Semiconductors

AN10256

Philips ARM LPC microcontroller family

Rev. 02 — 25 October 2004 Application note

Document information

Info	Content
Keywords	ARM LPC, IAP
Abstract	This application note provides code samples in C and assembly, which will help the end user use the In-Application Programming (IAP) technique for programming the on-chip Flash.

Revision history

Rev	Date	Description
02	20041025	• The format of this application note has been redesigned to comply with the new presentation and information standard of Philips Semiconductors • Added a note on enabling the ARM/Thumb interworking option in the compiler/assembler • Corrected the data type mismatch between the IAP function call and pointer initialization
01	20031212	Initial version

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

1. Introduction

In-Application (IAP) programming is the performing of erase and write operations on the on-chip Flash memory as directed by the end-user application code. The Flash boot loader provides the interface for programming the Flash memory. For detailed information on the In-Application Programming please refer to the Flash Memory System and Programming chapter in the *ARM LPC device User Manual*. In this application note, code samples are provided in C and assembly, which show how IAP may be used. The IAP routine resides at 0x7FFFFFF0 and is Thumb code.

2. IAP code in C

The IAP function could be called in the following way using C. This section is taken from the *User Manual*.

Define the IAP location entry point. Since the 0th bit of the IAP location is set there will be a change to the Thumb instruction set when the program counter branches to this address.

```
#define IAP_LOCATION 0x7ffffff1
```

Define data structure or pointers to pass IAP command table and result table to the IAP function

```
unsigned int command[5];
unsigned int result[2];
```

```
or
```

```
unsigned int * command;
unsigned int * result;
command=(unsigned int *) 0x……
result= (unsigned int *) 0x……
```

Define pointer to function type, which takes two parameters and returns void. Note the IAP returns the result with the base address of the table residing in R1.

```
typedef void (*IAP)(unsigned int [],unsigned int[]);
IAP iap_entry;
```

Setting function pointer

```
iap_entry=(IAP) IAP_LOCATION;
```

Whenever user wishes to call IAP within the application, the following statement could be used.

```
iap_entry (command, result);
```


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

3. IAP code in Assembly

The IAP routine may be called in the following way using ARM assembly code. This code was developed using the ARM Developer Suite (ADS1.2). The assembler directives will change depending upon the assembler the end-user will use.

```
;--------------------------------------------------------
AREA arm_code, CODE
CODE32
EXPORT initial ; This routine could be
               ; linked to other
               ; routines using this
               ; global symbol

;--------------------------------------------------------
;  Symbol definitions
;--------------------------------------------------------

IAP_ENTRY  EQU   0x7ffffff1  ; IAP entry point
COMMAND    EQU   0x.....     ; Command table pointer
RESULT     EQU   0x.....     ; Result table pointer

;--------------------------------------------------------
;  Main
;--------------------------------------------------------

initial
STMFD SP!,{R0-R2,R14} ;Push the register set
                      ; and link register into
                      ; stack

LDR R0,=COMMAND       ; Set the pointers for
LDR R1,=RESULT        ; command and result
                      ; tables

;--------------------------------------------------------
; Once the pointers are set, the command code and its
; respective parameters need to be stored in the command
; table. An example is provided below where the command
; code (54) for IAP command 'Read Part ID' is stored into
; the command table
;--------------------------------------------------------

MOV R2,#0x36
STR R2, [R0]

;--------------------------------------------------------
; Please look below (after END) for description for how the
; IAP routine is called
;--------------------------------------------------------
```


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

```
BL jump_to_IAP

;-------------------------------------------------------
; At this point user has to analyze the result table and
; take action depending upon the status code returned by
; the IAP routine.(Code not shown)
;-------------------------------------------------------

LDMFD SP!,{R0-R2,R14} ; Pop link register
                      ; and register workspace
MOV PC,LR

;-------------------------------------------------------
;Call IAP routine
;-------------------------------------------------------

jump_to_IAP
LDR R12,=IAP_ENTRY

BX R12              ; Branch to 0x7FFFFFF1
                     ; and Change to thumb
                    ; instruction set

END
```

To call the IAP function, we branch and link (BL) to a small routine jump_to_IAP and then we call the IAP function using BX. By performing BL jump_to_IAP we get R14 to point to the next instruction and then using BX instruction we can directly jump to the IAP routine and change to Thumb instruction set.

If user wishes to call the IAP routine using Thumb code, then the code could be as follows.

```
;-------------------------------------------------------
AREA thumb_code, CODE
CODE16
EXPORT initial    ; this routine could be
                  ; linked to other
                  ; routines using this
                  ; global symbol
;-------------------------------------------------------
;  Symbol definitions
;-------------------------------------------------------

IAP_ENTRY   EQU  0x7ffffff1; IAP entry point
COMMAND     EQU  0x.....    ; Command table pointer
RESULT      EQU  0x.....    ; Result table pointer

;-------------------------------------------------------
;  Main
;-------------------------------------------------------
```


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

```
Initial
PUSH {R0-R2,R14}
                ; Push the register
                ; workspace and link
                ; register into stack
LDR R0,=COMMAND ; Set the pointers for
LDR R1,=RESULT  ; command and
                ; result tables

;-------------------------------------------------------
; Once the pointers are set, the command code and its
; respective parameters need to be stored in the command
; table. An example is provided below where the command
; code (54) for IAP command "Read Part ID" is stored into
; the command table
;-------------------------------------------------------

MOV R2,#0x36
STR R2, [R0]

;-------------------------------------------------------
; Please look below (after END) for description for how the
; IAP routine is called
;-------------------------------------------------------

BL jump_to_IAP

;-------------------------------------------------------
; At this point user has to analyze the result table and
; take action depending upon the status code returned by
; the IAP routine.(Code not shown)
;-------------------------------------------------------

POP{R0-R2,R3}
BX R3           ; Pop the link register
                ; contents and go back to
                 ; ARM mode

;-------------------------------------------------------
;Call IAP routine
;-------------------------------------------------------

jump_to_IAP
LDR R2,=IAP_ENTRY
BX R2
 END
```

The differences in the Thumb code as compared to the ARM code being the assembler directive CODE16 and the push and pop instructions for the stack.

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

4. Using the ARM Developer Suite (ADS 1.2) tools

There is one more way of calling the IAP routine using the symbol definitions (symdefs) file but this is specific to the ARM development tools. The IAP routine could be looked as an image residing in Flash. Now, an image residing in RAM can access the global symbols of this image residing in Flash using the symdefs file. The symdefs file can be considered to be an object file, which contains symbols and their values. Please refer to Chapter 4 in the ARM Developer Suite Linker and Utilities Guide for detailed information on accessing symbols.

The symdefs file could be defined as follows for the IAP routine.

```
#<SYMDEFS># ARM Linker, ADS1.2 [Build 805]: Last Updated: Fri Jun 06 15:46:24 2003
0x7ffffff0 T iap_entry
```

The first 11 characters #<SYMDEFS># of this text file recognizes this file as a symdefs file. We then provide the symbol information with regard to the IAP routine in the second line. This file could then be linked to user application using the –F option at command line for the ARM linker. To do this on the Metrowerks CodeWarrior, open the Debug settings window for the project, then click on ARM linker and then “Equivalent Command Line” could be seen (under Output tab) where the following option could be added:

```
-F C:\°\symdefs
```

where

symdefs is the symdefs file.

Once the symdefs file has been defined and added the to the project using the –F option, then in the user application the following needs to be done (Only C code is shown as an example):

Define data structure or pointers for IAP command table and result table

```
unsigned int command[5];
unsigned int result[2];

or

unsigned int * command;
unsigned int * result;
command=(unsigned int *) 0x……
result= (unsigned int *) 0x……
```

Call IAP routine

```
iap_entry(command,result);
```

As seen above, iap_entry does not have to be defined anywhere in the application, as the linker now knows it is been defined in the image residing in Flash through the symdefs file.

Remark: If the end-user application, which runs from SRAM, is ARM code, and if the IAP function is called, it may lead to an exception if the compiler/assembler options are not set properly. While compiling the code, user needs to instruct the compiler that the application

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

is going to make a call to Thumb code (IAP code residing in Flash is Thumb code). Most of the compilers/assemblers will have an option called ARM/Thumb interworking and this needs to be enabled by the end user.

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

5. 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.

6. 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: 25 October 2004
Document number: 9397 750 14073

Published in The U.S.A.