TMS470/570 Platform F035 Flash API Reference Guide v1.09

User's Guide

Literature Number: SPNU493E
April 2012–Revised February 2014

Contents

List of Figures

List of Tables

User's Guide
SPNU493E – April 2012 – Revised February 2014

TMS470/570 Platform F035 Flash API Reference Guide v1.09

This reference guide documents the Application Programming Interface for Flash operations on the TMS470/570 Platform F035 devices.

1 Types, Structures, Enumerators, and Macros

1.1 Types

The following types are defined in the Flash470.h header file.

1.1.1 UINT64

```
typedef unsigned long long int UINT64;
```

and is used for the following purpose:

UINT64	Unsigned 64 bit integer

1.1.2 UINT32

```
typedef unsigned long int UINT32;
```

and is used for the following purpose:

UINT32	Unsigned 32-bit integer

1.1.3 UINT16

```
typedef unsigned short int UINT16;
```

and is used for the following purpose:

UINT16	Unsigned 16-bit integer

1.1.4 UINT8

```
typedef unsigned char UINT8;
```

and is used for the following purpose:

UINT8	Unsigned 8 bit integer

1.1.5 UBYTE

```
typedef unsigned char UBYTE;
```

and is used for the following purpose:

UBYTE	Unsigned bye (8 bits wide)

1.1.6 INT64

```
typedef long long int INT64;
```

and is used for the following purpose:

INT64	Signed 64 bit integer

1.1.7 INT32

```
typedef long int INT32;
```

and is used for the following purpose:

INT32	Signed 32-bit integer

1.1.8 INT16

```
typedef short int INT16;
```

and is used for the following purpose

INT16	Signed 16-bit integer

1.1.9 INT8

```
typedef char INT8;
```

and is used for the following purpose:

INT8	Signed 8 bit integer

1.1.10 BOOL

```
typedef int BOOL;
```

and is used for the following purpose:

BOOL	Boolean variable (1=TRUE, 0=FALSE)

1.1.11 FLASH_ARRAY_ST

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

Therefore, 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 F035.h header file.

1.2 Structures

1.2.1 FLASH_STATUS_ST

The FLASH_STATUS_ST is used as a repository of information (pulse counts, PSA values, failing addresses and data, and so forth) 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, and so forth

1.2.2 FLASH_FSM_INFO_ST

The FLASH_FSM_INFO_ST structure is used to contain timing and CT values for the Flash State Machine (FSM). Timings in this structure are to properly scaled according to the delay parameter (see Section 4), and all values are written to the FSM using the setup_state_machine function. The function of each element in the structure can be inferred from the comments below.

```
typedef struct {
  UBYTE  psetup;     /* FSMPESETUP bits 15: 8 */
  UBYTE  esetup;     /* FSMPESETUP bits  7: 0 */
  UINT16 csetup;     /* FSMCSETUP: V5STAT (bits 15:12)
                            *            CmpctSetup (bits 11:0)
                            */
  UBYTE  pvsetup;    /* FSMPVEVSETUP bits 15:8 */
  UBYTE  evsetup;    /* FSMPVEVSETUP bits 7:0 */
  UINT16 cvsetup;    /* FSMCVSETUP: Address/EXECUTEZ access time
              *               (bits 15:12)
                            *            CmpctVerSetup (bits 11:0)
                            */
  UBYTE  max_fosc_delay; /* Max allowed FOSC delay parameter */
  UBYTE  pvevaccess;  /* FSMPVEVACCESS bits 7:0 */
  UBYTE  phold;      /* FSMPCHOLD bits 15:8 */
  UBYTE  chold;      /* FSMPCHOLD bits 7:0 */
  UINT16 ehold;      /* FSMEHOLD bits 15:0 */
  UBYTE  pvhold;     /* FSMPVEVHOLD bits 15:8 */
  UBYTE  evhold;     /* FSMPVEVHOLD bits 7:0 */
  UINT16 pwidth;     /* FSMPWIDTH bits 15:0 */
  UINT16 cwidth;     /* FSMCWIDTH bits 15:0 */
  UINT32 ewidth;     /* FSMEWIDTH bits 31:0 */
  UINT16 vwlcmpctct; /* FSMVWLCMPCTCT bits 11:0 */
  UINT16 maxpp;      /* FSMMAXPP: Start VNV CT (bits 15:12)
                            *           Max Prog Pulses (bits 11:0)
                            */
  UINT16 maxep;      /* FSMMAXEP: Stop VNV CT (bits 15:12)
                            *           Max Erase Pulses (bits 11:0)
                            */
  UINT16 maxcp;      /* FSMMAXCP: Step VNV CT (bits 15:12)
                            *           Max Cmpct Pulses (bits 11:0)
                            */
  UINT32 vhvct1;     /* FSMVVHCT1 bits 31:0 */
  UINT16 vhvct2;     /* FSMVVHCT2 bits 11:0 */
  UINT16 vreadct;    /* FSMVREADCT bits 3:0 */
  UINT16 vppct;      /* FSMVPPCT bits 15:0 */
  UINT16 vwlct;      /* FSMVWLCT bits 15:4 */
  UINT32 chksum;     /* 32 bit checksum for structure */
} FLASH_FSM_INFO_ST;
```

1.3 Enumerators

1.3.1 FLASH_CORE

The FLASH_CORE type is an enumeration of the available banks that can be used in the Flash module. In F035, up to eight banks are supported, but the number of banks in a given Flash module are device dependent.

```
typedef enum {
   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 */
} FLASH_CORE;
```

1.3.2 FLASH_SECT

This type is an enumeration of the sector that can be chosen in a specified bank of the Flash module. In TMS570Px F035, sixteen sectors (numbered 0-15) are supported.

```
typedef enum {
   FLASH_SECT0    /* Sector 0 selected */
   FLASH_SECT1    /* Sector 1 selected */
      .
      .
      .
   FLASH_SECT15    /* Sector 15 selected */
} FLASH_SECT;
```

1.3.3 FAPI_CORE_SELECTOR

This enumeration is used to indicate which core the user is addressing when one or more cores exist on a device.

```
typedef enum
{
   FAPI_MASTER_CORE,    // Master Core
   FAPI_SLAVE_CORE0    // First Slave Core
} FAPI_CORE_SELECTOR;
```

1.3.4 FAPI_WRITE_SIZE

This enumeration is used to indicate what Flash write size is supported by the device.

```
typedef enum
{
   FAPI_WRITE_16BIT,
   FAPI_WRITE_32BIT
} FAPI_WRITE_SIZE;
```

1.3.5 FAPI_ERROR_CODE

This enumeration defines the error codes that can be returned by functions.

```
typedef enum
{
    // Function completed successfully
    FAPI_ERROR_CODE_SUCCESS,
    // Generic Function Fail code
    FAPI_ERROR_CODE_FAIL,
   // State machine polling never returned ready and timed out
    FAPI_ERROR_CODE_STATE_MACHINE_TIMEOUT,
    // Returned if OTP checksum does not match expected value
    FAPI_ERROR_CODE_OTP_CHECKSUM_MISMATCH,
    // Returned if the Calculated RWAIT value exceeds 15
    FAPI_ERROR_CODE_INVALID_DELAY_VALUE,
    // Returned if the specified core does not exist
    FAPI_ERROR_CODE_INVALID_CORE,
   // Returned if address is not properly aligned on 32-bit boundary
    FAPI_ERROR_CODE_UNALIGNED_32BIT,
   // Returned if address is not properly aligned on 64-bit boundary
   FAPI_ERROR_CODE_UNALIGNED_64BIT,
   // Returned if address is not properly aligned on 128-bit boundary
   FAPI_ERROR_CODE_UNALIGNED_128BIT,
   // Returned if length of data/array is invalid
   FAPI_ERROR_CODE_INVALID_DATA_LENGTH
} FAPI_ERROR_CODE;
```

1.4 Macros

1.4.1 FAPI_GET_WRITE_SIZE

FAPI_GET_WRITE_SIZE(oFlash Control)	This macro returns the FAPI_WRITE_SIZE based on the Flash write size supported by the device.

1.4.2 Parity and ECC Address Translation

The following macros are included to simplify calculation of the appropriate Parity and ECC address from a given main Flash address **a**.

PAR_ADDR_TRANS_U32 (a)	Translate UINT32 * pointer 'a' to corresponding Parity Flash address
PAR_ADDR_TRANS_U16 (a)	Translate UINT16 * pointer 'a' to corresponding Parity Flash address
PAR_ADDR_TRANS_U8 (a)	Translate UBYTE * pointer 'a' to corresponding Parity Flash address
ECC_ADDR_TRANS_U32 (a)	Translate UINT32 * pointer 'a' to corresponding ECC Flash address
ECC_ADDR_TRANS_U16 (a)	Translate UINT16 * pointer 'a' to corresponding ECC Flash address
ECC_ADDR_TRANS_U8 (a)	Translate UBYTE * pointer 'a' to corresponding ECC Flash address
COTP_PAR_TRANS_U32 (a)	Translate UINT32 * pointer 'a' to corresponding COTP Parity Flash address
COTP_PAR_TRANS_U16 (a)	Translate UINT16 * pointer 'a' to corresponding COTP Parity Flash address
COTP_PAR_TRANS_U8 (a)	Translate UBYTE * pointer 'a' to corresponding COTP Parity Flash address
TIOTP_PAR_TRANS_U32 (a)	Translate UINT32 * pointer 'a' to corresponding TIOTP Parity Flash address
TIOTP_PAR_TRANS_U16 (a)	Translate UINT16 * pointer 'a' to corresponding TIOTP Parity Flash address
TIOTP_PAR_TRANS_U8 (a)	Translate UBYTE * pointer 'a' to corresponding TIOTP Parity Flash address
COTP_ECC_TRANS_U32 (a)	Translate UINT32 * pointer 'a' to corresponding COTP ECC Flash address
COTP_ECC_TRANS_U16 (a)	Translate UINT16 * pointer 'a' to corresponding COTP ECC Flash address
COTP_ECC_TRANS_U8 (a)	Translate UBYTE * pointer 'a' to corresponding COTP ECC Flash address
TIOTP_ECC_TRANS_U32 (a)	Translate UINT32 * pointer 'a' to corresponding TIOTP ECC Flash address
TIOTP_ECC_TRANS_U16 (a)	Translate UINT16 * pointer 'a' to corresponding TIOTP ECC Flash address
TIOTP_ECC_TRANS_U8 (a)	Translate UBYTE * pointer 'a' to corresponding TIOTP ECC Flash address

1.4.3 Flash_Prog_B

This macro makes a call to *Flash_Prog_Data_B()* by predefining the command parameter as CMND_PROG_DATA_MAIN. For additional information, see the description for *Flash_Prog_Data_B()*.

```
#define Flash_Prog_B(m_start,m_buff,m_length,m_core,m_delay,m_cntl,m_status)\
       Flash_Prog_Data_B(m_start,
                  m_buff,
                  m_length,
        m_core,
        m_delay,
        m_cntl,
        m_status,
        m_length,
        CMND_PROG_DATA_MAIN)
```

1.4.4 Flash_Verify_B

This macro makes a call to *Flash_Verify_Data_B()*. For additional information, see the description for *Flash_Verify_Data_B()*.

```
#define Flash_Verify_B(m_start,m_buff,m_length,m_core,m_cntl,m_status)\
       Flash_Verify_Data_B(m_start,
                  m_buff,
                  m_length,
        m_core,
        m_cntl,
        m_status,
        m_length)
```

1.4.5 OTP_Prog_B

This macro makes a call to *Flash_Prog_Data_B()* by predefining the command parameter as CMND_PROG_DATA_COTP. For additional information, see the description for *Flash_Prog_Data_B()*.

```
#define OTP_Prog_B(m_start,m_buff,m_length,m_core,m_delay,m_cntl,m_status)\
       Flash_Prog_Data_B(m_start,
                  m_buff,
                  m_length,
        m_core,
        m_delay,
        m_cntl,
        m_status,
        m_length,
        CMND_PROG_DATA_COTP)
```

2 API Description

The F035 Flash API is a library of routines which, when called with the proper parameters in the proper sequence, erases, programs or verifies 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. Interrupts must be disabled during execution of API functions. Aborts during execution of API functions may leave the flash in an indeterminate state. The API internally adjusts the RWAIT value accordingly so that the Flash Oscillator frequency (Fosc) does not exceed the frequency configured by the max_fosc_delay element programmed to the TI OTP when the Flash State Machine is being utilized.

2.1 Pulse Size and Limits

NOTE: The values shown in Table 1 are the current target values for these parameters, but these values can be overridden in the TI OTP on per Flash bank basis as needed depending on the performance of a given device.

Table 1. Pulse Size and Limits

Operation	Pulse Length	Maximum Number of Pulses
Program	6us	200
Erase	5ms	4095
Compact	500us	2000

2.2 Build Environment

The current version of the Flash API library was built with version 4.4.11 of the Texas Instruments TMS470 C compiler using TIABI object format (pf035a_api_tiabi.lib), EABI object format (pf035a_api_eabi.lib) and EABI object format compiled with vfp option (pf035a_api_eabi_vfp.lib). The functions are written in C in 16-bit instruction mode.

The following defines and options are common to all libraries:

```
Defines:  PLATFORM, FLASH_API_VERSION=12h, F035a, FLASH_API_TECH_ID=0x13010000,
FLASH_API_MAJOR_VERSION=1, FLASH_API_MINOR_VERSION=4, F035_4MB_WRAPPER

Options:  -o2 -al -mt --issue_remarks --c_src_interlist
```

The following compiler options were used in addition to the common compiler options to create the TIABI compiled Flash API library **pf035a_api_tiabi.lib:**

```
-mv4
--abi=tiabi
--symdebug:coff
```

NOTE: The deprecated library name of this version is still available – pf035a_api.lib.

The following compiler options were used in addition to the common compiler options to create the EABI compiled Flash API library **pf035a_api_eabi.lib:**

```
-mv4
--abi=eabi
--symdebug:none
```

The following compiler options were used in addition to the common compiler options to create the EABI vfp compiled Flash API library **pf035a_api_eabi_vfp.lib:**

```
-mv7r4
--float_support=VFPv3D16
--abi=eabi
--symdebug:none
```

2.3 API Include Files and Recommended Usage

The Flash API includes several header files to be included when using these functions in customers code. They are as follows:

f035.h --	This contains definitions for the F035 process Flash devices. This defines the F035 that is used by the flash470.h include file and, therefore, should be included first.
flash470.h –	This is the main include file that contains the API function definitions, structures, and macros.
Flash470ErrorDefines.h –	This contains the error code return values. This file is already included by flash470.h and, therefore, does not need to be explicitly included.

2.4 API Defines and their Usage

The Flash API includes makes use of several defines to determine specific sections of code to use. They are as follows:

PLATFORM –	This defines that the device is a Platform architecture. This should be defined before referencing the f035.h include file.
DUALCPU –	This is not used by this API and should not be defined. It is there for backwards capability in flash470.h for F05 architectures.
F035 –	This defines the Flash Technology used by the API. This is already defined in the f035.h include file.
F05/F10 –	These are defines for other Flash Technologies and exist for backwards compatibility. They must not be defined in the user's code.
F035_4MB_WRAPPER –	This defines the Flash Wrapper technology the API and other code is compiled for. This is defined by default in the f035.h include file.
F035_16MB_WRAPPER –	This is maintained for backwards capability with older Flash Wrapper technologies. It must not be defined in the user's code.

3 API Functions

3.1 *Fapi_CalculateParity()*

[calculate Flash Parity]

```
FAPI_ERROR_CODE Fapi_CalculateParity(UINT32 *pu32Start,
                       UINT16 *pu16ParityData,
                             UINT32 u32Length
                                ); 
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Pointer to the location in Flash where the Parity will be calculated. The address of this pointer is used for the address portion of the Parity calculation routine.
pu16ParityData	UINT16 *	Pointer to buffer where Parity data is stored.
u32Length	UINT32	Number of 32-bit words for which Parity is calculated. The value of length is required to be multiples of 4.

Return Value

This function returns an FAPI_ERROR_CODE value of either FAPI_ERROR_CODE_SUCCESS for a valid operation or FAPI_ERROR_CODE_FAIL due to:

1. Fails if “u32Length” is not a multiple of 4

Description

This function calculates Parity for a device. The user must provide a data buffer where the newly calculated Parity bits are stored. This data can then be programmed in Flash using the usual programming method. The number of elements in the "pu16ParityData" buffer must be equal to the number of 32-bit words of data, "u32Length" for which the Parity is calculated.

3.2 *Fapi_getApiVersion32()*

[get 32bit BCD API version and Tech ID]

```
UINT32 Fapi_getApiVersion32(void)
```

Parameters

None

Return Value

Unsigned 32-bit integer (UINT32)

Description

This function returns the programming algorithm version number in BCD (Binary Coded Decimal) notation and Tech ID code. For example, the number 0x0007 represents version 0.07 and the Tech ID code is listed below.

```
0xAABBCCDD
    AA = Tech
        01 -> F10
           02 -> F05
           12 -> Platform F05
           03 -> F035
           13 -> Platform F035
    BB = Revision
        00 -> original
        01 -> A
        02 -> B
```

```
CC = Major Version
DD = Minor Version
```

3.3 Fapi_HardwareCalculateEcc()

[calculate ECC using R4 core]

```
FAPI_ERROR_CODE Fapi_HardwareCalculateEcc(UINT32 *pu32Start,
                                UINT16 *pu16EccData,
                                UINT32 u32Length,
                                FLASH_ARRAY_ST oFlashControl
                                  );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Pointer to the location in Flash where the Parity will be calculated. The address of this pointer is used for the address portion of the ECC calculation routine. This address needs to be aligned on a 128-bit boundary.
pu16EccData	UINT16 *	Pointer to buffer where ECC data is stored.
u32Length	UINT32	Number of 32-bit words for which ECC is calculated. The "pu16EccData" buffer size needs to be "u32Length number of 16-bit words. The value of length is required to be multiples of 4.
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash region resides.

Return Value

This function returns an FAPI_ERROR_CODE value of either FAPI_ERROR_CODE_SUCCESS for a valid operation or FAPI_ERROR_CODE_FAIL due to:

1. Fails if “u32Length” is not a multiple of 4
2. Fails if “pu32Start” is not aligned on a 128-bit boundary

Description

This function calculates ECC for a device that utilizes the R4 ECC encoding scheme. This function takes advantage of the F035 Flash wrapper ECC calculation logic to optimize performance. Proper ECC is calculated on the data located at "pu32Start". Since the R4 ECC encoding scheme includes address, the data for which the ECC is calculated must already be programmed at the desired location. This routine does not work on a data buffer that is located in RAM. The user must provide a data buffer where the newly calculated ECC bits are stored. This data can then be programmed in Flash using the usual programming method. The number of elements in the "pu16ECCData" buffer must be equal to the number of 32-bit words of data, "u32Length" for which ECC is calculated.

3.4 Fapi_SetupFlashPump()

[Select the correct Flash Pump with multiple Flash Wrappers]

```
FAPI_ERROR_CODE Fapi_SetupFlashPump(FAPI_CORE_SELECTOR oCoreSelector)
```

Parameters

Parameter	Type	Purpose
oCoreSelector	FAPI_CORE_SELECTOR	Specifies which core to switch the Flash Pump to on multi core devices.

Return Value

This function returns an FAPI_ERROR_CODE value of either FAPI_ERROR_CODE_SUCCESS for a valid operation or FAPI_ERROR_CODE_INVALID_CORE if the core specified in oCoreSelector is invalid.

Description

This function sets up the Flash Pump to the core specified by the user on devices that have multiple cores using a System Slave module. The user must be careful when calling other Flash Functions that the delay value matches the core selected.

3.5 *Feed_Watchdog_V()*

[prevents AWD and DWD watchdog resets]

```
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 that performs a busy-wait on the Flash State Machine or some other time-consuming operation. 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) and the Digital Watchdog (DWD). This function is called at least every 200 µs. This function cannot be executed from the bank that is being programmed or erased. The following functions call *Feed_Watchdog_V()*:

Flash_Blank_B()	*Flash_Read_V()*	*Flash_Vt_Verify_Data_B()*
Flash_Erase_Sector_B()	*Flash_Verify_Data_B()*	*Flash_PSA_Vt_Verify_B()*
exec_pulse()	*Flash_PSA_Verify_B()*	*Flash_Prog_Wide_B()*
Flash_Prog_Data_B()	*verify_read()*	
Flash_PSA_Calc_U32()	*Flash_Vt_Read_V()*	

3.6 *Flash_Aux_Engr_U16()*

[reads the 16-bit auxiliary engineering ID]

```
UINT16 Flash_Aux_Engr_U16(UINT16 *address,
                          FLASH_ARRAY_ST cntl
                          );
```

Parameters

Parameter	Type	Purpose
address	UINT16 *	Pointer to the location in the TI engineer row to be read
cntl	FLASH_ARRAY_ST	Pointer to Flash array control register structure

Return Value

This function returns an unsigned 16-bit value read from the specified address of the TI engineering row.

Description

This function can be used to read one 16-bit value from the TI engineering row.

3.7 *Flash_Blank_B()*

[blank checks a given region of Flash]

```
BOOL Flash_Blank_B(UINT32 *pu32Start,
                   UINT32 u32Length,
                   FLASH_CORE oFlashCore,
                   FLASH_ARRAY_ST oFlashControl,
                   FLASH_STATUS_ST *oFlashStatus
          );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Pointer to the first word in Flash that will be blank-checked
u32Length	UINT32 *	Number of 32-bit words to be blank-checked
oFlashCore	FLASH_CORE	Bank select (0-7) of region of Flash being blank checked
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash region resides

oFlashStatus	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information. Note that word length must be passed to the Flash_Blank_B function via the status.stat1 element:
		status.stat1=128 => No Parity and ECC
		status.stat1=132 => Include Parity bits
		status.stat1=144 => Include ECC bits

NOTE: This function must be executed from RAM.

This function calls *Feed_Watchdog_V()* that also must be executed from RAM.

Return Value

This function returns a boolean value. Pass = 1 = Region is blank, Fail = 0 = Region is not blank (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. This value should be FSPRD_RDM1 (see f035.h listing), which indicates the region did not read as blank in read margin 1 mode.

Description

This function verifies that the Flash has been properly erased by using the read-margin 1 mode 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. The user must also specify via the status.stat1 parameter whether to also blank check the corresponding parity and ECC bits for the given main Flash address range. For more details, see the table above.

3.8 *Flash_Compact_B()*

[compact sector using Flash state machine]

```
BOOL Flash_Compact_B(UINT32 *pu32Start,
                     FLASH_CORE oFlashCore,
                     FLASH_SECT oFlashSector,
                     UINT32 u32Delay,
                     FLASH_ARRAY_ST oFlashControl,
                     FLASH_STATUS_ST *oFlashStatus
                     );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Points to the first word in the Flash sector that will be compacted (the first address in the sector)
oFlashCore	FLASH_CORE	Bank select (0-7) of sector being compacted
oFlashSector	FLASH_SECTOR	Sector select (0-15) of sector being compacted
u32Delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document.
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash region resides.
oFlashStatus	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 3.22 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.

3.9 Flash_EngInfo_V()

[retrieve lot, wafer, X, Y, flowchk information from OTP]

```
void Flash_EngInfo_V(UINT32 *start,
                     FLASH_ARRAY_ST cntl,
                     FLASH_ENGR_INFO_ST *info
                     );
```

Parameters

Parameter	Type	Purpose
start	UINT32 *	Pointer to the first word in the Flash array (first bank, first sector)
cntl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash region resides
info	FLASH_ENGR_INFO_ST *	Pointer to a structure that is the same size as a FLASH_STATUS_ST, but is laid out differently for the purpose of continuing engineering information.

Return Value

The following values are stored in the FLASH_ENGR_INFO_ST structure. The numbers (except FlowCheck, which is binary) are in BCD format.

```
#define DevID AsicId  /* Left for Backwards Compatibility */

typedef struct
{
  UINT32    AsicId;
  UINT32    LotNo;
  UINT16    FlowCheck;
  UINT16    WaferNo;
  UINT16    Xcoord;
  UINT16    Ycoord;
} FLASH_ENGR_INFO_ST;
```

Description

This function reads data stored in the Engineering row of the Flash and returns the parameters specified above. All values (except FlowCheck) are stored in Binary Coded Decimal (BCD) format (if LotNo = 0x07123456, then the actual lot number should be interpreted as decimal 7123456). FlowCheck is a binary value whose lower 8 bits are used internally by TI to track whether a device has passed each required test step during production testing.

3.10 Flash_Erase_B()

```
BOOL Flash_Erase_B(UINT32 *pu32Start,
                   UINT32 u32Length,
                   FLASH_CORE oFlashCore,
                   FLASH_SECT oFlashSector,
                   UINT32 u32Delay,
                   FLASH_ARRAY_ST oFlashControl,
                   FLASH_STATUS_ST *oFlashStatus
                   );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Points to the first word in the Flash sector that will be erased. Note: This must correspond to the first address in the sector.
u32Length	UINT32	Number of 32-bit words in the sector
oFlashCore	FLASH_CORE	Bank select (0-7) of sector being erased
0FlashSector	FLASH_SECTOR	Sector select (0-31) of sector being erased
u32Delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash sector resides
oFlashStatus	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 assure 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. For more information, see Recommended Erase Flows in Section 5.

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 (for description, see Flash_Erase_Status_U16)
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 the above Parameters 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 F035 Flash State Machine only requires a correct core and start address value to correctly erase a sector. See Section 5.2 for recommended Erase flow guidelines.

3.11 Flash_Erase_Bank_B()

```
BOOL Flash_Erase_Bank_B(UINT32 *pu32Start,
                        UINT32 u32Length,
                        FLASH_CORE oFlashCore,
                        UINT32 u32Delay,
                        FLASH_ARRAY_ST oFlashControl,
                        FLASH_STATUS_ST *oFlashStatus
                        );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Points to the first word in the Flash bank that will be erased. Note: This must correspond to the first address in the bank.
u32Length	UINT32	Number of 32-bit words in the bank
oFlashCore	FLASH_CORE	Bank select (0-7) of bank being erased
u32Delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document.
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where Flash bank resides
oFlashStatus	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information
		Note: Determining a bank is blank using Flash_Blank_B is insufficient to make sure a bank 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 assure a bank is adequately erased, erase must be performed on the bank (regardless of the contents).
		If the bank 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_Bank_B to disable preconditioning and speed up erase. Non-blank banks 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. For more information, see *Recommended Erase Flows* in Section 5.

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 (for description, see Flash_Erase_Status_U16)
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 bank until it is completely erased, and to collect pulse count information. This function is unique from *Flash_Erase_Sector_B()* and similar to Flash_Erase_B in that it allows for the disabling of preconditioning during erase (see the above Parameters table). The length parameter is ignored by the function. The F035 Flash State Machine only requires a correct core and start address value to correctly erase a bank. For recommended Erase Bank flow guidelines, see Section 5.3.

3.12 Flash_Erase_Sector_B()

[erases entire sector with precondition]

```
BOOL Flash_Erase_Sector_B(UINT32 *pu32Start,
                          UINT32 u32Length,
                          FLASH_CORE oFlashCore,
                          FLASH_SECT oFlashSector,
                          UINT32 u32Delay,
                          FLASH_ARRAY_ST oFlashControl
                         );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Points to the first word in the Flash sector that will be erased. Note: This must correspond to the first address in the sector.
u32Length	UINT32	Number of 32-bit words in the sector
oFlashCore	FLASH_CORE	Bank select (0-7) of sector being erased
0FlashSector	FLASH_SECTOR	Sector select (0-31) of sector being erased (actual value is ignored)
u32Delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document.
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash sector resides

Return Value

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

Description

This function erases a sector. Preconditioning is enabled by default and cannot be disabled, as is possible using Flash_Erase_B or Flash_Erase_Bank_B. The length parameter and sector number are not used in F035. For recommended Erase Sector flow guidelines, see Section 5.4.

3.13 Flash_Prog_Data_B()

[program data to any Flash using FSM]

```
BOOL Flash_Prog_Data_B (UINT32 *pu32Start,
                        UINT32 *pu32Buffer,
                        UINT32 u32Length,
                        FLASH_CORE oFlashCore,
                        UINT32 u32Delay,
                        FLASH_ARRAY_ST oFlashControl,
                        FLASH_STATUS_ST *poFlashStatus,
                        UINT32 u32BufferLength,
                        UINT16 u16Command
                         );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Points to the first word in Flash that will be programmed
pu32Buffer	UINT32 *	Pointer to the starting address of a buffer with data to program
u32Length	UINT32	Number of 32-bit words to program
oFlashCore	FLASH_CORE	Bank select (0-7) of region being programmed
u32Delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document.
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash sector resides
oFlashStatus	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information
U32BufferLength	UINT32	Cyclic buffer used to program the Flash. The buffer will be repeatedly programmed into Flash in the amount of buflen until length is reached.
u16Command	UINT16	Programming command to pass to state machine. Allowed values are (see F035.h): CMND_PROG_DATA_MAIN CMND_PROG_CBIT_MAIN CMND_PROG_DATA_COTP CMND_PROG_CBIT_COTP Any other values cause the function to return a fail. Note that OTP address translation is performed when commands to program the Customer OTP are issued.

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 programs the Flash from the starting address “start” for “length” 32-bit words. This function only programs the Flash; it does 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 Flash_Erase_Sector_B or Flash_Erase_B functions can be used to erase an entire sector. For more details on programming flow guidelines, See Section 5.6.

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

3.14 *Flash_PSA_Calc_U32()*

[calculates PSA for given Flash region]

```
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 Flash that 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 where the Flash region resides

NOTE: This function must be executed from RAM.

This function calls *Feed_Watchdog_V()* and *psa_u32()* that also must be executed from RAM.

Return Value

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

Description

This function uses 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. Compare the result of the final region's calculation against the known PSA value for the entire flash.

NOTE: Bank boundaries must not be crossed by any single region

3.15 *Flash_PSA_Verify_B()*

[fast verify of Flash using PSA]

```
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 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 where the Flash region resides
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information

NOTE: This function must be executed from RAM.

This function calls *Feed_Watchdog_V()* and *psa_u32()* that also must be executed from RAM.

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 verifies proper programming by using normal read, read-margin 0, read-margin 1 modes, and generating a 32 bit PSA checksum for the data in the region in each mode. Verification starts from the start address “start” and checks “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”.

3.16 *Flash_Read_V()*

[read contents of Flash region to buffer]

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

Parameters

Parameter	Type	Purpose
start	UINT32 *	Pointer to the first word in Flash that will be read
buff	UINT32 *	Pointer to the buffer in RAM where the read data will be copied
length	UINT32	Number of 32-bit words to be read
mode	UINT32	Mode in which to read 0 = Normal Read mode 1 = Read Margin 0 mode 2 = Read Margin 1 mode All other values => Normal read mode
core	FLASH_CORE	Bank select (0-7) of bank within which customer OTP sector resides
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module

NOTE: This function must be executed from RAM.

This function calls *Feed_Watchdog_V()* that also must be executed from RAM.

Return Value

This function does not return any value, but "length" words starting at address "buff" will be written with data from Flash.

Description

This function will read data stored in the Flash in the given mode and copy it to successive locations pointed to by "buffer".

3.17 Flash_Sector_Select_V()

[sector erase, prog, compact enable]

```
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 where the Flash sector resides

Return Value

This function has no return value.

Description

This function is called by all functions that attempt erase or programing within Flash sectors, but is generally not called except from within other functions. 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 disables programming and erases 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.

3.18 Flash_Start_Async_Command_B()

[issue command to FSM]

```
BOOL Flash_Start_Async_Command_B(UINT32 *pu32Start,
                         UINT16 u16Command,
                         UINT32 u32Data,
                                 FLASH_ARRAY_ST oFlashControl
                         );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Points to the first word in Flash sector that the specified command operates on
u16Command	UINT16	Command to be sent to the F035 State machine
u32Data	UINT32	Commands appropriate data to send to the F035 state machine after the command is issued
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash sector resides

Return Value

This function always returns a Boolean value of Pass = 1.

Description

This function is used to cause the Flash State Machine to execute a specified command. This function returns immediately and does not check on the status of the executed command. u32Data is typecast to the appropriate size depending on the device supporting either 32-bit or 16-bit writes.

3.19 *Flash_Start_Compact_B()*

[issue compaction command to FSM]

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

Parameters

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in the Flash sector that is to be compacted
core	FLASH_CORE	Bank select (0-7) of sector being compacted
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document.
cntl	FLASH_ARRAY_ST	Flash Control Base address of module where Flash sector resides

Return Value

This function returns a Boolean value. Pass = 1, Fail = 0. A failure indicates that the delay value was too large (see Section 4).

Description

This function is used to start the compaction of a sector. Either the *Flash_Compact_Status_U16()* (that calls Flash_Status_U16) or the *Flash_Status_U16()* function can be directly used to determine when the compaction is complete. This function allows the user to perform some other tasks while the state machine is performing compaction such as feeding a watchdog or servicing the peripherals. No attempts should be made to read from Flash locations in the same bank as the sector being compacted until the compaction completes. For an example implementation of Flash_Start_Compact_B, see Section 3.30.5, Flash_Compact_Status_U16. For recommended Erase flow guidelines, see Section 5.2 - Section 5.5.

3.20 Flash_Start_Erase_B()

[issue erase command to FSM]

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

Parameters

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in the Flash sector that is to be erased
core	FLASH_CORE	Bank select (0-7) of sector being erased
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
cntl	FLASH_ARRAY_ST	Flash Control Base address of module where Flash sector resides

Return Value

This function returns a Boolean value. Pass = 1, Fail = 0. A failure indicates that the delay value was too large (see Section 4).

Description

This function is used to start the erase of a sector. Either the *Flash_Erase_Status_U16()* (that calls Flash_Status_U16) or *Flash_Status_U16()* functions can be directly used to determine when the erase is complete. These functions allow the user to perform some other tasks, such as feeding a watchdog, while a sector is being erased. No attempts should be made to read from Flash locations in the same bank as the sector being erased until the erase completes. This function does not support the disabling of preconditioning. For an example implementation of Flash_Start_Erase_B, see Section 3.30.7, *Flash_Erase_Status_U16()*. For recommended Start Erase flow guidelines, see Section 5.5.

3.21 *Flash_Start_Prog_B()*

[issue erase command to FSM]

```
BOOL Flash_Start_Prog_B(UINT32* pu32Start,
                        UINT32  pu32Data,
                        FLASH_CORE oFlashCore,
                        UINT32 u32Delay,
                        FLASH_ARRAY_ST oFlashControl
                         );
```

Parameters

Parameter	**Type**	**Purpose**
pu32Start	UINT32 *	Points to the first word in the Flash sector that is to be programmed
pu32Data		32-bit data to be programmed
oFlashCore	FLASH_CORE	Bank select (0-7) of sector being erased
u32Delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where the Flash sector resides

Return Value

This functions returns TRUE if Flash_Start_Async_Command_B is run properly.

Description

This function is used to initiate the Flash State Machine to start the programming of a 32-bit data at the address pointed to by start. This function returns immediately. The user has to use Flash_Status_U16 in order to determine if the programming occurred successfully.

3.22 *Flash_Status_U16()*

[retrieve current FSM status]

```
UINT16 Flash_Status_U16(FLASH_ARRAY_ST cntl);
```

Parameters

Parameter	**Type**	**Purpose**
cntl	FLASH_ARRAY_ST	Flash Control Base address of module whose status is being checked

Return Value

This function returns a 16-bit value that defines the status of the program and erase state machine.

31-15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	ILA	DBT	PGV	PCV	EV	CV	BUSY	ERS	PGM	INVDAT	CSTAT	VSTAT	ESUSP	PSUSP	SLOCK

Bits 31-15	**Reserved** - Read values are 0. Writes have no effect.

Bit 14	**ILA** - Illegal Address When set, indicates that an illegal address is detected. Three conditions can set illegal address flag. 1. Writing to a hole (un-implemented logical address space) within a Flash bank 2. Writing to an address location to an un-implemented Flash space 3. Input address for write is decoded to select a different bank from the bank ID register
Bit 13	**DBF** - Disturbance Test Fail When set, indicates that a disturbance is detected when Program Sector command is issued during disturbance test mode. Disturbance is created when an un-selected bit is programmed to 0 when programming a selected bit. The sets this flag when it compares the unselected bits to 1s and fails.
Bit 12	**PGV** - Program Verify When set, indicates that a word is not successfully programmed after the maximum allowed number of program pulses are given for program operation.
Bit 11	**PCV** - Precondition verify When set, indicates that a sector is not successfully preconditioned (pre-erased) after the maximum allowed number of program pulses are given for precondition operation for any applied command such as Erase Sector command. During Precondition verify command, this flag is set immediately if a Flash bit is found to be 1. If Precondition Terminate Enable bit is cleared then PCV is not set when preconditioning fails. Setting Precondition Terminate Enable bit allows FSM to terminate the command immediately if precondition operation fails during any type of erase commands.
Bit 10	**EV** - Erase verify When set, indicates that a sector is not successfully erased after the maximum allowed number of erase pulses are given for erase operation. During Erase verify command, this flag is set immediately if a bit is found to be 0.
Bit 9	**CV** - Compact Verify When set, indicates that a sector contains one or more bits in depletion after the maximum allowed number of compaction pulses are given for compact operation. During compact verify command, this flag is set immediately if a bit is found to be 1.
Bit 8	**Busy** - Busy When set, this bit indicates that a program, erase, or suspend operation is being processed.

Bit 7	**ERS** - Erase Active When set this bit indicates that the Flash module is actively performing an erase operation. This bit is set when erasing starts and is cleared when erasing is complete. It is also cleared when the erase is suspended and set when the erase resumes.
Bit 6	**PGM** - Program Active When set this bit indicates that the Flash module is currently performing a program operation. This bit is set when programming starts and is cleared when programming is complete. It is also cleared when programming is suspended and set when programming resumes.
Bit 5	**INVDAT** - Invalid Data When set, this bit indicates that the user attempted to program a “1” where a “0” was already present. This bit is cleared by the Clear Status command.
Bit 4	**CSTAT** - Command Status When set, this bit informs the host that the program, erase, or validate sector command failed and the command was stopped. This bit is cleared by the Clear Status command.
Bit 3	**5VSTAT** - VDD5V Status When set, this bit indicates if the 5.0 V power supply dipped below the lower limit allowable during a program or erase operation. This bit is cleared by the Clear Status command.
Bit 2	**ESUSP** - Erase Suspend When set, this bit indicates that the Flash module has received and processed an erase suspend command. This bit remains set until the erase resume command has been issued.
Bit 1	**PSUSP** - Program Suspend When set, this bit indicates that the Flash module has received and processed a program suspend command. This bit remains set until the program resume command has been issued.
Bit 0	**SLOCK** - Sector Lock Status When set, this bit indicates that the operation was halted because the target sector was locked for erasing and programming either by the sector protect bit or by write protection key logic. This bit is cleared by the Clear Status command.

Description

This function is used to check the status of a Flash State Machine command operation started by Flash_Start_Compact_B, FLash_Start_Erase_B, or Flash_Start_Command_B. These functions allow the user to perform some other tasks, such as feeding a watchdog, while a sector is being operated on by the state machine. No attempts should be made to read from Flash locations in the same bank as the sector being operated on by the state machine until the command completes (indicated by BUSY going back to 0). Once the BUSY bit is low, indicating command completion, it is recommended to flag a failure if any other status bits read as anything other than 0.

Example Function Usage

```
UINT16 result;
if (Flash_Start_Compact_B (start, core, delay, cntl)
{
    // feed watchdog while BUSY is high
    while ((result=Flash_Status_U16(cntl))&0x100)
    {
        Feed_Watchdog_V();
    }
    if (result!=0)  // fail if any other bits set
    {
        compact_fail();
    }
}
else
{
    compact_start_fail();
}
```

3.23 Flash_Verify_Data_B()

[verify Flash against 'cyclical' buffer]

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

Parameters

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in Flash to be verified
buff	UINT32 *	Pointer to the starting address of the 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 where the Flash region resides
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information
buflen	UINT32	Length of cyclical data buffer in 32-bit words

NOTE: This function must be executed from RAM.

This function calls *Feed_Watchdog_V()* that also must be executed from RAM.

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, 1 means read margin zero failure, and 2 means read margin one failure

Description

This function verifies proper programming by using normal read, read-margin 0, and read-margin 1 modes. Verification starts from the start address “start” and checks “length” words from the start address. The parameter 'buflen' is used to allow for a cyclical data buffer. This means that 'buflen' number 32-bit words starting at the location pointed to by 'buff' are repeatedly used to verify the data until length number of 32-bit words have been verified.

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 compares the data stored in the Flash to the data stored in the buffer pointed to by “buffer”.

3.24 Flash_Verify_Zeros_B()

[verify Flash region contains all 0's]

```
BOOL Flash_Verify_Zeros_B(UINT32 *start,
                          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 Flash to be verified for all 0x00000000
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 where the 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 zeros
stat2	Data at first location failing verify zeros
stat3	Data expected at failing address (0x00000000)
stat4	Value of 0 means normal read failure, 4 means read margin zero failure

Description

This function verifies the result of running Flash_Zeros_B by using normal read and read-margin 0 modes. Verification starts from the start address “start” and checks “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”.

3.25 Flash_Zeros_B()

[program all 0's to a region of Flash]

```
BOOL Flash_Zeros_B(UINT32 *start,
                   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 Flash that will be programmed
length	UINT32	Number of 32-bit words to program
core	FLASH_CORE	Bank select (0-7) of bank within region being programmed
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
cntl	FLASH_ARRAY_ST	Flash Control Base address of module where the 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 (only on failure):

stat1	If fail, first address that fails to program to zeros
stat2	If fail, data at first failing address
stat3	If pass, Total number of pulses applied to program all locations to zero, else if fail this is the last MSTAT value read from the state machine to help determine the fail mode.
stat4	Maximum number of pulses applied to a single location

Description

This function is not needed in F035 since the erase routines automatically clear the sector before erasing. It is included for compatibility.

3.26 get_timing()

[calculate timing based on FCLK]

```
UINT32 get_timing(UINT32 n,
                  UINT32 delay,
                  UINT32 presc,
                  UINT32 shft,
                  UINT32 min,
                  UINT32 max
                  );
```

Parameters

Parameter	Type	Purpose
n	UINT32	Count of 500 ns reference clocks to give magnitude of intended timing. For example, if timing is intended to be 20 µs, then n=40.
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
presc	UINT32	Prescaler and Flash clock divider. This value should be inferred from the RWAIT field in the FRDCNTL register (bits 11:8), presc=(2*(RWAIT+1))
shft	UINT32	it shift inferred by prescaler divider. presc=1<<shft This argument is passed instead of calculated in order to save cycles.
min	UINT32	Minimum allowed clocks
max	UINT32	Maximum allowed clocks

Return Value

This function returns an unsigned 32 bit value (UINT32) that is clock count scaled according to the given arguments.

Description

This function is used to properly scale the values intended for the Flash State machine timing registers according to the current delay parameter and RWAIT setting. If a certain timing requires a minimum number of clocks, it should be specified by the min parameter, otherwise 0 should be specified. If a timing has a maximum allowed number of clocks (based on the bit field in the register), then that value should be specified as the max parameter. Generally this function is only called by init_state_machine or setup_state_machine.

3.27 *get_presc_shift()*

[determine bit shift of FCLK and HCLK ratio]

```
UINT32 get_presc_shift(FLASH_ARRAY_ST cntl);
```

Parameters

Parameter	Type	Purpose
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module being initialized.

Return Value

This function returns an unsigned 32 bit value (UINT32), which is the number of bits to shift the timing values passed to *wait_delay()* in the COD routines based on the ratio of HCLK to FCLK.

Description

When the software interface is used to perform program, erase, and compaction, pulse lengths are based on calls to wait-delay. The timing values initially calculated by setup_state_machine are based on FCLK cycles, but in software mode, these values need to be re-scaled based on HCLK. get_presc_shift calculates the ratio of HCLK to FLCK in powers of two and returns the appropriate bit shift with which to multiply the FCLK based timings to achieve the same timings using HCLK cycles.

The basic assumption of get_presc_shift is that FCLK=(HCLK/(2*(RWAIT+1))).

3.28 *psa_u32()*

```
UINT32 PSA_U32(UINT32 *pdwStart,
               UINT32 dwLength,
               UINT32 dwInitialSeed
               );
```

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 seed value for the PSA calculation

Return Value

This function returns an unsigned 32-bit integer that is the PSA value computed by the CPU based on the length data words read from the start address.

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 regarding crossing bank boundaries. Generally, this function is called from another function to make sure the PSA is calculated in the proper Flash read mode. This function is deprecated and can only be used directly from the TIABI version of the compile. Use the macro definition PSA_U32 instead.

3.29 setup_state_machine()

[setup state machine timings from OTP]

```
FAPI_ERROR_CODE setup_state_machine(UINT32 *pu32Start,
                                    FLASH_CORE oFlashCore,
                                    UINT32 u32Delay,
                                    FLASH_ARRAY_ST oFlashControl
                                   );
```

Parameters

Parameter	Type	Purpose
Pu32Start	UINT32 *	Points to the first word to be read.
oFlashCore	FLASH_CORE	Bank select (0-7) of the bank in which the customer OTP sector resides
U32Delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
oFlashControl	FLASH_ARRAY_ST *	Flash Control Base address of module where the Flash State Machine resides

NOTE: This function must be executed from RAM only!

This function is also called by the following functions:

- *Flash_Compact_B()*
- *Flash_Erase_Bank_B()*
- *Flash_Erase_B()*
- *Flash_Erase_Sector_B()*
- *Flash_Prog_Data_B()*
- *Flash_Start_Command_B()*
- *Flash_Vt_Read_V()*
- *Flash_Vt_Verify_Data_B()*
- *Flash_PSA_Vt_Verify_B()*

This function calls the following functions that also must be executed from RAM:

- *get_timing()*
- *Feed_Watchdog_V()*

Description

This function is used to setup timings and CT values in the Flash State machine. **It is a replacement to the *deprecated* init_state_machine function.** The default timings and CT values are read from the TI OTP sector along with a 32-bit checksum for the 16-bit data. If the checksum matches the sum of all the 16 bit values, then the timing values from the TI OTP are scaled based on the delay parameter and stored to the appropriate register, and the appropriate CT values are written to their appropriate registers. If the checksum does not match, then default timings and CT values are used. Generally, this function is called by all other functions that utilize the Flash State Machine (Flash_Erase_B, Flash_Prog_B, and so forth).

Return Value

This function returns a FAPI_ERROR_CODE value:

FAPI_ERROR_CODE_SUCCESS	-> Operation completed successfully
FAPI_ERROR_CODE_OTP_CHECKSUM_MISMATCH	-> Device OTP checksum does not Match expected value
FAPI_ERROR_CODE_INVALID_DELAY_VALUE	-> Delay value passed to the function generated an invalid RWAIT value

3.30 wait_delay()

```
void wait_delay(volatile UINT32 u32Delay);
```

Parameters

Parameter	Type	Purpose
U32Delay	volatile UINT32	Count value based on the number of 2 µs increments to delay scaled based on the delay parameters found in Table 4, Flash Delay Parameter Values in Section 4 of this document.

Description

This function is used to enable a non-interruptible delay of 'delay' * 4 * 1/HCLK (in MHz). This function is deprecated and can only be used directly from the TIABI version of the compile. Use the macro definition WAIT_DELAY instead.

Deprecated API Functions

3.30.1 aligned_byte_width()- Deprecated

[calculate byte length]

```
UINT32 aligned_byte_width(UINT32 address,
                          UINT32 length,
                          UINT32 width
                          );
```

Parameters

Parameter	Type	Purpose
Address	UINT32	Absolute address value
Length	UINT32	Length in bytes
Width	UINT32	Alignment width in bytes (1, 2, or 4)

Return Value

This function returns a 32-bit unsigned integer (UINT32). This is the number of bytes available in the current 'width' aligned range of bits starting at the specified 'address' and within the specified 'length.'

Description

The purpose of this function is to calculate a byte length no greater than 'width' and no less than 'length', which is byte aligned according to 'width' bytes with respect to 'address.' This is mainly used to align the bytes verified in a single pass when *Flash_Prog_Wide_B()* is called.

3.30.2 ceil_div_by_x()- Deprecated

[divide by x and round up result with remainder]

```
UINT32 aligned_byte_width(UINT32 val,
                          UINT32 denom
                          );
```

Parameters

Parameter	Type	Purpose
Val	UINT32	Numerator to be divided
Denom	UINT32	Denominator

Return Value

This function returns a 32-bit unsigned integer (UINT32), which is the result of dividing val by denom, and rounding up the result in case there is a remainder.

Description

The purpose of this function is to simply perform a ceiling (rounding up) type divide in software, which is useful for making sure timings always round up to the next whole integer value.

3.30.3 Flash_API_Version_U16()- Deprecated

[get 16bit BCD API version]

```
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 Binary Coded Decimal (BCD) notation. For example, the number 0x0007 represents version 0.07. The direct call of this function is deprecated and is replaced by the *Fapi_getApiVersion32().*

3.30.4 Flash_Calculate_Parity_B()- Deprecated

[Calculate Flash Parity]

```
BOOL Flash_Calculate_Parity_B(UINT32 *pu32Start,
                              UINT16 *pu16ParityData,
                                 UINT32 u32Length
                                    );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Pointer to the location in Flash where the Parity will be calculated. The address of this pointer will be used for the address portion of the Parity calculation routine.
pu16ParityData	UINT16 *	Pointer to buffer where Parity data will be stored
u32Length	UINT32	Number of 32-bit words for which Parity will be calculated. The value of length is required to be multiples of 4.

Return Value

This function returns an BOOL value of either TRUE for a valid operation or FALSE due to:

1. Fails if “u32Length” is not a multiple of 4

Description

This function calculates Parity for a device. The user must provide a data buffer where the newly calculated Parity bits will be stored. This data can then be programmed in Flash using the usual programming method. The number of elements in the "pu16ParityData" buffer must be equal to the number of 32-bit words of data, "u32Length" for which Parity will be calculated.

3.30.5 Flash_Compact_Status_U16()- Deprecated

[retrieve current FSM status]

```
UINT16 Flash_Compact_Status_U16(FLASH_ARRAY_ST cntl);
```

Parameters

Parameter	Type	Purpose
cntl	FLASH_ARRAY_ST	Flash Control Base address of module whose status is being checked.

Return Value

This function returns a 16-bit value that defines the status of the program and erase state machine. For more information, see *Flash_Status_U16()*.

Description

This function is used to check the status of a compaction operation started by Flash_Start_Compact_B or Flash_Start_Command_B. These functions allow the user to perform some other tasks, such as feeding a watchdog, while a sector is being compacted. No attempts should be made to read from Flash locations in the same bank as the sector being compacted until the compaction completes (indicated by BUSY going back to 0). Once the BUSY bit is low, indicating command completion, it is recommended to flag a failure if any other status bits read as anything other than 0.

3.30.6 Flash_EngRow_V()- Deprecated

[read engineering data from OTP]

```
void Flash_EngRow_V(UINT32 *start,
                    FLASH_ARRAY_ST cntl,
                    FLASH_STATUS_ST *status
                    ):
```

Parameters

Parameter	Type	Purpose
start	UINT32 *	Pointer to first word in Flash array (first bank, first sector)
cntl	FLASH_ARRAY_ST	Flash Control Base address of module where Flash region resides
status	FLASH_STATUS_ST *	Pointer to a structure of the type FLASH_STATUS_ST. The function puts statistical information in this structure.

Return Value

The following values are stored in the FLASH_STATUS_ST structure:

stat1	Engineering Row Word #1
stat2	Engineering Row Word #2
stat3	Engineering Row Word #3
stat4	Engineering Row Word #4

Description

This function is called by Flash_EngInfo_V to read the engineering information from the TI OTP. Because of the way the data is arranged, there is not functional difference between this function and Flash_EngInfo_V.

Note

The data obtained by this function can be read the following way: the four words are in Binary Coded Decimal (BCD) notation (except FlowCheck, which is binary). Unused means that the value of these bits is indeterminate.

Table 2. Stat Return Value Reference

	8 Bits	8 Bits
stat1	ASIC ID	
stat2	Lot Number	
stat3	FlowCheck	Wafer Number
stat4	X Coordinate	Y Coordinate

3.30.7 Flash_Erase_Status_U16()- Deprecated

[returns status of FSM during erase]

```
UINT16 Flash_Erase_Status_U16(FLASH_ARRAY_ST cntl);
```

Parameters

Parameter	Type	Purpose
cntl	FLASH_ARRAY_ST	Flash Control Base address of module whose status is being checked

Return Value

This function returns a 16 bit value that defines the status of the program and erase state machine. For more information, see *Flash_Status_U16()*.

Description

This function is used to check the status of a erase operation started by Flash_Start_Erase_B or Flash_Start_Command_B. These functions allow the user to perform some other tasks, such as feeding a watchdog, while a sector is being erased. No attempts should be made to read from Flash locations in the same bank as the sector being erased until the erase completes (indicated by BUSY going back to 0). Once the BUSY bit is low, indicating command completion, it is recommended to flag a failure if any other status bits read as anything other than 0.

3.30.8 Flash_Match_Key_B()- Deprecated

```
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 fourth word from the end of the first 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 where Flash sector resides

Return Value

This function always returns TRUE.

Description

This function is a dummy function that exists merely for the purpose of portability with F05 and F10 applications and is, therefore, deprecated, because protection keys are not implemented in the F035 Flash wrapper. This function merely returns TRUE.

3.30.9 Flash_Match_Key_V()- Deprecated

```
BOOL Flash_Match_Key_V(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 fourth word from the end of the first 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 where Flash sector resides

Description

This function exists merely for the purpose of portability from F05 and F10 applications and is deprecated, because protection keys do not exist in the F035 architecture. This function merely returns immediately.

3.30.10 Flash_PSA_Vt_Verify_B()- Deprecated

[Vt fast verify using PSA]

```
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
                           );
```

Parameters

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in Flash to be verified in Vt mode
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
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document.
cntl	FLASH_ARRAY_ST	Flash Control Base address of module where Flash region resides
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information

NOTE: This function must be executed from RAM in non-pipeline mode.

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 verifies proper programming by using Vt mode and an external voltage provided on the TEST1 device pin by generating a 32 bit PSA checksum for the data region. Verification starts from the start address “start” and checks “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”.

Warning

The result of the read is dependent on the voltage threshold of the bits in the address range being read and the voltage that is being applied externally on the TEST1 pin. This function also requires that the NTRST device pin be held at logic '1' and that your hardware supports supplying a DC voltage between 0.0 V and 8.0 V to the TEST1 pin after powering up the DUT but before calling the function. Before powering down the device, the user must take care to first power down the TEST1 pin to 0.0 V, otherwise damage to the device will likely occur. This function is not recommended for use in a production environment.

3.30.11 Flash_Set_Vread_V()- Deprecated

```
void Flash_Set_Vread_V(FLASH_ARRAY_ST cntl);
```

Parameters

Parameter	Type	Purpose
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module

NOTE: This function must be executed from RAM in non-pipeline mode.

Description

This function is used to set the read voltage to 4.914 V. The default word-line voltage during read (VREAD) is 4.914 V after reset (FVREADCT[3:0]=0xF). This function is provided merely for backwards compatibility to F05 applications, but is deprecated, because VREAD is governed primarily through the settings programmed to the TI OTP sector as set by the setup_state_machine function.

The VREAD voltage is controlled by the Flash control register bits FVREADCT[15:12]. The voltage is set according to Table 3.

Table 3. FVREAD Value and Voltage

FVREADCT[3:0]	VREAD Voltage	FVREADCT[3:0]	VREAD Voltage
0x0	2.50V	0x8	4.333V
0x1	2.75V	0x9	4.416V
0x2	3.00V	0xA	4.499V
0x3	3.25V	0xB	4.582V
0x4	3.50V	0xC	4.665V
0x5	3.75V	0xD	4.748V
0x6	4.00V	0xE	4.831V
0x7	4.25V	**0xF (default)**	**4.914V**

3.30.12 Flash_Start_Command_B()- Deprecated

[issue command to FSM]

```
BOOL Flash_Start_Command_B(UINT32 *pu32Start,
                           FLASH_CORE oFlashCore,
                           UINT32 u32Delay,
                           FLASH_ARRAY_ST oFlashControl,
                           UINT16 u16Command,
                           UINT32 u32Data
                           );
```

Parameters

Parameter	Type	Purpose
pu32Start	UINT32 *	Points to the first word in the Flash sector that is to be compacted
oFlashCore	FLASH_CORE	Bank select (0-7) of sector being compacted
u32Delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
oFlashControl	FLASH_ARRAY_ST	Flash Control Base address of module where Flash sector resides.
u16Command	UINT16	Command to be sent to the F035 State machine
u32Data	UINT32	Command appropriate data to send to the F035 state machine after the command is issued

Return Value

This function returns a Boolean value. Pass = 1, Fail = 0. A failure indicates that the delay value was too large (see Section 4).

Description

This function is used to issue a command to the Flash State Machine. The function *Flash_Status_U16()* can be used to determine when the command has completed. This function allows the user to perform some other tasks while the state machine is performing the command such as feeding a watchdog or servicing the peripherals. No attempts should be made to read from Flash locations in the same bank as the area being operated on by the Flash State Machine until the command completes. This function is deprecated and it is recommended to use *Flash_Start_Async_Command_B()* instead.

3.30.13 Flash_Vt_Bit_Count()- Deprecated

[Counts number of erased bits in Vt mode]

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

Parameters

Parameter	Type	Purpose
start	UINT32 *	Points to the first word to be read
buff	UINT32 *	Pointer to data buffer that will store the number of “1's” seen during the Vt Read
length	UINT32	Number of 32-bit words to be read
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
core	FLASH_CORE	Bank select (0-7) of bank within which region to be read resides
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module

NOTE: This function must be executed from RAM in non-pipeline mode.

Return Value

This function does not return any value. The number of set bits seen during the Vt mode read for the range of Flash specified is returned in data locations referenced by the buff pointer. Vt mode is a special test mode that allows the word line voltage to the Flash to be supplied externally through the TEST1 device pin.

Description

This function will read data stored in the Flash in Vt mode and count the number of set bits, “1's”, seen store the count in location pointed to by buff.

Warning

The result of the read is dependent on the voltage threshold of the bits in the address range being read and the voltage that is being applied externally on the TEST1 pin. This function also requires that the NTRST device pin be held at logic '1' and that your hardware supports supplying a DC voltage between 0.0 V and 8.0 V to the TEST1 pin after powering up the DUT but before calling the function. Before powering down the device, the user must take care to first power down the TEST1 pin to 0.0 V, otherwise, damage to the device will likely occur. This function is not recommended for use in a production environment.

3.30.14 Flash_Vt_Blank_B()- Deprecated

[Vt Verify Flash region is blank]

```
BOOL Flash_Vt_Blank_B(UINT32 *start,
                      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 Flash that will be read
length	UINT32	Number of 32-bit words to be read
core	FLASH_CORE	Bank select (0-7) of bank within region to be read resides
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document.
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information. Note that word length must be passed to the Flash_Blank_B function via the status.stat1 element: **status.stat1=128 => No Parity and ECC** **status.stat1=132 => Add Parity bits** **status.stat1=144 => Add ECC bits**

NOTE: This function must be executed from RAM in non-pipeline mode.

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 blank check (based on voltage on TEST1 pin)
stat2	Data at first location failing blank check (based on voltage on TEST1 pin)
stat3	Data expected at failing address (0xFFFFFFFF)
stat4	Total number of failing bits found in region

Vt mode is a special test mode that allows the word line voltage to the Flash to be supplied externally through the TEST1 device pin.

Description

This function verifies that the Flash has been properly erased by using Vt mode and an external voltage provided on the TEST1 device pin 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_Vt_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. The user must also specify via the status.stat1 parameter whether to also blank check the corresponding parity and ECC bits for the given main Flash address range (for more details, see the above Parameter table).

Warning

The result of the read is dependent on the voltage threshold of the bits in the address range being read and the voltage that is being applied externally on the TEST1 pin. This function also requires that the NTRST device pin be held at logic '1' and that your hardware supports supplying a DC voltage between 0.0 V and 8.0 V to the TEST1 pin after powering up the DUT but before calling the function. Before powering down the device, the user must take care to first power down the TEST1 pin to 0.0 V, otherwise, damage to the device will likely occur. This function is not recommended for use in a production environment.

3.30.15 Flash_Vt_Read_V()- Deprecated

[Vt read Flash data to buffer]

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

Parameters

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in Flash that will be read
buff	UINT32 *	Pointer to buffer in RAM where the read data will be copied
length	UINT32	Number of 32-bit words to be read
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
core	FLASH_CORE	Bank select (0-7) of bank within which region to be read resides
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module

NOTE: This function must be executed from RAM in non-pipeline mode.

Return Value

This function returns an FAPI_ERROR_CODE, which reflects the status of the *setup_state_machine()*. The value of "length" words starting at address "buffer" will be written with data from Flash in Vt mode. Vt mode is a special test mode that allows the word line voltage to the Flash to be supplied externally through the TEST1 device pin.

Description

This function reads data stored in the Flash in Vt mode and copies it to successive locations pointed to by "buffer".

Warning

The result of the read is dependent on the voltage threshold of the bits in the address range being read and the voltage that is being applied externally on the TEST1 pin. This function also requires that the NTRST device pin be held at logic '1' and that your hardware supports supplying a DC voltage between 0.0 V and 8.0 V to the TEST1 pin after powering up the DUT but before calling the function. Before powering down the device, the user must take care to first power down the TEST1 pin to 0.0 V, otherwise, damage to the device will likely occur. This function is not recommended for use in a production environment.

3.30.16 Flash_Vt_Verify_B()- Deprecated

[Vt Verify Flash region against data buffer]

```
BOOL Flash_Vt_Verify_B(UINT32 *start,
                       UINT32 *buff,
                       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 Flash that will be read
buff	UINT32 *	Pointer to data buffer against which to verify data in Flash in Vt mode
length	UINT32	Number of 32-bit words to be read
core	FLASH_CORE	Bank select (0-7) of bank within region to be read
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information.

NOTE: This function must be executed from RAM in non-pipeline mode.

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 (based on voltage on TEST1 pin)
stat2	Data at first location failing verify (based on voltage on TEST1 pin)
stat3	Data expected at failing address
stat4	Total number of failing bits found in region

Vt mode is a special test mode that allows the word line voltage to the Flash to be supplied externally through the TEST1 device pin.

Description

This function verifies a region of Flash against a data buffer pointed to by "buff" by using Vt mode and an external voltage provided on the TEST1 device pin starting from the address passed in the parameter “start”. “length” words are read, starting at the starting address. The area specified for verify must be within a single bank specified by “core”. If the array contains multiple banks, Flash_Vt_Verify_B must be called separately for each bank to be verified. Regions may cross sector boundaries, as long as the sectors all reside in the same bank.

Warning

The result of the read is dependent on the voltage threshold of the bits in the address range being read and the voltage that is being applied externally on the TEST1 pin. This function also requires that the NTRST device pin be held at logic '1' and that your hardware supports supplying a DC voltage between 0.0 V and 8.0 V to the TEST1 pin after powering up the DUT but before calling the function. Before powering down the device, the user must take care to first power down the TEST1 pin to 0.0 V, otherwise, damage to the device will likely occur. This function is not recommended for use in a production environment.

3.30.17 Flash_Vt_Verify_Data_B()- Deprecated

[Vt verify against 'cyclical' buffer]

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

Parameters

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in Flash that will be read
buff	UINT32 *	Pointer to data buffer against which to verify data in Flash in Vt mode
length	UINT32	Number of 32-bit words to be read
core	FLASH_CORE	Bank select (0-7) of bank within region to be read
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information.
buflen	UINT32	Length of cyclical data buffer in 32-bit words

NOTE: This function must be executed from RAM in non-pipeline mode.

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 (based on voltage on TEST1 pin)
stat2	Data at first location failing verify (based on voltage on TEST1 pin)
stat3	Data expected at failing address
stat4	Total number of failing bits found in region

Vt mode is a special test mode that allows the word line voltage to the Flash to be supplied externally through the TEST1 device pin.

Description

This function verifies a region of Flash against a cyclical data buffer pointed to by "buff" by using Vt mode and an external voltage provided on the TEST1 device pin starting from the address passed in the parameter “start”. “length” words are read, starting at the starting address. The area specified for verify must be within a single bank specified by “core”. If the array contains multiple banks, Flash_Vt_Verify_Data_B must be called separately for each bank to be verified. Regions may cross sector boundaries, as long as the sectors all reside in the same bank.

Warning

The result of the read is dependent on the voltage threshold of the bits in the address range being read and the voltage that is being applied externally on the TEST1 pin. This function also requires that the NTRST device pin be held at logic '1' and that your hardware supports supplying a DC voltage between 0.0 V and 8.0 V to the TEST1 pin after powering up the DUT but before calling the function. Before powering down the device, the user must take care to first power down the TEST1 pin to 0.0 V, otherwise, damage to the device will likely occur. This function is not recommended for use in a production environment.

3.30.18 Flash_Vt_Zeros_B()- Deprecated

[Vt verify region contains all 0's]

```
BOOL Flash_Vt_Zeros_B(UINT32 *start,
                      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 Flash that will be read for all 0x00000000
length	UINT32	Number of 32-bit words to be read
core	FLASH_CORE	Bank select (0-7) of bank within region to be read
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module
status	FLASH_STATUS_ST *	Pointer to status structure for storing statistical information

NOTE: This function must be executed from RAM in non-pipeline mode.

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 (based on voltage on TEST1 pin)
stat2	Data at first location failing verify (based on voltage on TEST1 pin)
stat3	Data expected at failing address (0x00000000)
stat4	Total number of failing bits found in region

Vt mode is a special test mode that allows the word line voltage to the Flash to be supplied externally through the TEST1 device pin.

Description

This function verifies a region of Flash contains all 0x00000000 by using Vt mode and an external voltage provided on the TEST1 device pin starting from the address passed in the parameter “start”. “length” words are read, starting at the starting address. The area specified for Vt read must be within a single bank specified by “core”. If the array contains multiple banks, Flash_Vt_Zeros_B must be called separately for each bank to be verified. Regions may cross sector boundaries, as long as the sectors all reside in the same bank.

Warning

The result of the read is dependent on the voltage threshold of the bits in the address range being read and the voltage that is being applied externally on the TEST1 pin. This function also requires that the NTRST device pin be held at logic '1' and that your hardware supports supplying a DC voltage between 0.0 V and 8.0 V to the TEST1 pin after powering up the DUT but before calling the function. Before powering down the device, the user must take care to first power down the TEST1 pin to 0.0 V, otherwise, damage to the device will likely occur. This function is not recommended for use in a production environment.

3.30.19 OTP_Blank_B()- Deprecated

```
BOOL OTP_Blank_B(UINT32 *start,
                 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 customer OTP sector that will be blank-checked
length	UINT32	Number of 32-bit words to be blank-checked
core	FLASH_CORE	Bank select (0-7) of bank within which customer OTP sector resides
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module
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 the first non-blank location
stat2	Data read at the first non-blank location
stat3	Expected data (0xFFFFFFFF)
stat4	Read mode value in REGOPT register when non-blank location read. 0xA is value for read margin 1 mode, 0x2 is value for normal read mode.

Description

This function verifies that the OTP section is still blank. The area specified for blank check must be within the OTP sector of the bank specified by “core” and the control register should be passed in as “cntl”. This function is simply a wrapper around Flash_Blank_B, so it should be considered deprecated in favor of simply calling Flash_Blank_B.

3.30.20 OTP_PSA_Verify_B()- Deprecated

```
BOOL OTP_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 customer OTP sector that 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 bank within which the customer OTP sector resides
cntl	FLASH_ARRAY_ST	Flash Control Base address of module where customer OTP 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 (only on failure):

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

Description

This function verifies proper programming by using normal read, read-margin 0 and read-margin 1 modes, but will do so through the Parallel Signature Analysis (PSA) module. Verification starts from the start address Flash_Start_PU32 and checks Flash_Length_U32 words from the start address. The area specified for PSA verify must be within the OTP sector of the bank specified by “core” and the control register should be passed in as “cntl”. This function is simply a wrapper around Flash_PSA_Verify_B, so it should be considered deprecated in favor of simply calling Flash_PSA_Verify_B.

3.30.21 OTP_Read_V()- Deprecated

```
void OTP_Read_V(UINT32 *start,
                UINT32 *buffer,
                UINT32 length,
                UINT32 core,
                FLASH_ARRAY_ST *cntl
                );
```

Parameters

Parameter	Type	Purpose
start	UINT32 *	Points to the first word in the customer OTP sector that will be read
buffer	UINT32 *	Pointer to buffer in RAM where the read data will be copied
length	UINT32	Number of 32-bit words to be read
core	FLASH_CORE	Bank select (0-7) of bank within which customer OTP sector resides
cntl	FLASH_ARRAY_ST	Flash Control Base address of Flash module

Return Value

This function does not return any value, but "length" words starting at address "buffer" will be written with data from customer OTP sector.

Description

This function reads data stored in the customer OTP sector of the Flash and copies it to successive locations pointed to by "buffer". This function is simply a wrapper around Flash_Read_V, so it should be considered deprecated in favor of simply calling Flash_Read_V.

3.30.22 OTP_Verify_B()- Deprecated

```
BOOL OTP_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 customer OTP sector 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 bank where the customer OTP sector resides
cntl	FLASH_ARRAY_ST	Flash Control Base address of module where customer OTP 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 (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 0x2 means normal read failure, 0x6 means read margin 0 failure, and 0xA means read margin 1 failure

Description

This function verifies proper programming by using normal read, read-margin 0 and read-margin 1 modes. Verification starts from the start address “start” and checks “length” words from the start address.

The area specified for verify must be within the OTP sector of the bank specified by “core” and the control register should be passed in “cntl”. The verify routine compares the data stored in the Flash to the data stored in the buffer pointed to by “buffer”. This function is simply a wrapper aroung Flash_Verify_B, so this function should be considered deprecated in favor of simply calling Flash_Verify_B directly.

3.30.23 verify_read()- Deprecated

[verify until a mismatch is found]

```
UINT16 verify_read(volatile UINT16 *addr,
                   UINT32 *length,
                   FLASH_ARRAY_ST cntl,
                   UINT32 delay,
                   FLASH_TIMING_ST *timing
                   );
```

Parameters

Parameter	Type	Purpose
addr	volatile UINT16 **	Pointer to address pointer, which is the current starting address to begin verifying. This address pointer is incremented for every address that compares correctly, and when a mismatch occurs, the address pointer now points to the address of the failing location.
length	UINT32 *	Pointer to a length value in bytes that is decremented for every matching address. On mismatch, the length being pointed to is the remaining bytes before the end of the buffer, starting at the last failing address.
cntl	FLASH_ARRAY_ST *	Flash Control Base address of module where the Flash State Machine resides
delay	UINT32	From Table 4, Flash Delay Parameter Values in Section 4 of this document
timing	FLASH_TIMING_ST *	Timing structure containing the timing for the given verify mode and also the 16-bit data value against which to compare the 16-bit values being verified.

NOTE: This function must be executed from RAM in non-pipeline mode.

Return Value

This function returns the 16-bit failing data at the first failing address, and it also updates the address pointed to by 'addr' and the length pointed to by 'length' so that the following verify can pick up where it left off.

Description

This function is used to verify a range of data of '*length' bytes compared against a 16 bit value specified in the FLASH_TIMING_ST structure. If any location does not match the compare data, the function returns the failing data and also makes sure the address pointer pointed to by 'addr' is updated to the address of the failing location, and that the length pointed to by 'length' is updated to the reflect the remaining bytes before the end of the region.

4 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. This value is used to scale timings based HCLK and the maximum allowed FCLK as configured in the OTP via scaling RWAIT in the FDRCNTL register or calculating appropriate values to pass to the wait_delay function.

The generic calculation for the delay parameter is as follows:

delay = (ceiling)(HCLK (in Mhz) / 2)

For example:

Table 4. Example Flash Delay Parameter Values

HCLK frequency (in MHz)	Delay value
HCLK ≤ 2	1
2 ≤ HCLK ≤ 4	2
4 ≤ HCLK ≤ 6	3
6 ≤ HCLK ≤ 8	4
8 ≤ HCLK ≤ 10	5
10 ≤ HCLK ≤ 12	6
12 < HCLK ≤ 14	7
14 < HCLK ≤ 16	8
16 < HCLK ≤ 18	9
18 < HCLK ≤ 20	10
20 < HCLK ≤ 22	11
22 < HCLK ≤ 24	12
24 < HCLK ≤ 26	13
26 < HCLK ≤ 28	14
28 < HCLK ≤ 30	15
30 < HCLK ≤ 32	16
32 < HCLK ≤ 34	17
34 < HCLK ≤ 36	18
36 < HCLK ≤ 38	20
...	...
158 < HCLK ≤ 160	80
...	...
178 < HCLK ≤ 180	90
...	...

5 Recommended Flow Guidelines

5.1 New Devices from Factory

Devices are shipped erased from the Factory. It is recommended, but not required to do a blank check on devices received to verify they are erased.

5.2 Using Flash_Erase_B()

Figure 1 describes the flow for erasing a single sector on a device when using the Flash_Erase_B function to accomplish the erase.

The user should familiarize themselves with the Flash_Erase_B function description in Section 3.10.

To save erase time, Flash_Erase_B supports disabling preconditioning (program to 0's prior to applying erase pulses) on sectors that return TRUE from the Flash_Blank_B function. This preconditioning disable is accomplished via a special key (0x12345678) passed via the FLASH_STATUS_ST status.stat1 element (for more information, see Flash_Erase_B desciption in Section 3.10) to the Flash_Erase_B function.

While this feature does enable potentially faster throughput for erase on blank sectors, it does require the user to take extra care to make sure no depleted bits exist in the target sector prior to calling Flash_Erase_B with preconditioning disabled. This is the reason for the Flash_Compact_B call prior to the Flash_Blank_B call in Figure 1. A device shipped from TI should have no depleted columns, but a device that may have been reset or power cycled during a previous erase operation may contain depleted columns or marginally erased bits. The flow shown in Figure 1 is designed to repair any such bits if allowed to run to completion on a given sector.

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.

Flash_Erase_B also allows for collecting erase pulse statistics for the sector if the user is interested.

Figure 1. Using *Flash_Erase_B()* With Optional Preconditioning Disabled Flow

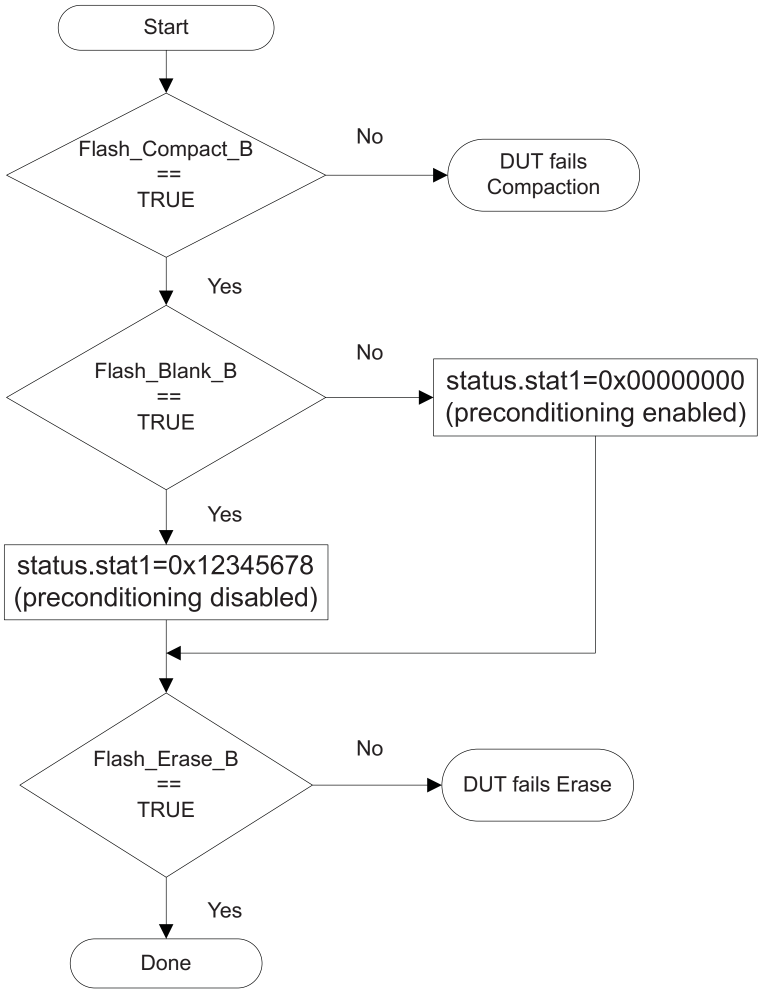
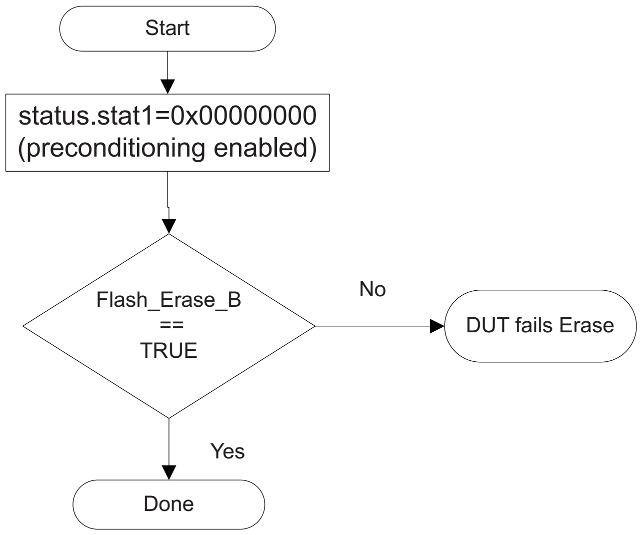


Figure 2. Using *Flash_Erase_B()* Simple Flow



5.3 *Using Flash_Erase_Bank_B()*

Figure 3 describes the flow for erasing a single bank on a device when using the Flash_Erase_Bank_B function to accomplish the erase.

The user should familiarize themselves with the Flash_Erase_Bank_B function description in Section 3.11.

To save erase time, Flash_Erase_Bank_B supports disabling preconditioning (program to 0's prior to applying erase pulses) on banks that return TRUE from the Flash_Blank_B function. This preconditioning disable is accomplished via a special key (0x12345678) passed via the FLASH_STATUS_ST status.stat1 element (for more information, see the *Flash_Erase_Bank_B* desciption in Section 3.11) to the Flash_Erase_Bank_B function.

While this feature does enable potentially faster throughput for erase on blank banks, it does require the user to take extra care to make sure no depleted bits exist in the target sector prior to calling Flash_Erase_Bank_B with preconditioning disabled. This is the reason for the Flash_COD_Compact_B call prior to the Flash_Blank_B call in Figure 3. A device shipped from TI should have no depleted columns, but a device that may have been reset or power cycled during a previous erase operation may contain depleted columns or marginally erased bits. The flow shown in Figure 3 is designed to repair any such bits if allowed to run to completion on a given sector.

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_Bank_B.

Flash_Erase_Bank_B also allows for collecting erase pulse statistics for the sector if the user is interested.

Figure 3. Using *Flash_Erase_Bank_B()* With Optional Preconditioning Disabled Flow

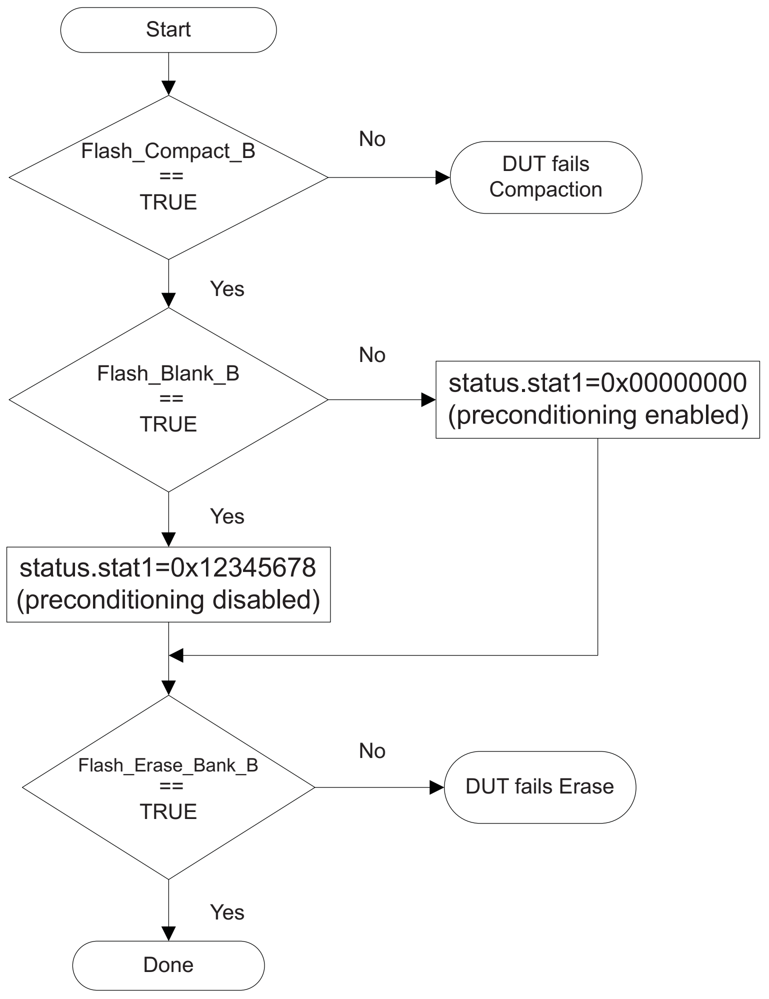
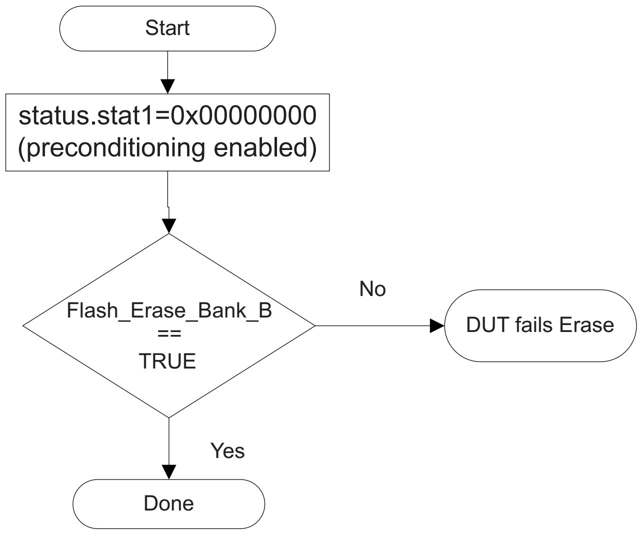

Start
Flash_Compact_B == TRUE
No
DUT fails Compaction
Yes
Flash_Blank_B == TRUE
No
status.stat1=0x00000000 (preconditioning enabled)
Yes
status.stat1=0x12345678 (preconditioning disabled)
Flash_Erase_Bank_B == TRUE
No
DUT fails Erase
Yes
Done

Figure 4. Using *Flash_Erase_Bank_B()* Simple Flow

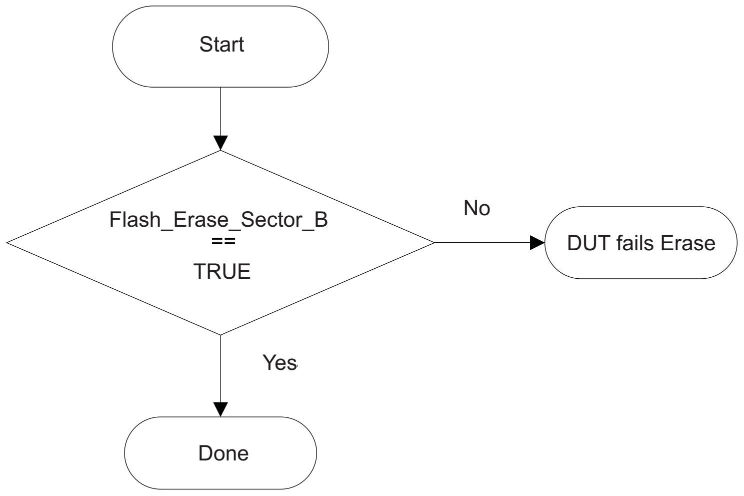


5.4 Using Flash_Erase_Sector_B()

Figure 5 describes the flow for erasing a single bank on a device when using the Flash_Erase_Sector_B function to accomplish the erase.

The user should familiarize themselves with the Flash_Erase_Sector_B function description in Section 3.12.

Figure 5. Using Flash_Erase_Sector_B() Flow

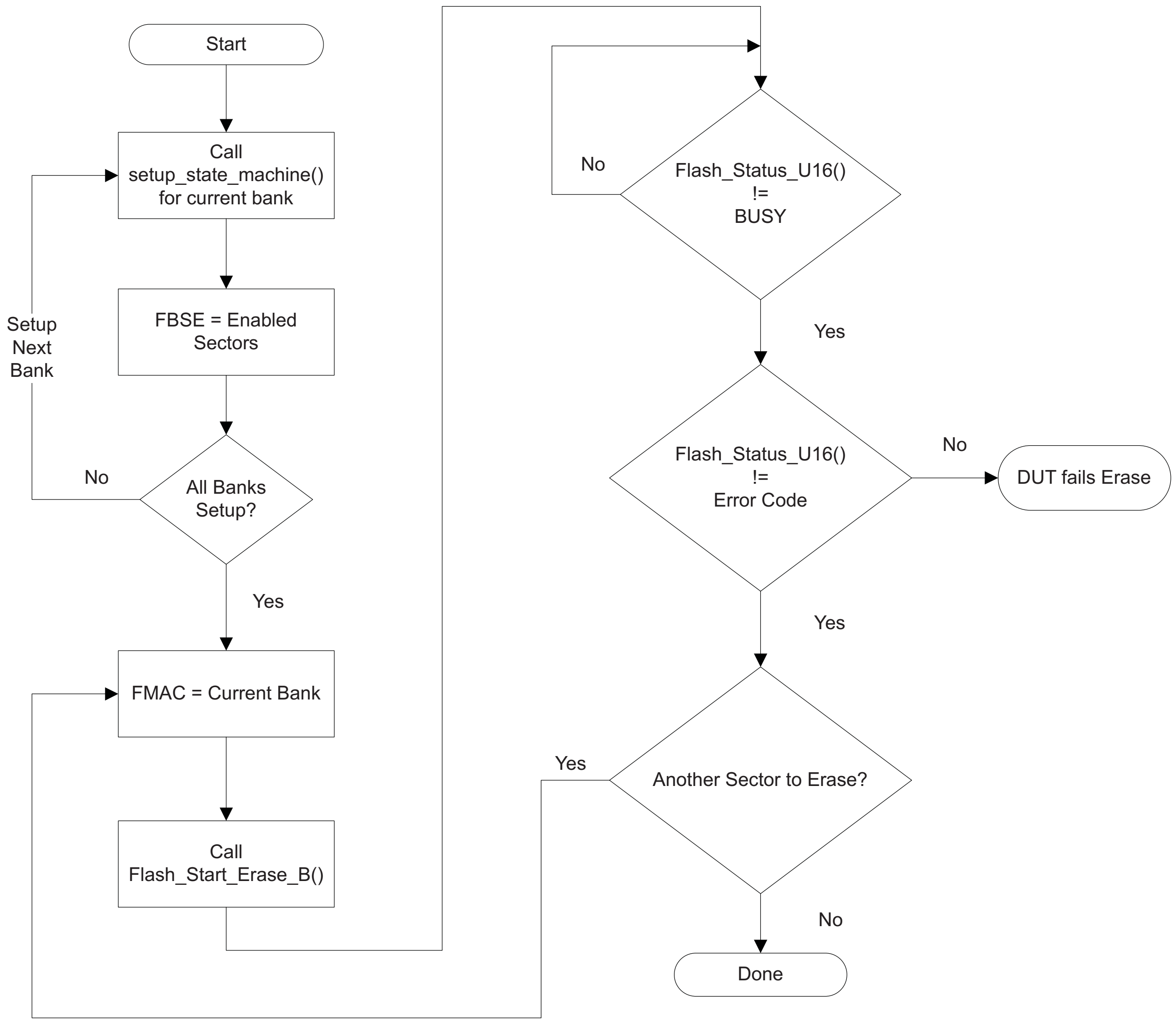


5.5 Using Flash_Start_Erase_B()

Figure 6 describes the flow for erasing a sector or sectors on a device when using the Flash_Start_Erase_B function to accomplish the erase.

The user should familiarize themselves with the Flash_Start_Erase_B function description in Section 3.20.

Figure 6. Using *Flash_Start_Erase_B()* Flow



5.6 Recommended Programming Flow

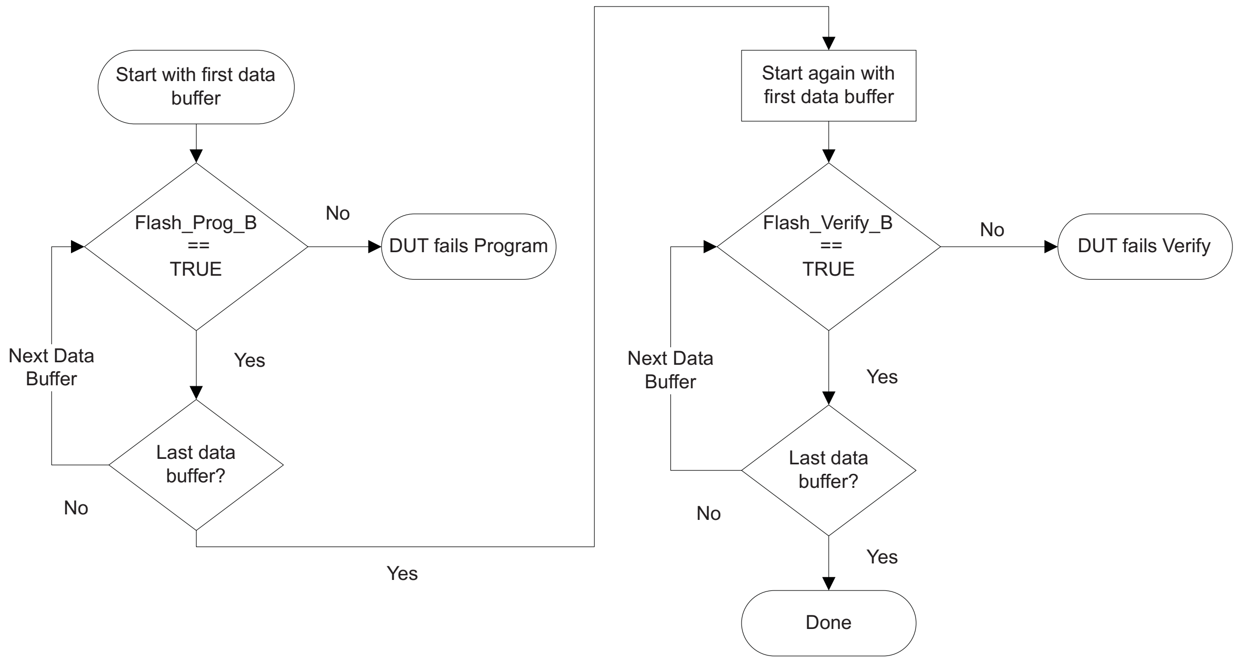
The flow in Figure 7 assumes the user has already erased all affected sectors using one of the previously described erase flow(s) (see Section 5.2 - Section 5.5). The user needs to manage the data buffers being programmed to Flash such that they do not cross boundaries between Flash banks. Also note that the user must take care to make sure data buffers are 32-bit aligned.

For example, if a user has 1KB of data to write starting at the last 768 bytes of bank 0 on a device with more than 1 bank, you 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.

In another example, if the user has 3 bytes to program to the device, the user should read from the 32-bit location, to which the data will be programmed, to appropriately fill in the 4th byte in the data buffer.

TI recommends programming all data buffers using Flash_Prog_B, then performing verify on all buffers using either Flash_Verify_B or Flash_PSA_Verify_B. It is NOT recommended to program a buffer, verify the buffer, then program the next buffer, verify, and so forth. The reason for this is the very small risk that a device damaged in handling may contain a Flash bit susceptible to charge loss from bit-bit stresses introduced during programming. If the bit that drags the damaged bit down is in a separate data buffer, then the damaged bit may pass verify after its initial programming, but programming subsequent buffers may cause the damaged bit to fail a subsequent verify. Note that bit damage is not expected in any part shipped from TI, but since the possibility of damage does exist in any subsequent manufacturing process, programming all data before verifying all data will more likely catch any damaged bits at the program and verify stage rather than in the field.

Figure 7. Recommended Programming Flow



Appendix A Revision History

Table 5 lists the API versions.

Table 5. API Version History

Version	Additions, Modifications and Deletions
< 1.05	For revisions prior to v1.05, please see the v1.04 documentation
1.05	• Corrected compilation issue with *Fapi_HardwareCalculateECC()* • Corrected accumulated pulse count information returned by *Flash_Prog_Data_B()*
1.06	• Resolved Advisory #SDOCM00084916. For additional details , see SPNZ185.
1.07	• Removed deprecated function *exec_pulses()* • The following functions are not recommended for use in production code and have been deprecated: – *Flash_PSA_Vt_Verify()* – *Flash_Vt_Bit_Count()* – *Flash_Vt_Blank_B()* – *Flash_Vt_Read_V()* – *Flash_Vt_Verify_Data_B()* – *Flash_Vt_Zeros_B()* – *verify_read()* • Resolved Advisory #SDOCM00104972. For additional details, see SPNZ185.
1.08	Corrected error with blank check at the end of the flash
1.09	• Resolved Advisory #SDOCM00105927. For additional details , see SPNZ185.

Table 6 lists the API changes made since the previous revision of this document.

Table 6. Document Revision History

Reference	Additions, Modifications and Deletions
-	Initial revision
A	Updated for v1.04
B	Updated for v1.05
C	Updated for v1.06
D	Updated for v1.08
E	Updated for v1.09

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		