TEXAS INSTRUMENTS

TMS470 Family – F05 Flash Module Software Peripheral Driver User’s Specification

User's Guide

December 2005 TMS470 Catalog

SPNU257

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Copyright © 2005, Texas Instruments Incorporated

Contents

User's Guide
SPNU257–December 2005

TMS470 Family – F05 Flash Module Software Peripheral Driver User’s Specification

1 Objectives

The objective of this document is to define a set of software peripheral functions intended to program/erase a TMS470R1x Flash module.

This document is the specification of the **User interface**.

2 Predefined User Types and Names

2.1 UINT32, UINT16, UBYTE, and BOOL

The above types are defined in the Flash470.h header file as follows :

```
typedef int BOOL;
typedef unsigned char UBYTE;
typedef unsigned short int UINT16;
typedef unsigned long int UINT32;
```

The purpose of each type is as follows:

BOOL	Boolean variable (i.e. 1=TRUE, 0=FALSE)
UBYTE	Unsigned byte (8 bits wide)
UINT16	Unsigned 16 bit integer
UINT32	Unsigned 32 bit integer

This chapter summarizes the list of parameter types and names, i.e. enumerations used to define function parameters and their elements.

2.2 FLASH_ARRAY_ST

The above type is defined as follows in flash470.h:

```
typedef volatile UINT32 * FLASH_ARRAY_ST;
```

Thus a FLASH_ARRAY_ST is a volatile unsigned 32 bit integer array pointer. The control base address of the Flash module is declared as a FLASH_ARRAY_ST so that writing to and reading from registers is done in a volatile manner and all Flash control register offsets are defined as 32 bit offsets from this base address in the f05.h header file.

2.3 FLASH_STATUS_ST

The FLASH_STATUS_ST is used as a repository of information (pulse counts, PSA values, failing addresses and data, etc.) generated by a function in addition to the return value. A pointer to this type of structure is passed to most functions, and the contents of each element of the structure are dependent on the purpose of the function.

The FLASH_STATUS_ST structure is declared as follows in the flash470.h

```
typedef struct {
  UINT32 stat1;
  UINT32 stat2;
  UINT32 stat3;
  UINT32 stat4;
} FLASH_STATUS_ST;
```

The purpose of each element is as follows:

```
stat1                  Statistic 1: Meaning depends upon function.
stat2                  Statistic 2: Collects statistics on flash
stat3                  Statistic 3: operations such as the number of
stat4                  Statistic 4: pulses applied, the worst case
                                    number of pulses, address, etc.
```

2.4 FLASH_CORE

This type is an enumeration of the banks that can be use in the Flash module. Up to eight banks are supported.

```
FLASH_CORE0            Bank 0 selected
FLASH_CORE1            Bank 1 selected
FLASH_CORE2            Bank 2 selected
FLASH_CORE3            Bank 3 selected
FLASH_CORE4            Bank 4 selected
FLASH_CORE5            Bank 5 selected
FLASH_CORE6            Bank 6 selected
FLASH_CORE7            Bank 7 selected
```

2.5 FLASH_SECT

This type is an enumeration of the sector that can be chosen in a specified bank of the Flash module. In R1x F05, thirty-two sectors are supported

```
FLASH_SECT0          Sector 0 selected
FLASH_SECT1          Sector 1 selected
            .
            .
            .
FLASH_SECT31         Sector 31 selected
```

3 API Description

The Flash API is a library of routines which when called with the proper parameters in the proper sequence will erase, program or verify flash memory on the TMS470 family of Texas Instruments microcontrollers. These routines must be run in a privileged mode (mode other than user) to allow access to the flash control registers and to the interrupt disable bits. Most of the routines enter Flash Configuration Mode and therefore the system clock should not exceed 24MHz.

3.1 Pulse Size and Limits

	Pulse Length	**Maximum Number of Pulses**
Program	4 µS	50
Erase	2700 µS	4095
Compact	500 µs	2000

3.2 Build Environment

The current version of the Flash API library was built with version 4.30 of the IAR Workbench C compiler. Most of the functions are written in C with some critical functions coded in assembly language. Most of the code is in 32-bit instruction mode. The following compiler options were used to create the Flash API library:

Generate Interwork Code, Optimize for size (Maximum Optimization)

3.3 Interrupts

Interrupts are disabled during the PSA calculation. The number of cycles the interrupts are disabled is:

10+10w+x(7+6w)

Where w is the number of wait states and **x** is the number of words being checked by the PSA_U32 function. If a 64K byte sector is being checked at 20MHz with no wait states, the interrupts would be disabled for 5.7mS.

3.4 Function Size

The Flash API library functions are designed to be small enough that they can be loaded into RAM and executed on devices with as little as 2K bytes of RAM. The sizes listed below do not include the subroutines needed, but the required subroutines are noted for each function. Size is given in bytes. The code size assumes that the function is called from 16-bit instruction set mode. If the function is called from a 32-bit routine, add 24 bytes for the 32-bit instruction set veneer. Stack size does not include the argument list and the reserved word for dual state support in the calling function.

```
Function Name                Code + Shared + Const Size  Stack Size  Notes
=============                ==========================  ==========  =====
Feed_Watchdog_V()                 14 +  8                         0
Flash_API_Version_U16()            8 +  0                         0
Flash_Blank_B()                  228 +  8                        52
Flash_Compact_B()                102 + 24                        28       2,3
Flash_Erase_B()                  180 + 32                        40       2,3,5
Flash_Erase_Sector_B()           132 + 40                        36       1,2,5
Flash_Match_Key_B()              120 + 24                        20       1
Flash_Prog_B()                   264 + 40                        52       1,2,5
Flash_PSA_Calc_U32()             128 + 16                        28       4
Flash_PSA_Verify_B()             192 + 16                        44       4
Flash_Sector_Select_V()           12 +  8                         0
Flash_Track_Pulses_V()           136 + 24                        24       1
Flash_Verify_B()                 244 +  8 + 6                    60
init_state_machine()             180 +  8                        16
PSA_U32()                         60 +  0                         0

Key:
1 Requires Feed_Watchdog_V()
2 Requires init_state_machine()
3 Requires Flash_Track_Pulses_V()
4 Requires PSA_U32()
5 Requires Flash_Sector_Select_V()
```

4 API Functions

4.1 Feed_Watchdog_V

```
void Feed_Watchdog_V();
```

Parameters:

None

Return Value:

None

Description:

The purpose of this function is to allow feeding a watchdog during the program or erase functions or any function which performs a busy-wait on the Flash State Machine. This function can be replaced by a user defined watchdog function or a null function that returns without doing anything. The default function provided in the API library feeds the analog watchdog (AWD). This function is called at least every 200us. This function can not be executed from the bank which is being programmed or erased. See section 3.4 to determine which functions call this function.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/*******************************************************************
 *                                                                 *
 *  FILENAME: feed_dog.c                                           *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved *
 *                                                                 *
 *  This function is used to feed the watchdog timer to prevent    *
 *  a reset                                                        *
 *                                                                 *
 *******************************************************************/
void Feed_Watchdog_V()
{
  *(volatile UINT32 *)0xffffff0c=0xe5;
  *(volatile UINT32 *)0xffffff0c=0xa3;
}
```

4.2 Flash_API_Version_U16

```
UINT16 Flash_API_Version_U16(void)
```

Parameters:

None

Return Value:

Unsigned 16 bit integer (UINT16

Description:

This function returns the programming algorithm version number in BCD (Binary Coded Decimal) notation. For example, the number 0x0031 represents version 0.31.

Source Listing (IAR ARM7 Assembly):

```
#include "fapiver.h"
        MODULE f_ver
        RSEG CODE:NOROOT(2)
        PUBLIC __Flash_API_Version_U16
        CODE32
__Flash_API_Version_U16:
        MOV R0,#con0
        MOV       PC,LR
con0   EQU fapiver
        ENDMOD
        END
```

4.3 Flash_Blank_B

```
BOOL Flash_Blank_B (UINT32 *start,

                    UINT32 length,
                    FLASH_CORE core,
                    FLASH_ARRAY_ST cntl,
                    FLASH_STATUS_ST *status
                    );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Pointer to the first word in the flash that will be blank-checked
length	UINT32	Number of 32-bit words to be blank-checked
core	FLASH_CORE	Bank select (0-7) of region of flash being blank checked
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash region resides.
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information.

Return Value:

This function returns a Boolean value. Pass = 1 = Region is blank, Fail = 0 = Region is not blank (i.e. a location that reads as something other than 0xFFFFFFFF was found). If the function returns Fail, the following values are stored in the FLASH_STATUS_ST structure:

stat1 Address of first non-blank location

stat2 Data read at first non-blank location

stat3 Value of compare data (always 0xFFFFFFFF)

stat4 Mode in which first fail occurred. If this value is 0x00000000, then the failure was seen in normal read mode, but read margin 1 read was not performed. If this value is 0x00000008, then the region was read as blank in normal read mode, but did not read as blank in read margin 1 mode.

Description:

This function verifies that the flash has been properly erased by using the normal read and read-margin 1 modes starting from the address passed in the parameter “start”. “length” words are read, starting at the starting address. The area specified for blank check must be within a single bank specified by “core”. If the array contains multiple banks, Flash_Blank_B must be called separately for each bank to be blank checked. Regions may cross sector boundaries, as long as the sectors all reside in the same bank.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/********************************************************************
 *                                                                  *
 *  FILENAME: blank.c                                               *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved  *
 *                                                                  *
 *     Blank Check Algorithm                                        *
 *                                                                  *
 *     Returns 1 if blank check passes                              *
 *     Returns 0 if blank check fails                               *
 *                                                                  *
 ********************************************************************/
BOOL Flash_Blank_B(UINT32 *start,
                   UINT32 length,
                   FLASH_CORE core,
                   FLASH_ARRAY_ST cntl,
                   FLASH_STATUS_ST *status
                   )
{
  volatile UINT32 *dest_addr;            /* Address pointer to flash location */
  UINT32 i,j,k,result=TRUE,sm,sw;
  UBYTE m[2]={READMODE,RDM1MODE};        /* read modes */
  UBYTE w[2]={0x00,0x11};                /* wait states */

  SET_CFG_BIT;
  sm=cntl[REGOPT];                       /* save current read mode */
  cntl[REGOPT]=READMODE;                 /* force normal read mode */
  cntl[MAC2]=(cntl[MAC2]&~0x7)|core;     /* enable the appropriate core */
  cntl[TCR]=0x2fc0;                      /* 0x2fc0 = TCR=0 */
  cntl[DPTR]=0xb;                        /* clear TEZ */
  sw=cntl[BAC2];                         /* save current wait states */
  cntl[BAC2]=(sw|0xff);                  /* force max wait states */
  for (i=0;(i<2)&&(result==TRUE);++i) {  /* loop through read modes */
    cntl[REGOPT]=m[i];                   /* Set read mode */
    dest_addr=start;                     /* Initialize programmer pointer */
    cntl[BAC2]=(sw&~0xff)|w[i];          /* set wait states */
    for (k=0;k<length;k++) {             /* step through each flash location */
      READ_FLASH_MEM_U32(dest_addr,j);   /* read data from flash */
      if (j!=0xffffffff) break;          /* break if compare fails */
      dest_addr++;                       /* increment address */
    }
    if (k!=length) {
      status->stat1=(UINT32)dest_addr;   /* save last address read */
      status->stat2=j;                   /* save actual data */
      status->stat3=0xffffffff;          /* save expected data */
      status->stat4=m[i];                /* save read mode */
      result=FALSE;
    }
  }
  cntl[DPTR]=0xf;                        /* reset TEZ */
  cntl[BAC2]=sw;                         /* reset wait states */
  cntl[REGOPT]=sm;                       /* reset read mode */
  CLR_CFG_BIT;
  return result;
} /* end of blank function */
```

4.4 Flash_Compact_B

```
BOOL Flash_Compact_B(UINT32 *start,

                     FLASH_CORE core,
                     FLASH_SECT sector,
                     UINT32 delay,
                     FLASH_ARRAY_ST cntl,
                     FLASH_STATUS_ST *status
                     );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Points to first word in the flash sector which will be compacted. Note: This must correspond to the first address in the sector.
core	FLASH_CORE	Bank select (0-7) of sector being compacted
sector	FLASH_SECTOR	Sector select (0-31) of sector being compacted
delay	UNIT32	From Table 1, Flash Delay Parameter Values in section 5 of this document
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash region resides.
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information.

Return Value:

This function returns a boolean value. Pass = 1, Fail = 0. In addition, the following values are stored in the FLASH_STATUS_ST structure:

stat1 Not used

stat2 Final value of MSTAT register (see section 4.10 for description of MSTAT register)

stat3 Total number of compaction pulses executed for all sticks.

stat4 Maximum number of compaction pulses for any one stick (One stick = 16 columns)

Description:

This function adjusts depleted (over-erased) flash memory bits in the target sector so they are not in depletion. This function only performs compaction on the target sector, so compaction of N sectors requires N calls to Flash_Compact_B() with each call having the proper core, start, and cntl address information passed to it.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/*********************************************************************
 *                                                                   *
 *  FILENAME: compact.c                                              *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved   *
 *                                                                   *
 *     Compaction Algorithm                                          *
 *                                                                   *
 *     Returns 1 if compaction completed                             *
 *     Returns 0 if compaction failed                                *
 *                                                                   *
 *********************************************************************/
BOOL Flash_Compact_B(UINT32 *start,
                     FLASH_CORE core,
                     FLASH_SECT sector,
                     UINT32 delay,
                     FLASH_ARRAY_ST cntl,
                     FLASH_STATUS_ST *status
                     )
{
  UINT32 save;
  SET_CFG_BIT;
  save=cntl[REGOPT];                   /* save initial mode */
  cntl[REGOPT]=READMODE;               /* force normal read mode */
  init_state_machine(core,delay,cntl); /* initialize */
  *(volatile UINT16 *)start=0x40;      /* clear status register */
  *(volatile UINT16 *)start=0x1000;    /* Validate Sector command */
  *(volatile UINT16 *)start=0xffff;    /* Dummy data value */
  Flash_Track_Pulses_V(cntl,status);   /* track compaction pulses */
  cntl[REGOPT]=save;                   /* restore mode */
  CLR_CFG_BIT;
  /* The below is to preserve backwards compatibility!! */
  status->stat3=status->stat4;         /* Total Compaction Pulses */
  status->stat4=status->stat1;         /* Max Compaction pulses */
  status->stat1=0;                     /* not used */
  return (status->stat2==0);           /* pass if no bits set in MSTAT */;
} /* end of compaction function */
```

4.5 Flash_Erase_B

```
BOOL  Flash_Erase_B(UINT32 *start,
                  UINT32 length,
                  FLASH_CORE core,
                  FLASH_SECT sector,
                  UINT32 delay,
                  FLASH_ARRAY_ST cntl,
                  FLASH_STATUS_ST *status
                  );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Points to first word in the flash sector which will be erased. Note: This must correspond to the first address in the sector
length	UINT32	Number of 32 bit words in the sector.
core	FLASH_CORE	Bank select (0-7) of sector being erased
sector	FLASH_SECTOR	Sector select (0-31) of sector being erased
delay	UINT32	From Table 1, Flash Delay Parameter Values in section 5 of this document.
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash sector resides.
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information. Note: Determining a sector is blank using Flash_Blank_B is insufficient to make sure a sector is sufficiently erased, especially if a previous erase was interrupted by a power glitch or some other user intervention, because Flash_Blank_B is unable to detect depleted bits or bits that are marginal to a read margin 1 read. To ensure a sector is adequately erased, erase must be performed on the sector (regardless of the contents). If the sector has already been determined to be blank using Flash_Blank_B prior to erase, the status->stat1 element can be initialized to 0x12345678 prior to calling Flash_Erase_B to disable preconditioning and speed up erase. Non-blank sectors by default should have the value of status->stat1 initialized to 0x00000000 (or some value other than 0x12345678) to make sure preconditioning is enabled during erase. *See Recommended Erase Flows in section 5 for more information.*

Return Value:

This function returns a Boolean value. Pass = 1, Fail = 0. In addition, the following values are stored in the FLASH_STATUS_ST structure:

stat1 Maximum number of compaction pulses for any one stick (16 columns)

stat2 Status register value (see Flash_Erase_Status_U16 for description)

stat3 Number of pulses applied to erase all locations

stat4 Total number of compaction pulses

Description:

This function is used to apply programming, erase, and compaction pulses to the targeted sector until it is completely erased. and to collect pulse count information. This function is unique from Flash_Erase_Sector_B() in that it allows for the disabling of preconditioning during erase (see above table). The length parameter and sector numbers are only provided to maintain the same interface as the F10 API, as they are ignored by the function. The F05 Flash State Machine only requires a correct core and start address value to correctly erase a sector. See section 6.1 for recommended Erase flow guidelines.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/********************************************************************
 *                                                                  *
 *  FILENAME: erase.c                                               *
 *  Copyright (c) Texas instruments 2000-2005, All Rights Reserved  *
 *                                                                  *
 *     Erase Algorithm with or w/o Precondition step                *
 *                                                                  *
 *     If status->stat1==0x12345678, precondition not performed     *
 *     Returns 1 if erase completed                                 *
 *     Returns 0 if erase failed                                    *
 *                                                                  *
 ********************************************************************/
BOOL Flash_Erase_B(UINT32 *start,
                   UINT32 length,
                   FLASH_CORE core,
                   FLASH_SECT sector,
                   UINT32 delay,
                   FLASH_ARRAY_ST cntl,
                   FLASH_STATUS_ST *status
                   )
{
  UINT32 j,k,noprcnd=status->stat1;       /* noprcnd = no precondition flag */
  if (delay>MAX_DELAY) {
    status->stat2=0xff;                   /* Error - delay value too high */
  } else {
    SET_CFG_BIT;
    k=cntl[REGOPT];                       /* save initial mode */
    cntl[REGOPT]=READMODE;                /* force normal read mode */
    init_state_machine(core,delay,cntl);  /* initialize for erase */
    Flash_Sector_Select_V(core,cntl);
    if (noprcnd==0x12345678) {
      j=cntl[MAXPP];                      /* save original MAXPP for later restore */
      cntl[MAXPP]=(j&0xf000);             /* force zero programming pulses */
    }
    *(volatile UINT16 *)start=0x40;       /* clear status register */
    *(volatile UINT16 *)start=0x20;       /* Erase command */
    *(volatile UINT16 *)start=0xffff;     /* Dummy data value */
    Flash_Track_Pulses_V(cntl,status);    /* track erase and compaction pulses */
    cntl[BSEA]=0x0;                       /* Protect all sectors */
    cntl[BSEB]=0x0;                       /* Protect all sectors */
    if (noprcnd==0x12345678) {
      cntl[MAXPP]=j;                      /* restore original MAXPP */
    }
    cntl[REGOPT]=k;                       /* restore mode */
    CLR_CFG_BIT;
  }
  return (status->stat2==0);             /* pass if no bits set in MSTAT */
} /* end of erase function */
```

4.6 Flash_Erase_Sector_B

```
BOOL  Flash_Erase_Sector_B(UINT32  *start,
                      FLASH_CORE  core,
                      FLASH_SECT  sector,
                      FLASH_ARRAY_ST  cntl,
                      UINT32  delay,
                      FLASH_STATUS_ST  *status
                      );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Points to first word in the flash sector which will be erased. Note: This must correspond to the first address in the sector
length	UINT32	Number of 32 bit words in the sector.
core	FLASH_CORE	Bank select (0-7) of sector being erased
sector	FLASH_SECTOR	Sector select (0-31) of sector being erased
delay	UINT32	From Table 1, Flash Delay Parameter Values in section 5 of this document.
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash sector resides.

Return Value:

This function returns a Boolean value. Pass = 1, Fail = 0.

Description:

This function erases a sector. The length parameter and sector number are not used in F05. See section 6.1 for recommended Erase flow guidelines.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/********************************************************************
 *                                                                  *
 *  FILENAME: esector.c                                             *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved  *
 *                                                                  *
 *     Erase Sector Algorithm without pulse tracking                *
 *                                                                  *
 *     Returns 1 if erase completed                                 *
 *     Returns 0 if erase failed                                    *
 *                                                                  *
 ********************************************************************/
BOOL Flash_Erase_Sector_B(UINT32 *start,
                          UINT32 length,
                          FLASH_CORE core,
                          FLASH_SECT sector,
                          UINT32 delay,
                          FLASH_ARRAY_ST cntl
                          )
{
  int result=FALSE,k;
  if (delay<=MAX_DELAY) {                /* fail if delay value too high */
    SET_CFG_BIT;
    k=cntl[REGOPT];                      /* save initial mode */
    cntl[REGOPT]=READMODE;               /* force normal read mode */
    init_state_machine(core,delay,cntl);
    Flash_Sector_Select_V(core,cntl);
    *(volatile UINT16 *)start=0x40;      /* clear status register */
    *(volatile UINT16 *)start=0x20;      /* Erase command */
    *(volatile UINT16 *)start=0xffff;
    do {                                 /* while waiting for program to finish */
      Feed_Watchdog_V();                 /* feed watchdog timer */
    } while (cntl[MSTAT]&0x100);         /* until BUSY bit goes low */
    if ((cntl[MSTAT]&0x3ff)==0) {
      result=TRUE;                       /* success if no error bits set */
    }
    cntl[BSEA]=0x0;                      /* Protect all sectors */
    cntl[BSEB]=0x0;                      /* Protect all sectors */
    cntl[REGOPT]=k;                      /* restore mode */
    CLR_CFG_BIT;
  }
  return result;
} /* end of erase function */
```

4.7 Flash_Match_Key_B

```
BOOL Flash_Match_Key_B(volatile UINT32 *key_start,
                    const UINT32 key[],
                    FLASH_ARRAY_ST cntl
                    );
```

Parameters:

Parameter	Type	Purpose
key_start	volatile UINT32	Pointer to first key in flash array - this is usually the 4th word from the end of the 1st sector in the first bank of the Flash module.
key	const UINT32 [ ]	Pointer to an array of four keys to match against the protection keys in flash
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash sector resides.

Return Value:

If the keys are successfully matched this routine returns TRUE. Failure to match the keys returns FALSE

Description:

This function applies only to devices with flash protection keys. This function will attempt to match the values in key[ ] with the current protection key values, starting at the address key_start. A successful match must occur before any programming, erasing or verifying of the flash array can be executed.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/*******************************************************************
 *                                                                 *
 *  FILENAME: Match_Key_B.c                                        *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved *
 *                                                                 *
 *  This function is used to match the key values in the           *
 *  protection keys before an erase/program/read of the Flash      *
 *  can be performed.  It returns true if the keys read out        *
 *  match the expected keys, in which case the match was           *
 *  successful                                                     *
 *                                                                 *
 *******************************************************************/
BOOL Flash_Match_Key_B(volatile UINT32 *key_start,
                       const UINT32 key[],
                       FLASH_ARRAY_ST cntl
                       )
{
  int k,sm,sw;

  SET_CFG_BIT;
  k=cntl[MSTAT];                      /* retrieve MSTAT value */
  if (!(k&0x8)) {                     /* if 3VSTAT bit not set */
    cntl[MAC2]=(cntl[MAC2]&~0x7);     /* select bank 0 */
    do {
      Feed_Watchdog_V();              /* feed the dog */
    } while (!(cntl[BPTR]&0x200));    /* while waiting for pump ready */
    sw=cntl[BAC2];                    /* save current wait states */
    cntl[BAC2]=(sw|0xff);             /* force max wait states */
    sm=cntl[REGOPT];                  /* save current mode */
    cntl[REGOPT]=READMODE;            /* force normal read mode */
    for (k=0;k<4;k++) {
      key_start[k];
      cntl[PROTKEY]=key[k];
    }
    k=(cntl[BBUSY]&0x8000)>>15;       /* retrieve result */
    cntl[REGOPT]=sm;                  /* restore read mode */
    cntl[BAC2]=sw;                    /* restore wait states */
  }
  CLR_CFG_BIT;
  return k;                           /* return value */
}
```

4.8 Flash_Prog_B

```
BOOL Flash_Prog_B (UINT32 *start,

                     UINT32 *buffer,
                     UINT32 length,
                     FLASH_CORE core,
                     UINT32 delay,
                     FLASH_ARRAY_ST cntl,
                     FLASH_STATUS_ST *status
                     );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in the flash which will be programmed
buffer	UINT32 *	Pointer to the starting address of a buffer with data to program
length	UINT32	Number of 32 bit words to program.
core	FLASH_CORE	Bank select (0-7) of region being programmed.
delay	UINT32	From Table 1, Flash Delay Parameter Values in section 6 of this document.
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash sector resides.
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information.

Return Value:

This function returns a Boolean value. Pass = 1, Fail = 0. In addition, the following values are stored in the FLASH_STATUS_ST structure:

stat1 If programming failed, address of first failing location

stat2 If programming failed, data at first failing location

stat3 If programming failed, the last MSTAT value, otherwise the total number of pulses applied to program all locations

stat4 Maximum number of pulses required to program a single location

Description:

This function will program the flash from the starting address “start” for “length” 32 bit words. This function will only program the flash; it will not erase the flash array first. The user code must make sure that the areas to be programmed are already erased before calling this routine. The function Flash_Erase_Sector_B or Flash_Erase_B can be used to erase an entire sector. See section 6.2 for more details on programming flow guidelines.

The program routine will program the data that is stored in the buffer pointed to by “buffer”. Care must be taken to make sure that the data to be programmed does not cross a bank boundary.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/*********************************************************************
```

```
 *                                                                  *
 *  FILENAME: prog.c                                                *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved  *
 *                                                                  *
 *     Programming Algorithm                                        *
 *                                                                  *
 *     Returns 1 if program completed                               *
 *     Returns 0 if program failed                                  *
 *                                                                  *
 ********************************************************************/
BOOL Flash_Prog_B(UINT32 *start,
                  UINT32 *buff,
                  UINT32 length,
                  FLASH_CORE core,
                  UINT32 delay,
                  FLASH_ARRAY_ST cntl,
                  FLASH_STATUS_ST *status
                  )
{
  volatile UINT16 *cadr=(UINT16 *)start;
  UINT16 retries=0,maxpp;
  UINT32 i,j,k,mstat,result=TRUE;
  status->stat3=0;                      /* initialize total number of pulses */
  status->stat4=0;                      /* initialize max pulses */
  length<<=1;                           /* double word length due to 16 bit programming */
  SET_CFG_BIT;
  init_state_machine(core,delay,cntl);  /* initialize state machine */
  i=cntl[BAC2];                         /* save current wait states */
  cntl[BAC2]=(i|0xff);                  /* force max wait states */
  j=cntl[REGOPT];                       /* save initial mode */
  cntl[REGOPT]=READMODE;                /* force normal read mode */
  Flash_Sector_Select_V(core,cntl);     /* select core and enable sectors */
  maxpp=cntl[MAXPP];                    /* retrieve max allowed programming pulses */
  for (k=0;k<length;k++) {              /* Step through each flash location */
    *cadr=0x40;                         /* clear status register */
    *cadr=0x10;                         /* Program command */
    *cadr++=*((UINT16 *)buff+k);        /* data to be programmed */
    do {                                /* while waiting for program to finish */
      Feed_Watchdog_V();                /* feed watchdog timer */
      mstat=cntl[MSTAT];                /* retrieve MSTAT */
    } while (mstat&0x100);              /* until BUSY bit goes low */
    mstat&=0x3ff;                       /* keep only bits of interest */
    retries=maxpp-cntl[PULSE_COUNTER];  /* calculate retries */
    if (retries>status->stat4) {        /* if latest count is max */
      status->stat4=retries;            /* update max pulses */
    }
    status->stat3+=retries;             /* update total pulses */
    if ((retries==maxpp)                /* if max pulses reached */
        ||(mstat!=0)                    /* or MSTAT non-zero */
        ) {
      status->stat1=(UINT32)(--cadr);   /* retrieve last 32 bit address */
      status->stat2=*cadr;              /* retrieve last data */
      status->stat3=(mstat<<1);         /* retrieve last MSTAT value */
      result=FALSE;                     /* set fail flag */
      k=length;                         /* exit loop */
    }
  } /* end of for */
  cntl[BSEA]=0x0;                       /* Protect all sectors */
  cntl[BSEB]=0x0;                       /* Protect all sectors */
  cntl[BAC2]=i;                         /* restore wait states */
  cntl[REGOPT]=j;                       /* restore mode */
  CLR_CFG_BIT;
  status->stat3>>=1;                    /* div by 2 since prog 16-bits at a time */
  return result;
} /* end of program function */
```

4.9 Flash_PSA_Calc_U32

```
UINT32  Flash_PSA_Calc_U32(UINT32  *start,
                           UINT32  length,
                           UINT32  psa_seed,
                           UINT32  mode,
                           FLASH_CORE  core,
                           FLASH_ARRAY_ST  cntl
                           );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in the flash which will be read
length	UINT32	Number of 32 bit words to read.
psa_seed	UINT32	The initial value of the PSA calculation. If this is the first region being read, then this value should be 0x00000000. If this region is a continuation of a previous region, then the resulting PSA of the previous region should be used as the seed value.
Mode	UINT32	The read mode in which to do the PSA calculation 0 - Normal read mode 1 - Read Margin 0 mode 2 - Read Margin 1 mode
Core	FLASH_CORE	Bank select (0-7) of region being read.
Cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash region resides.

Return Value:

This function returns an unsigned 32-bit integer that is the computed PSA value.

Description:

This function will use the PSA_U32 function to calculate a 32 bit PSA checksum in the given read mode for a given region of flash defined by the start address and length in 32 bit words. The seed value for the PSA calculation is provided by the psa_seed value.

This function can be used for checking all of the flash on a device in a read margin mode against a single PSA value. Set the seed to zero for the calculation on the first region and then provide the result of the first region calculation as the seed for the calculation of the next region (Note: Bank boundaries must not be crossed by any single region). Compare the result of the final region's calculation against the known PSA value for the entire flash.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/********************************************************************
 *                                                                  *
 *  FILENAME: psa_calc.c                                            *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved  *
 *                                                                  *
 *     PSA Calculation Algorithm                                    *
 *                                                                  *
 *     Generates a 32 bit PSA for the given region of Flash         *
 *                                                                  *
 ********************************************************************/
UINT32 Flash_PSA_Calc_U32(UINT32 *start,
                          UINT32 length,
                          UINT32 psa_seed,
                          UINT32 mode,
                          FLASH_CORE core,
                          FLASH_ARRAY_ST cntl
                          )
```

```
{
  UINT32 sm,sw,psa;
  SET_CFG_BIT;
  sm=cntl[REGOPT];                    /* save initial mode */
  if (mode==1) {
    cntl[REGOPT]=RDM0MODE;            /* set read margin 0 mode */
  } else if (mode==2) {
    cntl[REGOPT]=RDM1MODE;            /* set read margin 1 mode */
  } else {
    cntl[REGOPT]=READMODE;            /* set normal read mode */
  }
  cntl[MAC2]=(cntl[MAC2]&~0x7)|core;  /* enable the appropriate core */
  cntl[TCR]=0x2fc0;                   /* 0x2fc0 = TCR=0 */
  cntl[DPTR]=0xb;                     /* clear TEZ */
  sw=cntl[BAC2];                      /* save current wait states */
  cntl[BAC2]=(sw|0xff);               /* set wait states */
  psa=PSA_U32(start,length,psa_seed); /* read to generate PSA */
  cntl[DPTR]=0xf;                     /* reset TEZ */
  cntl[BAC2]=sw;                      /* reset wait states */
  cntl[REGOPT]=sm;                    /* reset read mode */
  CLR_CFG_BIT;
  return psa;
} /* end of psa calc function */
```

4.10 Flash_PSA_Verify_B

```
BOOL  Flash_PSA_Verify_B(UINT32  *start,

                         UINT32  length,
                         UINT32  psa,
                         FLASH_CORE  core,
                         FLASH_ARRAY_ST  cntl,
                         FLASH_STATUS_ST  *status
                         );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in the flash which will be verified.
length	UINT32	Number of 32 bit words to be verified using PSA.
psa	UINT32	The expected PSA value against which the actual PSA values will be compared.
core	FLASH_CORE	Bank select (0-7) of region being read.
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash region resides.
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information.

Return Value:

This function returns a Boolean value. Pass = 1, Fail = 0 In addition the following values are stored in the FLASH_STATUS_ST structure:

stat1	Actual PSA for read-margin 0
stat2	Actual PSA for read-margin 1
stat3	Actual PSA for normal read
stat4	Unused

Description:

This function will verify proper programming by using normal read, read-margin 0, and read-margin 1 modes, and generating a 32 bit PSA checksum for the data in the region in each mode. Verification will start from the start address “start” and check “length” words from the start address. The area specified for PSA verify must be within the bank specified by “core” and the control register should be passed in as “cntl”.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/********************************************************************
 *                                                                  *
 *  FILENAME: verify_psa.c                                          *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved  *
 *                                                                  *
 *     PSA Verify Algorithm                                         *
 *                                                                  *
 *     Returns 1 if verify passes                                   *
 *     Returns 0 if verify fails                                    *
 *                                                                  *
 ********************************************************************/
BOOL Flash_PSA_Verify_B(UINT32 *start,
                        UINT32 length,
                        UINT32 psa,
                        FLASH_CORE core,
                        FLASH_ARRAY_ST cntl,
                        FLASH_STATUS_ST *status
                        )
{
  UINT32 sm,sw;
  SET_CFG_BIT;
  sm=cntl[REGOPT];                       /* save current read mode */
  cntl[REGOPT]=READMODE;                 /* set READMODE */
  cntl[MAC2]=(cntl[MAC2]&~0x7)|core;     /* enable the appropriate core */
  cntl[TCR]=0x2fc0;                      /* 0x2fc0 = TCR=0 */
  cntl[DPTR]=0xb;                        /* clear TEZ */
  sw=cntl[BAC2];                         /* save current wait states */
  cntl[BAC2]=(sw&~0xff);                 /* set wait states */
  status->stat3=PSA_U32(start,length,0); /* read PSA */
  cntl[REGOPT]=RDM0MODE;                 /* set RDM0MODE */
  cntl[BAC2]=(sw|0xff);                  /* set wait states */
  status->stat1=PSA_U32(start,length,0); /* read PSA */
  cntl[REGOPT]=RDM1MODE;                 /* set RDM1MODE */
  cntl[BAC2]=(sw&~0xff)|0x11;            /* set wait states */
  status->stat2=PSA_U32(start,length,0); /* read PSA */
  cntl[DPTR]=0xf;                        /* reset TEZ */
  cntl[BAC2]=sw;                         /* reset wait states */
  cntl[REGOPT]=sm;                       /* reset read mode */
  CLR_CFG_BIT;
  return ((status->stat1==psa)           /* normal read PSA matches */
          &&(status->stat2==psa)         /* read margin 0 PSA matches */
          &&(status->stat3==psa)         /* read margin 1 PSA matches */
          );
} /* end of psa verify function */
```

4.11 Flash_Sector_Select_V

```
void Flash_Sector_Select_V(FLASH_CORE core

                           FLASH_ARRAY_ST cntl
                           );
```

Parameters:

Parameter	Type	Purpose
core	FLASH_CORE	Bank select (0-7) of sectors to be selected.
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash sector resides.

Return Value:

This function has no return value:

Description:

This function is called by Flash_Compact_B(), Flash_Erase_B(), Flash_Erase_Sector_B(), Flash_Zeros_B(), Flash_Start_Erase_B() and Flash_Prog_B(). The purpose of this function is to allow only certain sections of a bank to be programmed or erased. This function can be replaced by a user-defined function that will disable programming and erasing of certain sectors. The default function provided in the API library enables all sectors of all banks. This function may be executed from the bank that is being programmed or erased.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/*********************************************************************
 *                                                                   *
 *  FILENAME: sector_select.c                                        *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved  *
 *                                                                   *
 *     This function enables all sectors in a flash core             *
 *                                                                   *
 *********************************************************************/
void Flash_Sector_Select_V(FLASH_CORE core,
                           FLASH_ARRAY_ST cntl
                           )
{
  cntl[BSEA]=0xffff; /* Enable sectors  0-15 */
  cntl[BSEB]=0xffff; /* Enable sectors 16-31 */
}
```

4.12 Flash_Track_Pulses_V

```
void  Flash_Track_Pulses_V(FLASH_ARRAY_ST  cntl,
                           FLASH_STATUS_ST  *status
                           );
```

Parameters:

Parameter	Type	Purpose
Cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash region resides.
Status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information.

Return Value:

This function returns no value, but the following values are stored in the FLASH_STATUS_ST structure:

stat1	Max Compaction pulses for the worst case stick in the sector.
stat2	Final MSTAT value
stat3	Max Erase Pulses executed to erase the sector (if applicable)
stat4	Total Compaction Pulses executed on the sector

Description:

This function tracks erase and/or compaction pulse counts after the erase or compaction commands have been issued to the state machine. This function is called by Flash_Erase_B and Flash_Compact_B. The pulse counts are only ensured to be accurate if the default Feed_Watchdog_V is implemented or a version of Feed_Watchdog_V that takes less than 500us to return is implemented.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/*********************************************************************
 *                                                                   *
 *  FILENAME: track_pulses.c                                         *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved   *
 *                                                                   *
 *     Algorithm to track erase pulses in the FSM                    *
 *     This function assumes that an erase or compaction             *
 *     command has been issued and that the config bit is enabled   *
 *                                                                   *
 *     The values in the status structure are updated as follows:   *
 *       status->stat1 => Max Compaction pulses                      *
 *       status->stat2 => final MSTAT value                          *
 *       status->stat3 => Max Erase Pulses                           *
 *       status->stat4 => Total Compaction Pulses                    *
 *                                                                   *
 *********************************************************************/
void Flash_Track_Pulses_V(FLASH_ARRAY_ST cntl,
                FLASH_STATUS_ST *status
                )
{
  UINT32 k,state,cc=0,ladr=0xFFFF,cadr;
  status->stat1=0;                          /* Max Compaction pulses */
  status->stat2=0;                          /* final MSTAT value */
  status->stat3=0;                          /* Max Erase Pulses */
  status->stat4=0;                          /* Total Compaction Pulses */
  do {
    state=(cntl[WSMREG]&0x7c);              /* retrieve current state */
    if (state==CMPCT_PULSE_ACTIVE) {        /* Compaction pulse */
      status->stat4++;                      /* increment total compaction pulse count */
      cadr=(cntl[ADDRCTRLOW]&0xffff);       /* retrieve lower bits of current address */
      if (cadr!=ladr) {                     /* if new address */
      cc=0;                               /* reset current count */
      ladr=cadr;                          /* update current address */
      }
      if (++cc>=status->stat1) {            /* update current and max pulse counts */
      status->stat1=cc;                   /* update max count */
      }
    } else if (state==ERASE_PULSE_ACTIVE) { /* Erase pulse */
      status->stat3++;                      /* increment max erase pulse count */
    } else {
      state=0xffffffff;                     /* force incorrect state for do loop */
    }
    do {
      Feed_Watchdog_V();                    /* feed watchdog */
    } while ((cntl[WSMREG]&0x7c)==state);   /* while current state matches */
    k=cntl[MSTAT];                          /* retrieve current MSTAT */
  } while (k&0x100);                        /* wait till BUSY bit low */
  status->stat2=(k&0x3ff);                  /* update final MSTAT value */
} /* end of track pulses function */
```

4.13 Flash_Verify_B

```
BOOL Flash_Verify_B(UINT32 *start,

                    UINT32 *buffer,
                    UINT32 length,
                    FLASH_CORE core,
                    FLASH_ARRAY_ST cntl,
                    FLASH_STATUS_ST *status
                    );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in the flash to be verified.
buffer	UINT32 *	Pointer to the starting address of buffer with data to verify against.
length	UINT32	Number of 32 bit words to be verified.
core	FLASH_CORE	Bank select (0-7) of region being read.
cntl	FLASH_ARRAY_ST	Flash Control Base address of module in which flash region resides.
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information.

Return Value:

This function returns a Boolean value. Pass = 1, Fail = 0. In addition, the following values are stored in the FLASH_STATUS_ST structure (only on failure):

stat1	address of first location failing verify
stat2	data at first location failing verify
stat3	data expected at failing address
stat4	value of 0 means normal read failure, 4 means read margin zero failure, and 8 means read margin one failure

Description:

This function will verify proper programming by using read-margin 0 and read-margin 1 modes. Verification will start from the start address “start” and check “length” words from the start address.

The area specified for verify must be within the bank specified by “core” and the control register should be passed in as “cntl”. The verify routine will compare the data stored in the flash to the data stored in the buffer pointed to by “buffer”.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/*********************************************************************
 *                                                                   *
 *  FILENAME: verify.c                                               *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved  *
 *                                                                   *
 *     Verify Algorithm                                              *
 *                                                                   *
 *     Returns 1 if verify passes                                    *
 *     Returns 0 if verify fails                                     *
 *                                                                   *
 *********************************************************************/
BOOL Flash_Verify_B(UINT32 *start,
                    UINT32 *buff,
                    UINT32 length,
                    FLASH_CORE core,
                    FLASH_ARRAY_ST cntl,
                    FLASH_STATUS_ST *status
                    )
{
  volatile UINT32 *dest_addr;            /* Address pointer to flash location */
  UINT32 i,j,k,result=TRUE,sm,sw;
  UBYTE m[3]={READMODE,
              RDM0MODE,
              RDM1MODE};                 /* read modes */
  UBYTE w[3]={0x00,0xff,0x11};           /* wait states */

  SET_CFG_BIT;
  sm=cntl[REGOPT];                       /* save current read mode */
  cntl[REGOPT]=READMODE;                 /* force normal read mode */
  cntl[MAC2]=(cntl[MAC2]&~0x7)|core;     /* enable the appropriate core */
  cntl[TCR]=0x2fc0;                      /* 0x2fc0 = TCR=0 */
  cntl[DPTR]=0xb;                        /* clear TEZ */
  sw=cntl[BAC2];                         /* save current wait states */
  cntl[BAC2]=(sw|0xff);                  /* force max wait states */
  for (i=0;(i<3)&&(result==TRUE);++i) {  /* loop through read modes */
    cntl[REGOPT]=m[i];                   /* Set read mode */
    dest_addr=start;                     /* Initialize programmer pointer */
    cntl[BAC2]=(sw&~0xff)|w[i];          /* set wait states */
    for (k=0;k<length;k++) {             /* step through each flash location */
      READ_FLASH_MEM_U32(dest_addr,j);   /* read the data from flash */
      if (j!=buff[k]) break;             /* break if compare fails */
      dest_addr++;                       /* increment address */
    }
    if (k!=length) {
      status->stat1=(UINT32)dest_addr;   /* save last address read */
      status->stat2=j;                   /* save actual data */
      status->stat3=buff[k];             /* save expected data */
      status->stat4=m[i];                /* save read mode */
      result=FALSE;
    }
  }
  cntl[DPTR]=0xf;                        /* reset TEZ */
  cntl[BAC2]=sw;                         /* reset wait states */
  cntl[REGOPT]=sm;                       /* reset read mode */
  CLR_CFG_BIT;
  return result;
} /* end of verify function */
```

4.14 init_state_machine

```
void  init_state_machine(FLASH_CORE  core,
                   UINT32  delay,
                   FLASH_ARRAY_ST  cntl
                   );
```

Parameters:

Parameter	Type	Purpose
core	FLASH_CORE	Bank select (0-7) of bank being initialized.
delay	UINT32	From Table 1, Flash Delay Parameter Values in section 5 of this document.
cntl	FLASH_ARRAY_ST	Flash Control Base address of flash module being initialized.

Return Value:

This function does not return any value,

Description:

This function is called by several of the Flash API functions to initialize the F05 state machine before performing flash program or erase operation. It does not need to be called directly by the user. This function may be executed from the bank that is being erased or programmed.

Source Listing (ANSI C):

```
#include "f05.h"
#include "flash470.h"
/*******************************************************************
 *                                                                 *
 *  FILENAME: init_state.c                                         *
 *  Copyright (c) Texas Instruments 2000-2005, All Rights Reserved *
 *                                                                 *
 *  This function is used to initialize the Flash state machine    *
 *  registers so that the intended timings and pump voltages are   *
 *  used during program/erase/compaction.                          *
 *                                                                 *
 *******************************************************************/
void init_state_machine(FLASH_CORE core,
            UINT32 delay,
            FLASH_ARRAY_ST cntl
            )
{
  UINT32 k;
  cntl[MAC2]=(cntl[MAC2]&~0x7)|core;    /* clear bank select bits and set core select */
  cntl[MAC1]|=0x8000;                   /* disable level 1 sector protection */
  cntl[TCR]=0x2fc0;                     /* 0x2fc0 = TCR=0 */
  cntl[MAXPP]=0x0000+MAX_PROG_PLS;      /* start VNV steps at 0, Max programming pulses */
  cntl[MAXCP]=0xf000+MAX_CMPCT_PLS;     /* 16 pulses per VNV step, Max compaction pulses */
  if((cntl[MAXEP]&0xf000)==0xf000) {    /* Baseline 4 material */
    cntl[PTR3]=0x9964;                  /* Set VHV to 9.0v during erase */
    cntl[MAXEP]=0xf000+MAX_ERASE_PLS;   /* max VNV step, Max erase pulses */
  } else {                              /* Baseline 5 or 6 */
    cntl[PTR3]=0x9b64;                  /* Set VHV to 9.0v during erase */
    cntl[MAXEP]=0xa000+MAX_ERASE_PLS;   /* max VNV step, Max erase pulses */
  }
  cntl[PTR4]=0xa000;                    /* Set Vread to 5.0v */
  cntl[PESETUP]=(delay<<4|delay<<8);    /* Set program setup to 500nS, erase setup to 8uS */
  k=(delay|delay<<8);
  cntl[PVEVACCESS]=k;                   /* Program and Erase Verify Access = 500ns */
  k<<=1;
  cntl[PCHOLD]=k;                       /* Program hold and compaction hold =1us */
  cntl[PVEVHOLD]=k;                     /* Program verify hold and erase verify hold =1us */
  cntl[PVEVSETUP]=k;
  cntl[CVACCESS]=delay*16;              /* Compaction verify access time = 8us */
  k=delay*20;
  cntl[CSETUP]=(k|0x3000);              /* Compaction setup = 10us */
  cntl[EHOLD]=(k<<2);                   /* Erase hold = 40us */
  cntl[PWIDTH]=(delay*PROG_PLS_WIDTH);  /* Program width = 4us */
  cntl[CWIDTH]=(delay*CMPCT_PLS_WIDTH); /* Compaction pulse width */
  cntl[EWIDTH]=(delay*ERASE_PLS_WIDTH); /* Erase pulse width */
}
```

4.15 PSA_U32

```
UINT32 PSA_U32(UINT32 *start,
                 UINT32 length,
                 UINT32 psa_seed
                 );
```

Parameters:

Parameter	Type	Purpose
start	UINT32 *	Points to the first word to be read
length	UINT32	Number of 32 bit words to be read
psa_seed	UINT32	initial value of the PSA calculation

Return Value:

This function returns an unsigned 32-bit integer that is the PSA value computed:

Description:

This function is used to calculate a PSA value over any 32-bit memory range. It can be used on Flash, RAM or ROM. There is no restriction about crossing bank boundaries. Generally this function is called from another function to make sure the PSA is calculated in the proper mode.

Source Listing (ARM7 Assembly):

```
    MODULE PSA
    RSEG CODE:NOROOT(2)
    PUBLIC __PSA_U32
    CODE16
__PSA_U32:
     BX        pc
     NOP
    CODE32
    PUBLIC PSA_U32
;----------------------------------------------------------------------
;   4 | UINT32 PSA_U32(UINT32 *pstart, UINT32 plength, UINT32 pseed)
;----------------------------------------------------------------------
;***************************************************************************
;* FUNCTION NAME: _PSA                                                       *
;***************************************************************************
PSA_U32:
        MRS       r12, CPSR              ; save int flags
        ORR       A4,r12,#00000C0h       ; set int disable bits
        MSR       CPSR,A4               ; disable ints
;----------------------------------------------------------------------
;   5 | { int k;
;   6 | volatile UINT32 *daddr;
;   8 | daddr=pstart;
;   9 | *(UINT32 *)0xffffff50=0;
;----------------------------------------------------------------------
        MVN       A4, #191
        STR       A1, [A4, #16]         ; A1 contains start address which
                                        ;   must have lsb = 0;
;----------------------------------------------------------------------
;  10 | *(UINT32 *)0xffffff40=pseed;     /* initialize register to seed */
;----------------------------------------------------------------------
        STR       A3, [A4, #0]          ; |10|
;----------------------------------------------------------------------
;  11 | for (k=0;k<plength;k++)          /* Step through each flash locatio
;  12 | *daddr++;                        /* Read flash data */
;----------------------------------------------------------------------
L1:
        LDR       A3, [A1], #4          ; |12|
        SUBS      A2, A2, #1            ; |12|
        BNE       L1                    ; |12|
;----------------------------------------------------------------------
;  13 | *(UINT32 *)0xffffff50=1;                     /* disable PSA */
;----------------------------------------------------------------------
        MOV       A3, #1                ; |13|
        STR       A3, [A4, #16]         ; |13|
        MSR       CPSR,r12               ; restore ints
;----------------------------------------------------------------------
;  14 | return *(UINT32 *)0xffffff40;
;----------------------------------------------------------------------
        LDR       A1, [A4, #0]          ; |14|
        BX        LR
        ENDMOD
        END
```

5 Flash Delay Parameter Values

Some flash algorithms rely on a delay parameter to generate timing. These algorithms take a UINT32 parameter "delay" that is meant to compensate the clock frequency. Because F05 uses a state machine, this parameter is independent of wait states.

Table 1. Flash Delay Parameter Values

Frequency (MHz)	
10	5
10 < f ≤ 12	6
12 < f ≤ 14	7
14 < f ≤ 16	8
16 < f ≤ 18	9
18 < f ≤ 20	10
20 < f ≤ 22	11
22 < f ≤ 24	12

6 Recommended Flow Guidelines

6.1 6.1. Reasoning Behind Fow Guidelines

In a perfect world there are no power glitches, hardware failures, or user interruptions to prevent an attempt to program/erase the Flash from completing as intended. Since we live in an imperfect world, the possibility of power glitches, hardware failures, or user interruptions, though small, is real and needs to be taken into account when implementing a robust flow for programming and erasing devices. The flow needs to be able to correct any depleted or under-erased bits that may be left over after just such an interruption.

6.1.1 Balanced Sectoring Scheme

Care must also be taken when erasing the Flash in light of the balanced sectoring scheme. Sensing of data in one sector is balanced by the sector physically opposite it in the same Flash bank. This improves data access time at the expense of having to account for the interaction between balancing sectors when erasing.

The risk imposed by this scheme is that if depleted columns exist (i.e. columns where bits are in an “always on” state) in a balancing sector, then erasing the sector it balances can result in over-erase of the target sector that can render the device unusable.

Because of this risk, it is recommended that when implementing your erase flow using the Flash API, all sectors on the device are to be compacted using “Flash_Compact_B” prior to any calls to “Flash_Erase_B”, “Flash_Sector_Erase_B”. Compaction poses no risk to current data in a sector, but it will correct any depletion that may exist in the sector.

6.1.2 Problems Caused by Interruptions During Erase

If erase is interrupted while erase pulses are being applied, the sector may appear to be erased if it is later tested with “Flash_Blank_B”, but it in fact may contain bits that are not erased with sufficient long-term margin. The “Flash_Erase_B” function can be flagged to “fix” such sectors by passing it a “key” through the “status.Stat1_U32” structure value. If status.Stat1_U32=0x12345678, then Flash_Erase_B will disable the preconditioning step (i.e. programming of all bits in sector to 0’s) and only apply erase and compaction pulses until all bits in the sector pass both an ERASE VERIFY read and a COMPACTION VERIFY read. Otherwise, a full erase is performed. The decision of whether to pass this key to Flash_Erase_B should be determined by the result of calling the “Flash_Blank_B” function prior to calling Flash_Erase_B. If the Flash_Blank_B returns TRUE, then the key (i.e. status.Stat1_U32=0x12345678) should initialized prior to calling Flash_Erase_B, otherwise status.Stat1_U32=0x0000000 (or some value other than 0x12345678) should be initialized prior to calling Flash_Erase_B. This key to disable preconditioning is not available in the Flash_Start_Erase_B and the Flash_Erase_Sector_B functions, because these functions do not have status as one of their arguments.

6.2 Recommended Erase Flows

6.2.1 Using Flash_Erase_B

The following flow chart describes the flow for erasing an arbitrary number of sectors on a device using the Flash_Erase_B function. This flow is desirable from a throughput standpoint in that sectors which already read as blank are processed much faster, but it is more expensive in terms of code complexity and code size. The hallmark of the flow is the ability to disable preconditioning in Flash_Erase_B using the first 32 bit value in the **FLASH_STATUS_ST status structure** as a “key” when erasing sectors read as blank by Flash_Blank_B. Disabling preconditioning significantly speeds up erase of blank sectors.

It is not advisable to skip erase altogether on sectors that read as blank, because these sectors may require repair to marginally erased bits or depleted columns that is performed during execution of Flash_Erase_B.

Also note that Flash_Erase_B is the only function which enables erase that allows for the collection of erase and compaction pulse counts

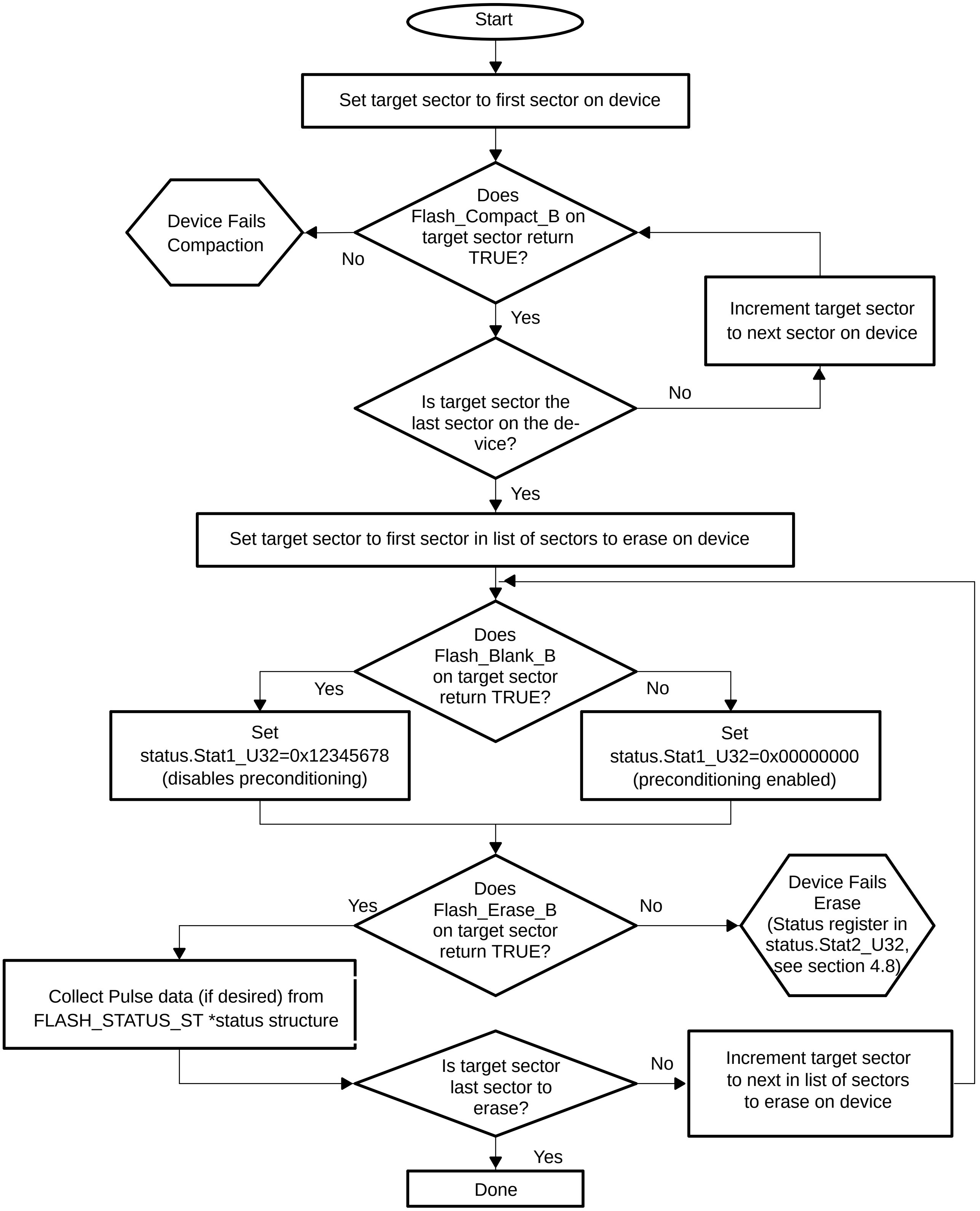
Start
Set target sector to first sector on device
Does Flash_Compact_B on target sector return TRUE?
No
Device Fails Compaction
Yes
Is target sector the last sector on the de- vice?
No
Increment target sector to next sector on device
Yes
Set target sector to first sector in list of sectors to erase on device
Does Flash_Blank_B on target sector return TRUE?
Yes
No
Set status.Stat1_U32=0x12345678 (disables preconditioning)
Set status.Stat1_U32=0x00000000 (preconditioning enabled)
Does Flash_Erase_B on target sector return TRUE?
Yes
No
Device Fails Erase (Status register in status.Stat2_U32, see section 4.8)
Collect Pulse data (if desired) from FLASH_STATUS_ST *status structure
Is target sector last sector to erase?
No
Increment target sector to next in list of sectors to erase on device
Yes
Done

Figure 1.

6.2.2 Using Flash_Erase_Sector_B

The following flow chart describes the flow for erasing an arbitrary number of sectors on a device using the Flash_Erase_Sector_B function. This flow is less desirable from a throughput standpoint in that all sectors undergo a full erase, regardless of the data in the sector, but it is less expensive in terms of code complexity and code size.

It is not advisable to skip erase altogether on sectors that read as blank, because these sectors may require repair to marginally erased bits or depleted columns that is performed during execution of Flash_Erase_Sector_B.

Also note that Flash_Erase_Sector_B does not datalog erase and compaction pulse counts.

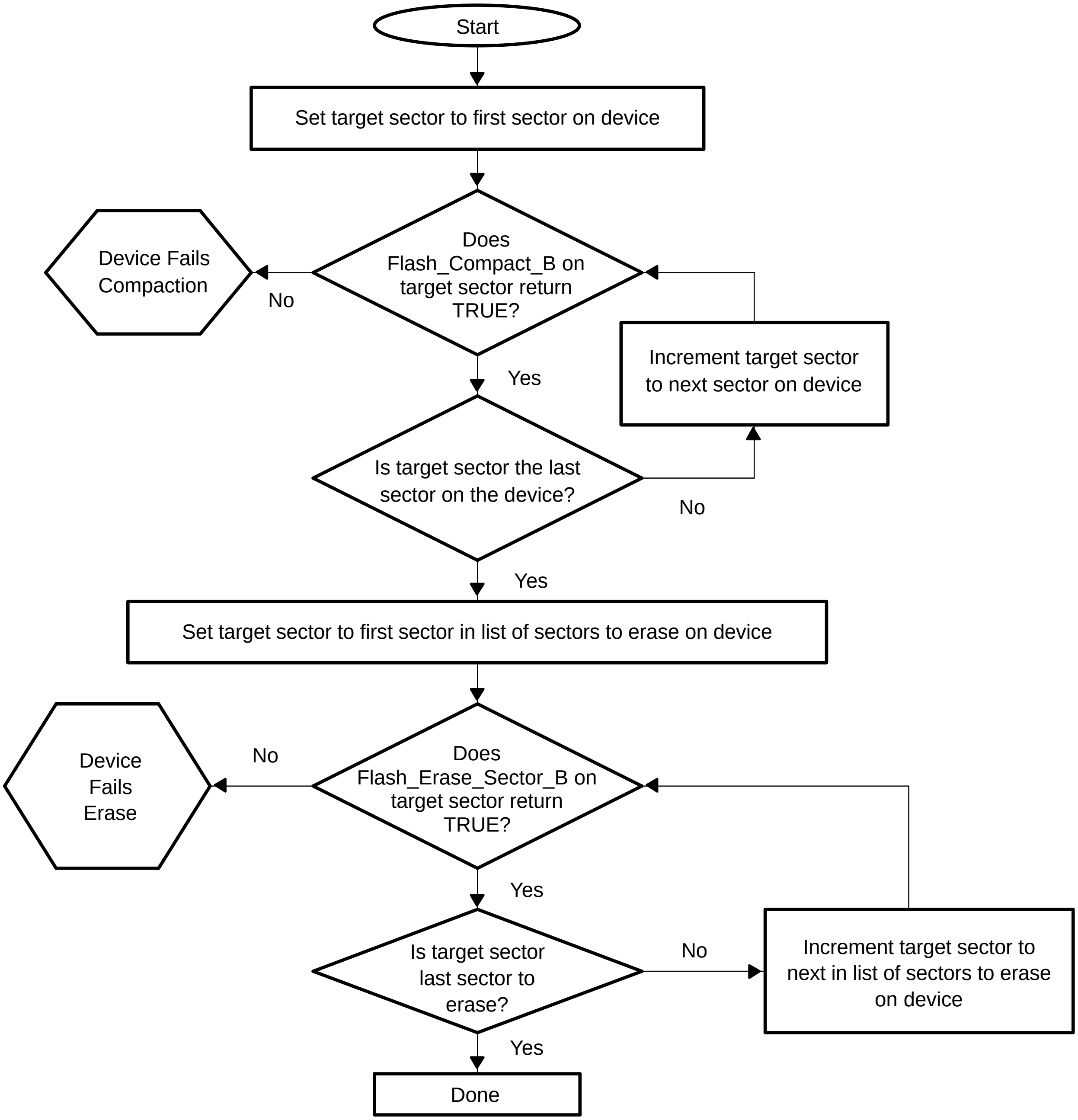
Start
Set target sector to first sector on device
Does Flash_Compact_B on target sector return TRUE?
No
Device Fails Compaction
Yes
Increment target sector to next sector on device
Is target sector the last sector on the device?
No
Yes
Set target sector to first sector in list of sectors to erase on device
Does Flash_Erase_Sector_B on target sector return TRUE?
No
Device Fails Erase
Yes
Is target sector last sector to erase?
No
Increment target sector to next in list of sectors to erase on device
Yes
Done

Figure 2.

6.3 Recommended Programming Fow

When programming the Flash, you should first erase all affected sectors using the previously described erase flow, and you will need to manage the data buffers being programmed to Flash such that they do not cross boundaries between Flash banks.

For example, if you have 1KB of data to write starting at the last 768 bytes of bank 0 on a device with more than 1 bank, you will need to divide the data into a 768 byte chunk to be written to bank 0 with one call to Flash_Prog_B, and the remaining 256 bytes are to be written to bank 1 with a second call to Flash_Prog_B. Within the same bank, you may program any amount of data within the limits of the available data buffer.

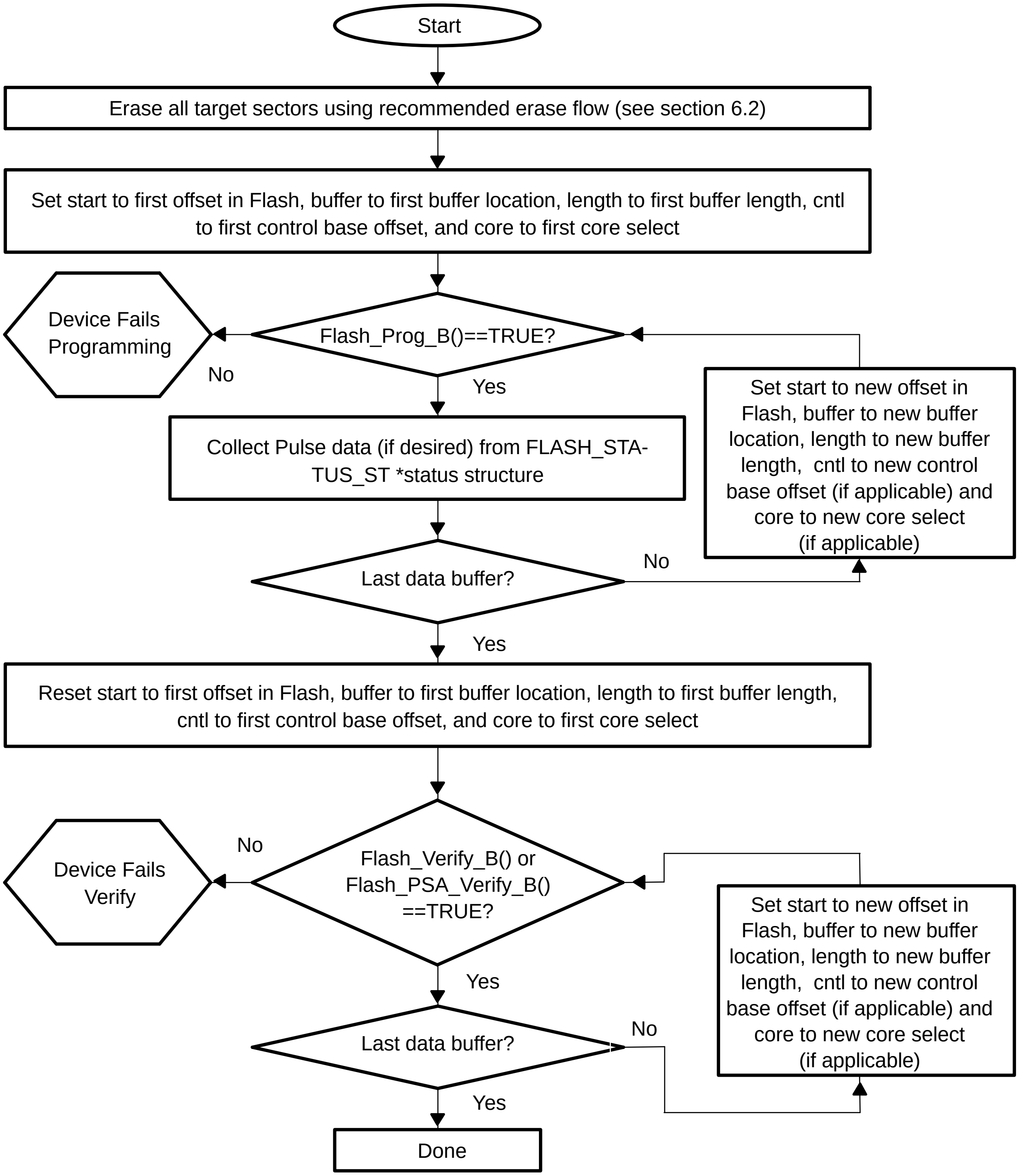
Start
Erase all target sectors using recommended erase flow (see section 6.2)
Set start to first offset in Flash, buffer to first buffer location, length to first buffer length, cntl to first control base offset, and core to first core select
Device Fails Programming
No
Flash_Prog_B()==TRUE?
Yes
Collect Pulse data (if desired) from FLASH_STATUS_ST *status structure
Set start to new offset in Flash, buffer to new buffer location, length to new buffer length, cntl to new control base offset (if applicable) and core to new core select (if applicable)
Last data buffer?
No
Yes
Reset start to first offset in Flash, buffer to first buffer location, length to first buffer length, cntl to first control base offset, and core to first core select
Device Fails Verify
No
Flash_Verify_B() or Flash_PSA_Verify_B() ==TRUE?
Yes
Set start to new offset in Flash, buffer to new buffer location, length to new buffer length, cntl to new control base offset (if applicable) and core to new core select (if applicable)
Last data buffer?
No
Yes
Done

Figure 3.

7 Related Documentation

7.1 Hardware Module Reference

Refer to Texas Instruments *F05 Flash Module*.

7.2 Software Module Reference

The Flash API Software modules are based on the *Version 0.31* of the Flash API code

Appendix A Header Files

The following additional header files are included during compilation of the object libraries contained in the Flash API library. They are included as an additional reference, and the source for these files is included with the object library.

A.0.3 fapiver.h

```
PUBLIC _fapiver
    _fapiver EQU 0x0031
```

A.0.4 Flash470.h

```
/* define variable types */
typedef int BOOL;
typedef unsigned char UBYTE;
typedef unsigned short int UINT16;
typedef unsigned long int UINT32;

/* define booleans */
#define FALSE 0
#define TRUE 1

/* define structures */

/*********************************************
 * The FLASH_STATUS_ST structure is used for *
 * storing and retrieving information beyond *
 * just the return value from a function.    *
 * for example, pulse data from program/erase*
 * and failing address and data              *
 *********************************************/
typedef struct {
  UINT32 stat1;
  UINT32 stat2;
  UINT32 stat3;
  UINT32 stat4;
} FLASH_STATUS_ST;

/*********************************************
 * The FLASH_ENG_INFO_ST structure contains  *
 * engineering information stored in the     *
 * engineering portion of the Flash (for F10 *
 * this is the ENGR0 row of the engineering  *
 * array, and for F05 the TI OTP sector)     *
 *********************************************/
typedef struct {
  UINT32 DevID;
  UINT32 LotNo;
  UINT16 FlowCheck;
  UINT16 WaferNo;
  UINT16 Xcoord;
  UINT16 Ycoord;
} FLASH_ENGR_INFO_ST;

/***************************************************
 * The FLASH_ARRAY_ST structure type defined       *
 * a UINT32 array pointer which points to          *
 * the first control base address of a given       *
 * flash module.                                   *
 * The index of the array defines what 32 bit      *
 * offset from the first address to store the      *
 * intended data.  For example:                    *
 *   FLASH_ARRAY_ST cntl=0xffe88000;               *
 *   cntl[4]=0xa5; (this stores 0xa5 to 0xffe88010)*
 ***************************************************/
typedef volatile UINT32 * FLASH_ARRAY_ST;
```

```
/*********************************************************
 * F10 devices can have up to 4 cores per module         *
 * F05, F035 devices can have up to 8 cores per module   *
 *********************************************************/
typedef enum {
#ifndef F10
  FLASH_CORE0
  ,FLASH_CORE1
  ,FLASH_CORE2
  ,FLASH_CORE3
  ,FLASH_CORE4
  ,FLASH_CORE5
  ,FLASH_CORE6
  ,FLASH_CORE7
#else
  FLASH_CORE0
  ,FLASH_CORE1
  ,FLASH_CORE2
  ,FLASH_CORE3
#endif
} FLASH_CORE;
/****************************************************
 * F10 devices can have up to  4 sectors per core   *
 * F05 devices can have up to  32 sectors per core  *
 * F035 devices can have up to 16 sectors per core  *
 ****************************************************/
typedef enum {
#ifndef F10
  FLASH_SECT0
  ,FLASH_SECT1
  ,FLASH_SECT2
  ,FLASH_SECT3
  ,FLASH_SECT4
  ,FLASH_SECT5
  ,FLASH_SECT6
  ,FLASH_SECT7
  ,FLASH_SECT8
  ,FLASH_SECT9
  ,FLASH_SECT10
  ,FLASH_SECT11
  ,FLASH_SECT12
  ,FLASH_SECT13
  ,FLASH_SECT14
  ,FLASH_SECT15
#ifdef F05
  ,FLASH_SECT16
  ,FLASH_SECT17
  ,FLASH_SECT18
  ,FLASH_SECT19
  ,FLASH_SECT20
  ,FLASH_SECT21
  ,FLASH_SECT22
  ,FLASH_SECT23
  ,FLASH_SECT24
  ,FLASH_SECT25
  ,FLASH_SECT26
  ,FLASH_SECT27
  ,FLASH_SECT28
  ,FLASH_SECT29
  ,FLASH_SECT30
  ,FLASH_SECT31
#endif
#else
  FLASH_SECT0
  ,FLASH_SECT1
  ,FLASH_SECT2
  ,FLASH_SECT3
#endif
} FLASH_SECT;
/* Declare function prototypes */
UINT16 Flash_API_Version_U16(void);
BOOL Flash_Erase_Sector_B(UINT32 *start,
                          UINT32 length,
                          FLASH_CORE core,
                          FLASH_SECT sector,
                          UINT32 delay,
                          FLASH_ARRAY_ST cntl
                          );
BOOL Flash_Blank_B(UINT32 *start,
                   UINT32 length,
                   FLASH_CORE core,
                   FLASH_ARRAY_ST cntl,
                   FLASH_STATUS_ST *status
                   );
BOOL Flash_Verify_Zeros_B(UINT32 *start,
                          UINT32 length,
                          FLASH_CORE core,
                          FLASH_ARRAY_ST cntl,
                          FLASH_STATUS_ST *status
                          );
BOOL Flash_Compact_B(UINT32 *start,
                     FLASH_CORE core,
                     FLASH_SECT sector,
                     UINT32 delay,
                     FLASH_ARRAY_ST cntl,
                     FLASH_STATUS_ST *status
                     );
BOOL Flash_Erase_B(UINT32 *start,
                   UINT32 length,
                   FLASH_CORE core,
                   FLASH_SECT sector,
                   UINT32 delay,
                   FLASH_ARRAY_ST cntl,
                   FLASH_STATUS_ST *status
                   );
BOOL Flash_Prog_B(UINT32 *start,
                  UINT32 *buffer,
                  UINT32 length,
                  FLASH_CORE core,
                  UINT32 delay,
                  FLASH_ARRAY_ST cntl,
                  FLASH_STATUS_ST *status
                  );
BOOL Flash_Zeros_B(UINT32 *start,
                   UINT32 length,
                   FLASH_CORE core,
                   UINT32 delay,
                   FLASH_ARRAY_ST cntl,
                   FLASH_STATUS_ST *status
                   s);
BOOL Flash_Verify_B(UINT32 *start,
                    UINT32 *buffer,
                    UINT32 length,
                    FLASH_CORE core,
                    FLASH_ARRAY_ST cntl,
                    FLASH_STATUS_ST *status
                    );
BOOL Flash_PSA_Verify_B(UINT32 *start,
                        UINT32 length,
                        UINT32 psa,
                        FLASH_CORE core,
                        FLASH_ARRAY_ST cntl,
                        FLASH_STATUS_ST *status
                        );
UINT32 Flash_PSA_Calc_U32(UINT32 *start,
                          UINT32 length,
                          UINT32 psa_seed,
                          UINT32 mode,
                          FLASH_CORE core,
                          FLASH_ARRAY_ST cntl
                          );
UINT32 PSA_U32(UINT32 *start,
               UINT32 length,
               UINT32 pseed
               );
BOOL Flash_Match_Key_B(volatile UINT32 *key_start,
                       const UINT32 key[],
                       FLASH_ARRAY_ST cntl
                       );
void Flash_Match_Key_V(volatile UINT32 *key_start,
                       const UINT32 key[],
                       FLASH_ARRAY_ST cntl
                       );
void Flash_EngRow_V(UINT32 *start,
                    FLASH_ARRAY_ST cntl,
                    FLASH_STATUS_ST *status
                    );
void Flash_EngInfo_V(UINT32 *start,
                     FLASH_ARRAY_ST cntl,
                     FLASH_ENGR_INFO_ST *info
                     );
BOOL Flash_Vt_Verify_B(UINT32 *start,
                       UINT32 *buffer,
                       UINT32 length,
                       FLASH_CORE core,
                       UINT32 delay,
                       FLASH_ARRAY_ST cntl,
                       FLASH_STATUS_ST *status
                       );
BOOL Flash_PSA_Vt_Verify_B(UINT32 *start,
                           UINT32 length,
                           UINT32 psa,
                           FLASH_CORE core,
                           UINT32 delay,
                           FLASH_ARRAY_ST cntl,
                           FLASH_STATUS_ST *status
                           );
BOOL Flash_Vt_Blank_B(UINT32 *start,
                      UINT32 length,
                      FLASH_CORE core,
                      UINT32 delay,
                      FLASH_ARRAY_ST cntl,
                      FLASH_STATUS_ST *status
                      );
BOOL Flash_Vt_Verify_Zeros_B(UINT32 *start,
                             UINT32 length,
                             FLASH_CORE core,
                             UINT32 delay,
                             FLASH_ARRAY_ST cntl,
                             FLASH_STATUS_ST *status
                             );
```

```
void wait_delay(UINT32 delay);

void Flash_Read_V(UINT32 *start,
                  UINT32 *buff,
                  UINT32 length,
                  UINT32 mode,
                  FLASH_CORE core,
                  FLASH_ARRAY_ST cntl
                  );

void Flash_Vt_Read_V(UINT32 *start,
                     UINT32 *buff,
                     UINT32 length,
                     UINT32 delay,
                     FLASH_CORE core,
                     FLASH_ARRAY_ST cntl
                     );

#ifdef F10
void Flash_Prog_Key_V(volatile UINT32 *key_start,
                      const UINT32 key[],
                      FLASH_CORE core,
                      UINT32 delay,
                      FLASH_ARRAY_ST cntl
                      );

void Flash_Zero_Key_V(volatile UINT32 *key_start,
                      const UINT32 key[],
                      FLASH_CORE core,
                      UINT32 delay,
                      FLASH_ARRAY_ST cntl
                      );
#endif

#ifdef F05
BOOL OTP_Blank_B(UINT32 *start,
                 UINT32 length,
                 FLASH_CORE core,
                 FLASH_ARRAY_ST cntl,
                 FLASH_STATUS_ST *status
                 );

BOOL OTP_Prog_B(UINT32 *start,
                UINT32 *buff,
                UINT32 length,
                FLASH_CORE core,
                UINT32 delay,
                FLASH_ARRAY_ST cntl,
                FLASH_STATUS_ST *status
                );

void OTP_Read_V(UINT32 *start,
                UINT32 *buff,
                UINT32 length,
                FLASH_CORE core,
                FLASH_ARRAY_ST cntl
                );

BOOL OTP_Verify_B(UINT32 *start,
                  UINT32 *buff,
                  UINT32 length,
                  FLASH_CORE core,
                  FLASH_ARRAY_ST cntl,
                  FLASH_STATUS_ST *status
                  );

BOOL OTP_PSA_Verify_B(UINT32 *start,
                      UINT32 length,
                      UINT32 psa,
                      FLASH_CORE core,
                      FLASH_ARRAY_ST cntl,
                      FLASH_STATUS_ST *status
                      );

void Flash_Set_Vread_V(FLASH_ARRAY_ST cntl);

BOOL Flash_Start_Command_B(UINT32 *start,
                           FLASH_CORE core,
                           UINT32 delay,
                           FLASH_ARRAY_ST cntl,
                           UINT16 cmnd,
                           UINT16 data
                           );

BOOL Flash_Start_Compact_B(UINT32 *start,
                           FLASH_CORE core,
                           UINT32 delay,
                           FLASH_ARRAY_ST cntl
                           );

BOOL Flash_Start_Erase_B(UINT32 *start,
                         FLASH_CORE core,
                         UINT32 delay,
                         FLASH_ARRAY_ST cntl
                         );

UINT16 Flash_Compact_Status_U16(FLASH_ARRAY_ST cntl);

UINT16 Flash_Erase_Status_U16(FLASH_ARRAY_ST cntl);

UINT16 Flash_Status_U16(FLASH_ARRAY_ST cntl);

UINT16 Flash_Aux_Engr_U16(volatile UINT16 *start,
                          FLASH_ARRAY_ST cntl
                          );

void init_state_machine(FLASH_CORE core,
                        UINT32 delay,
                        FLASH_ARRAY_ST cntl
                        );

void Flash_Sector_Select_V(FLASH_CORE core,
                           FLASH_ARRAY_ST cntl
                           );

void Feed_Watchdog_V();

#endif
```

A.0.5 F05.h

```
#define F05

/* define macros */

/* The DUALCPU flag can be specified in your cl470 CFLAGS as '-dDUALCPU'
 * or explicitly in a previous included header file using '#define DUALCPU'
 */
#ifdef DUALCPU
#define ENA_CLEAN  *(volatile UINT32 *)0xffffff50=0;
#define DIS_CLEAN *(volatile UINT32 *)0xffffff50=1;
#define CLEAN(x) *(volatile UINT32 *)0xffffff40=x;x=*(volatile UINT32 *)0xffffff40;
#else
#define ENA_CLEAN
#define DIS_CLEAN
#define CLEAN(x)
#endif /* DUALCPU */

#define READ_FLASH_MEM_U16(a,x) x=*(volatile UINT16 *)a;CLEAN(x);
#define READ_FLASH_MEM_U8(a,x) x=*(volatile UCHAR *)a;CLEAN(x);

/* The PLATFORM flag can be specified in your cl470 CFLAGS as '-dPLATFORM'
 * or explicitly in a previous included header file using '#define PLATFORM'
 */
#ifdef PLATFORM

/* macros */
#define SET_CFG_BIT
#define CLR_CFG_BIT
```

```
#define READ_FLASH_MEM_U32(a,x) { UINT32 y;x=*(volatile UINT16 *)
a;CLEAN(x);y=*(volatile UINT16 *)((UINT32)a+sizeof(UINT16));CLEAN(y);x<<=16;x|=y;}

/* register offsets */
#define FMREGOPT       0x0100/4 /* FMREGOPT[31:5] Reserved
                                 * FMREGOPT[4]=SPOTP
                                 * FMREGOPT[3]=RDMRGN1
                                 * FMREGOPT[2]=RDMRGN0
                                 * FMREGOPT[1]=READOTP
                                 * FMREGOPT[0]=ENPIPE
                                 */

#define FMBBUSY        0x0108/4 /* FMBBUSY[31:16] Reserved
                                 * FMBBUSY[15]=PROTL2DIS
                                 * FMBBUSY[14:8] Reserved
                                 * FMBBUSY[7:0]=BUSY[7:0]
                                 */

#define FMPKEY         0x010c/4 /* FMPKEY[31:00] Expected Protection Key
                                 */

#define FMEDACCTRL1    0x0110/4 /* FMEDACCTRL1[31:12]=Reserved
                                 * FMEDACCTRL1[11]=ECC-MAP
                                 * FMEDACCTRL1[10]=EOFEN
                                 * FMEDACCTRL1[9]=EZFEN
                                 * FMEDACCTRL1[8]=EPEN
                                 * FMEDACCTRL1[7:6]=Reserved
                                 * FMEDACCTRL1[5]=EOCV
                                 * FMEDACCTRL1[4]=EZCV
                                 * FMEDACCTRL1[3:0]=EDACEN[3:0]
                                 */

#define FMEDACCTRL2    0x0114/4 /* FMEDACCTRL2[31:16]=Reserved
                                 * FMEDACCTRL2[15:0]=SEC_THRESHOLD[15:0]
                                 */

#define FMEDACCNT      0x0118/4 /* FMEDACCNT[31:16]=Reserved
                                 * FMEDACCNT[15:0]=SEC_OCCR[15:0]
                                 */

#define FMCERRADDR     0x011c/4 /* FMCERRADDR[31:16]=Reserved
                                 * FMCERRADDR[15:0]=CERRADDR[15:0]
                                 */

#define FMCERRPOSITION 0x0120/4 /* FMCERRPOSITION[31:9]=Reserved
                                 * FMCERRPOSITION[8]=ERRTYPE
                                 * FMCERRPOSITION[7]='0'
                                 * FMCERRPOSITION[6]='0'
                                 * FMCERRPOSITION[5:0]=SERRPOSITION[5:0]
                                 */

#define FMEDACSTATUS   0x0124/4 /* FMEDACSTATUS[31:3]=Reserved
                                 * FMEDACSTATUS[2]=EOFS
                                 * FMEDACSTATUS[1]=EZFS
                                 * FMEDACSTATUS[0]=EPS
                                 */

#define FMUERRADDR     0x0128/4 /* FMUERRADDR[31:24]=Reserved
                                 * FMUERRADDR[23:16]=UERRADDR[23:16]
                                 * FMUERRADDR[15:0]=UERRADDR[15:0]
                                 */

#define FMEMUDMSW      0x012c/4 /* FMEMUDMSW[31:16]=EMUDW[63:48]
                                 * FMEMUDMSW[15:0]=EMUDW[47:32]
                                 */

#define FMEMUDLSW      0x0130/4 /* FMEMUDLSW[31:16]=EMUDW[31:16]
                                 * FMEMUDLSW[15:0]=EMUDW[15:0]
                                 */

#define FMEMUECC       0x0134/4 /* FMEMUECC[31:8]=Reserved
                                 * FMEMUECC[7:0]=EMUECC[7:0]
                                 */

#define FMSECDIS       0x0138/4 /* FMSECDIS[31:29]=bankID1_inverse[2:0]
```

```
                                 * FMSECDIS[28:24]=SectorID1_inverse[4:0]
                                 * FMSECDIS[23:21]=bankID1[2:0]
                                 * FMSECDIS[20:16]=SectorID1[4:0]
                                 * FMSECDIS[15:13]=bankID0_inverse[2:0]
                                 * FMSECDIS[12:8]=SectorID0_inverse[4:0]
                                 * FMSECDIS[7:5]=bankID0[2:0]
                                 * FMSECDIS[4:0]=SectorID0[4:0]
                                 */
#define FMDIAGCTRL     0x0134/4 /* FMDIAGCTRL[31:10]=Reserved
                                 * FMDIAGCTRL[9]=DUERR
                                 * FMDIAGCTRL[8]=DCERR
                                 * FMDIAGCTRL[7:2]=Reserved
                                 * FMDIAGCTRL[1:0]=DIAGMODE[1:0]
                                 */
/* 16 bit Flash control registers */
#define FMBAC1         0x0000/4 /* FMBAC1[15:8]=BAGP[7:0]
                                 * FMBAC1[7:2]=BSTDBY[5:0]
                                 * FMBAC1[1:0]=BNKPWR[1:0]
                                 */
#define FMBAC2         0x0004/4 /* FMBAC2[15]=OTPPROTDIS
                                 * FMBAC2[14:8]=BLEEP[6:0]
                                 * FMBAC2[7:4]=PAGEWAIT[3:0]
                                 * FMBAC2[3:0]=READWAIT[3:0]
                                 */
#define FMBSEA         0x0008/4 /* FMBSEA[15:0]=Enable sectors 15:0
                                 */
#define FMBSEB         0x000c/4 /* FMBSEB[15:0]=Enable sectors 31:16
                                 */
#define FMBRDY         0x0010/4 /* FMBRDY[15:6]=Reserved
                                 * FMBRDY[5]=BANKRDY
                                 * FMBRDY[4:0]=Reserved
                                 */
#define FMPRDY         0x0014/4 /* FMPRDY[15:10]=Reserved
                                 * FMPRDY[9]=PUMPRDY
                                 * FMPRDY[8:0]=Reserved
                                 */
#define FMMAC1         0x0018/4 /* FMMAC1[15]=PROTL1DIS
                                 * FMMAC1[14:0]=PSLEEP[14:0]
                                 */
#define FMMAC2         0x001c/4 /* FMMAC2[15:5]=PSTDBY[10:0]
                                 * FMMAC2[4:3]=PMPPWR[1:0]
                                 * FMMAC2[2:0]=BANK[2:0]
                                 */
#define FMPAGP         0x0020/4 /* FMPAGP[15:0]=PAGP[15:0]
                                 */
#define FMMSTAT        0x0024/4 /* FMMSTAT[15:9]=Reserved
                                 * FMMSTAT[8]=BUSY
                                 * FMMSTAT[7]=ERS
                                 * FMMSTAT[6]=PGM
                                 * FMMSTAT[5]=INVDAT
                                 * FMMSTAT[4]=CSTAT/MAXPLS
                                 * FMMSTAT[3]=3VSTAT
                                 * FMMSTAT[2]=ESUSP
                                 * FMMSTAT[1]=PSUSP
                                 * FMMSTAT[0]=SLOCK
                                 */
/* other registers */
#define FMTCREG        0x0028/4 /* FMTCREG[15:0] */
#define FMPTR0         0x0300/4 /* FMPTR0[15:0] */
#define FMPTR1         0x0304/4 /* FMPTR1[15:0] */
#define FMPTR2         0x0308/4 /* FMPTR2[15:0] */
```

```
#define FMPTR3         0x030c/4 /* FMPTR3[15:0] */
#define FMPTR4         0x0310/4 /* FMPTR4[15:0] */
#define FMPESETUP      0x0218/4 /* FMPESETUP[15:0] */
#define FMCSETUP       0x0220/4 /* FMCSETUP[15:0] */
#define FMPVEVSETUP    0x0224/4 /* FMPVEVSETUP[15:0] */
#define FMPCHOLD       0x0234/4 /* FMPCHOLD[15:0] */
#define FMEHOLD        0x0238/4 /* FMEHOLD[15:0] */
#define FMPVEVHOLD     0x0240/4 /* FMPVEVHOLD[15:0] */
#define FMCVHOLD       0x0248/4 /* FMCVHOLD[15:0] */
#define FMPWIDTH       0x0250/4 /* FMPWIDTH[15:0] */
#define FMEWIDTH       0x0254/4 /* FMEWIDTH[15:0] */
#define FMCWIDTH       0x0258/4 /* FMCWIDTH[15:0] */
#define FMPVEVACCESS   0x025c/4 /* FMPVEVACCESS[15:0] */
#define FMCVACCESS     0x0260/4 /* FMCVACCESS[15:0] */
#define FMWSMREG       0x026c/4 /* FMWSMREG[15:0] */
#define FMMAXPP        0x027c/4 /* FMMAXPP[15:0] */
#define FMMAXEP        0x0280/4 /* FMMAXEP[15:0] */
#define FMMAXCP        0x0284/4 /* FMMAXCP[15:0] */
#define FMPLSCNT       0x0288/4 /* FMPLSCNT[15:0] */
#define FMADDRCTRLOW   0x0298/4 /* FMADDRCTRLOW[15:0] */

/* define aliases for PLATFORM */
#define REGOPT     FMREGOPT
#define BAC1       FMBAC1
#define BAC2       FMBAC2
#define BBUSY      FMBBUSY
#define PROTKEY    FMPKEY
#define BSEA       FMBSEA
#define BSEB       FMBSEB
#define BRDY       FMBRDY
#define PRDY       FMPRDY
#define MAC1       FMMAC1
#define MAC2       FMMAC2
#define PAGP       FMPAGP
#define MSTAT      FMMSTAT
#define TCR        FMTCREG
#define PTR0       FMPTR0
#define PTR1       FMPTR1
#define PTR2       FMPTR2
#define PTR3       FMPTR3
#define PTR4       FMPTR4
#define DPTR       FMBRDY
#define BPTR       FMPRDY
#define PESETUP    FMPESETUP
#define CSETUP     FMCSETUP
#define PVEVSETUP  FMPVEVSETUP
#define PCHOLD     FMPCHOLD
#define EHOLD      FMEHOLD
#define PVEVHOLD   FMPVEVHOLD
#define CVHOLD     FMCVHOLD
#define PWIDTH     FMPWIDTH
#define EWIDTH     FMEWIDTH
#define CWIDTH     FMCWIDTH
#define PVEVACCESS FMPVEVACCESS
#define CVACCESS   FMCVACCESS
#define WSMREG     FMWSMREG
#define MAXPP      FMMAXPP
#define MAXEP      FMMAXEP
#define MAXCP      FMMAXCP
#define ADDRCTRLOW FMADDRCTRLOW
#define MAX_PROG_PULSE FMMAXPP
#define PULSE_COUNTER FMPLSCNT

#else /* not PLATFORM */

/* macros */
#define SET_CFG_BIT *(volatile UINT32 *)0xffffffdc|=0x10;ENA_CLEAN;
#define CLR_CFG_BIT DIS_CLEAN;*(volatile UINT32 *)0xffffffdc&=~0x10;
```

```
#define READ_FLASH_MEM_U32(a,x) x=*(volatile UINT32 *)a;CLEAN(x);

/* register offsets */
#define BAC1               0x0000/4
#define BAC2               0x0004/4
#define BSEA               0x0008/4
#define BSEB               0x000c/4
#define DPTR               0x0010/4
#define REGOPT             0x1c00/4
#define BBUSY              0x1c08/4
#define PROTKEY            0x1c0c/4
#define PESETUP            0x2018/4
#define CSETUP             0x2020/4
#define PVEVSETUP          0x2024/4
#define PCHOLD             0x2034/4
#define EHOLD              0x2038/4
#define PVEVHOLD           0x2040/4
#define CVHOLD             0x2048/4
#define PWIDTH             0x2050/4
#define EWIDTH             0x2054/4
#define CWIDTH             0x2058/4
#define PVEVACCESS         0x205c/4
#define CVACCESS           0x2060/4
#define WSMREG             0x206c/4
#define MAXPP              0x207c/4
#define MAXEP              0x2080/4
#define MAXCP              0x2084/4
#define PULSE_COUNTER      0x2088/4
#define ADDRCTRLOW         0x2098/4
#define PTR1               0x2804/4
#define PTR2               0x2808/4
#define PTR3               0x280c/4
#define PTR4               0x2810/4
#define BPTR               0x2814/4
#define CTR0               0x2818/4
#define CTR1               0x281c/4
#define CTR2               0x2820/4
#define CTR3               0x2824/4
#define DATALATCH          0x2828/4
#define MAC1               0x3c00/4
#define MAC2               0x3c04/4
#define PAGP               0x3c08/4
#define MSTAT              0x3c0c/4
#define TCR                0x3c10/4
#define MAX_PROG_PULSE     MAXPP
#endif /* not PLATFORM */

/* WSMREG values */
#define PROG_PULSE_ACTIVE  0x0028
#define ERASE_PULSE_ACTIVE 0x0044
#define CMPCT_PULSE_ACTIVE 0x0064

/* REGOPT modes */
#define READMODE           0x00
#define PIPEMODE           0x01
#define RDM0MODE           0x04
#define RDM1MODE           0x08

/* constants and timings */

/* The MAX_DELAY value can be specified in your cl470 CFLAGS as '-dMAX_DELAY=<x>'
 * or explicitly in a previous included header file using '#define MAX_DELAY <x>'
 */
#ifndef MAX_DELAY
#define MAX_DELAY 12
#endif

#if (MAX_DELAY>16)
#define MAX_DELAY 16
#endif
```

```
#define PROG_PLS_WIDTH     8 /* 500ns * PROG_PLS_WIDTH  =   4us */
#define CMPCT_PLS_WIDTH 1000 /* 500ns * CMPCT_PLS_WIDTH = 500us */

#if (MAX_DELAY>12)
#define ERASE_PLS_WIDTH 4000 /* 500ns * ERASE_PLS_WIDTH = 2.0ms */
#else
#define ERASE_PLS_WIDTH 5400 /* 500ns * ERASE_PLS_WIDTH = 2.7ms */
#endif

#define MAX_PROG_PLS    50
#define MAX_CMPCT_PLS 2000
#define MAX_ERASE_PLS 4095

asm(" .copy \"fapiver.h\"");
```