Semiconductors

AN10342

Using LPC900 code Flash as data storage

Rev. 01 — 13 December 2004 Application note

Document information

Info	Content
Keywords	LPC900, Flash Data storage, non volatile memory
Abstract	The P89LPC900 family of microcontrollers have In Application Programming Lite, which can be used to erase and reprogram the code Flash of the microcontroller.

PHILIPS

Revision history

Rev	Date	Description
01	20041213	Initial version

Contact information

For additional information, please visit: **http://www.semiconductors.philips.com**

For sales office addresses, please send an email to: **sales.addresses@www.semiconductors.philips.com**

© Koninklijke Philips Electronics N.V. 2004. All rights reserved.

1. Introduction

The P89LPC900 family of microcontrollers have IAP Lite, which can be used to erase and reprogram the code Flash of the microcontroller.

With the byte erase feature of the P89LPC900 Flash, the microcontroller can be used to use part of the Flash memory as data storage.

This application note describes the way the Flash memory of the LPC900 can be used as data storage. This way the Flash memory can also be used as storage for non volatile data. A code example of how to use the byte erase feature is also included in this application note in Appendix 5.1.

2. Using IAP Lite to erase and program code Flash

The mechanism to erase and program a single byte in the Flash memory of the LPC900 is by using IAP Lite.

IAP Lite has an erase / program (EP) command. To use this command first an address has to be set up.

The sequence that can be used to do a erase program command on a Flash byte is:

- Move the LOAD command in the FMCON SFR.
- Move the low address of the byte to be programmed in the FMADRL SFR.
- Move the high address of the byte to be programmed in the FMADRH SFR.
- Move the data to be programmed in the FMDATA SFR.
- Move the ERASEPROGRAM command into the FMCON SFR.

After this sequence the internal Flash state machine will first do an erase operation on the Flash byte and then program the data in the same Flash byte. The erase program command takes 4ms, 2ms to erase and 2ms to program the byte.

Appendix 5.1 will show the

```
bit flash_write(char addresshigh, char addresslow, char databyte)
```

function that will do the sequence.

3. Reading the code Flash

The code Flash can be read by a simple MOVC instruction.

In a C program in μVision 3 it is best to put the data in an array and read the location out like this:

x = data_array[i];

The μVision 3 compiler will translate this into a MOVC A, @A+PC or MOVC A, @A+DPTR assembly code.

© Koninklijke Philips Electronics N.V. 2004. All rights reserved.

4. Considerations when using Flash as data storage

When using Flash as data storage on the LPC900 family you have to make sure that the area you are using as program space is not overwritten by bytes intended for data storage, otherwise your application code might overwrite its own application code.

The best way to set up a section of data Flash in μVision 3 is to reserve code space in an array like this:

```
volatile unsigned char code dataflash[DATAFLASHSIZE] _at_ DATAFLASHSTART;
```

The “volatile” will indicate that the compiler will not optimize out the array if it isn’t initialized and always reserve this space for that specific array in code Flash.

The “code” will indicate that the array will be stored in Flash code space.

Also make sure that the sector the data is stored in is not MOVC protected. If it is MOVC protected then the user program will be inhibited from reading out the data.

For additional information on IAP Lite please see the Users Manual of the device you are using.

© Koninklijke Philips Electronics N.V. 2004. All rights reserved.

5. Appendix

5.1 Appendix A: Using Flash as data Storage code example

```
//*************************************************************************
//*  main.c
//*By : Bauke Siderius
//* Date : Oct 2004
//* Description : Flash Data Storage using LPC935
//*************************************************************************
#include <REG935.H>
#include <stdio.h> // declarations for IO functions
#define LOAD 0x00; // define the load command
#define ERASEPROGRAM 0x68; // define the eraseprogram command
#define OK 1 // define OK as 1
#define FAIL 0 // define fail as 0
#define DATAFLASHSIZE 4 // define size of the dataflash array
#define DATAFLASHSTART 0x0400 // define the startaddress of the array
#define LOWBYTE(address) ((unsigned char)(address))
#define HIGHBYTE(address) ((unsigned char)(((unsigned int)(address))>> 8))

volatile unsigned char code dataflash[DATAFLASHSIZE] _at_ DATAFLASHSTART;

//*************************************************************************
//* Functions
//*************************************************************************
void init(void);
bit flash_write(char addresshigh, char addresslow, char databyte);
unsigned char get_uart();
unsigned char get_ascii();
unsigned char ascii_to_hex(unsigned char ch);
void print_hex_to_ascii(unsigned char);
//*************************************************************************
//* init()
//* Input(s) : none.
//* Returns : none.
//* Description : initialization of P89LPC935
//*************************************************************************
void init(void)
{
  P1M1 = 0x00; // P1 in Quasi bi mode
  P1M2 = 0x00;
  SCON = 0x52; // init UART
  BRGR0 = 0x70; // set-up baudrate generator
  BRGR1 = 0x01; // for 19200 baud with
  BRGCON = 0x03; // internal RC oscillator
}
//*************************************************************************
//* main()
//* Input(s) : none.
//* Returns : none.
```


© Koninklijke Philips Electronics N.V. 2004. All rights reserved.

```
//* Description : main loop
//*****************************************************************************
void main(void)
{
  unsigned char index, temp, addrh, addrl, datab, line = 0;
  unsigned int address;
  init(); // initialize part
  printf("Flash as datastorage demo\n");
  while(1) // main loop
  {
    printf("\nPress: P for programming one byte.\n");
    printf("Press: R to read Flash array.\n");
    temp = get_uart();
    switch(temp) // switch on type of command
    {
      case 'P': // program byte function
      case 'p':
      {
        for(index = 0; index < 4; index++)
        {
          printf("\nEnter data byte to be programmed\n");
          datab = get_ascii();    // Read databyte to be programmed
          addrh = HIGHBYTE(DATAFLASHSTART);
          addrl = (LOWBYTE(DATAFLASHSTART) + index);
          if(!(flash_write(addrh, addrl, datab))) // Call byte programming function
          {
            printf("Flash byte programming error occured\n"); // Programming error
occured
          }
        }
        break;
      }
      case 'R': // read byte function
      case 'r':
      {
        for(index = 0; index < 4; index++)
        {
          address = (DATAFLASHSTART + index); // get high byte of address
          printf("\nDatabyte = ");
          print_hex_to_ascii(dataflash[address]);// Print databyte
        }
        break;
      }
      default: // incorrect record type
      {
        printf("\nIncorrect command.\n");
        break;
      }
    }
  }
}
```


© Koninklijke Philips Electronics N.V. 2004. All rights reserved.

```
//*************************************************************************
//* flash_write_byte()
//* Input(s) : Memory address to write to.
//* Returns : bit, flash write succes or fail.
//* Description : write databyte to flash
//*************************************************************************
bit flash_write(char addresshigh, char addresslow, char databyte)
{
  FMCON = LOAD; // set up load command
  FMADRL = addresslow; // set up low databyte
  FMADRH = addresshigh; // set up high databyte
  FMDATA = databyte; // set up databyte
  FMCON = ERASEPROGRAM; // erase program command
  if(FMCON&0x8F) // check for any error bits
  {
    return FAIL;   // return fail on error
  }
  else
  {
    return OK;  // return OK if operation was succesful
  }
}
//*************************************************************************
//* get_uart()
//* Input(s) : none.
//* Returns : character read from ISP record.
//* Description : check if in ICP mode
//*************************************************************************
unsigned char get_uart()
{
  unsigned char ch;
  while(!RI) // wait until reception is complete
  {
  }
  RI = 0;
  ch = SBUF; // read UART buffer
  while(!TI) // wait until previous
  {
  }
  TI = 0; // transmission is complete
  SBUF = ch; // send character
  return ch; // return byte
}
//*************************************************************************
//* get_ascii()
//* Input(s) : none.
//* Returns : character.
//* Description : ascii character read from uart converted to hex.
//*************************************************************************
unsigned char get_ascii()
{
```


© Koninklijke Philips Electronics N.V. 2004. All rights reserved.

```
  unsigned char record_byte, ch;
  ch = get_uart(); // send character
  record_byte = (ascii_to_hex(ch))<<4; // read high nibble
  ch = get_uart(); // send character
  record_byte += (ascii_to_hex(ch)); // read low nibble
  return record_byte; // return byte
}
//*****************************************************************************
//* ascii_to_hex()
//* Input(s) : character in ascii format
//* Returns : character in hex format
//* Description : converts ascii to hex
//*****************************************************************************
unsigned char ascii_to_hex(unsigned char ch)
{
  if (ch & 0x40) // convert ASCII character
  {
    ch += 0x09;
  }
  ch &= 0x0F; // into 2 digit Hex
  return ch;
}
//*****************************************************************************
//* print_hex_to_ascii()
//* Input(s) : character
//* Returns : none.
//* Description : converts hex to ascii and print the character to the UART
//*****************************************************************************
void print_hex_to_ascii(unsigned char ch)
{
  char temp_character;
  temp_character = ch;
  temp_character >>= 4; // get highest nible
  temp_character &= 0x0F; // save lower nibble
  if (temp_character >= 10) // check if A-F
  {
    temp_character += 7; // offset for A-F
  }
  temp_character += 0x30; // add '0' to get ascii
  printf("%c", temp_character); // print upper nible
  temp_character = ch;
  temp_character &= 0x0F; // save lower nibble
  if (temp_character >= 10) // check if A-F
  {
    temp_character += 7; // offset for A-F
  }
  temp_character += 0x30; // add '0' to get ascii
  printf("%c", temp_character); // print lower nible
}
```


© Koninklijke Philips Electronics N.V. 2004. All rights reserved.

6. Disclaimers

Life support — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status 'Production'), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

© Koninklijke Philips Electronics N.V. 2004. All rights reserved.

7. Contents

© Koninklijke Philips Electronics N.V. 2004

All rights are reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent- or other industrial or intellectual property rights.

Date of release: 13 December 2004
Document number: 9397 750 14438

Published in the U.S.A.