Porting Intel Wireless Flash (L18) Memory with Synchronous PSRAM for Locosto

ABSTRACT

Intel is moving to a new architecture which will require modifications to both the Flash Driver as well as Flash File System (FFS). This is the only Flash available from Intel in 256/64 configuration which will be required by several future cellular products. This app note explains a little about this new Flash and describes how to port this new flash on to software developed for Locosto processor based products. Note that in this document we have covered software aspects for the FFS porting.

Contents

1 Introduction 3
1.1 Porting in steps 3
1.2 An Introduction to the FLASH Memory 4
2 Locosto FFS 6
2.1 FFS Architecture 6
2.1.1 API Layer 6
2.1.2 Core Layer 7
2.1.3 Driver Layer 8
3 Locosto FFS Driver interface 11
3.1 Locosto FFS driver Data Structures 11
3.1.1 ffsdrv_s 11
3.1.2 flash_info_s 11
3.1.3 block_info_s 12
3.1.4 FFS_DRIVER 13
3.1.5 FFS_MANUFACTURER 13
3.1.6 flash_base_addr 14
3.1.7 #define directives used 14
4 Porting to "Intel Wireless Flash (L18) with PSRAM" FLASH Device 15
4.1 Difference in data sheets.(Intel 'L30' & 'L18 with PSRAM') 15
4.1.1 Based on Memory Map 15
4.1.2 Based on RAM interface 15
4.2 Data Structure & defines need to modify/create: 17

4.2.1 flash_info (refer section 2.1.2) 17
4.2.2 ffsdrv_s ffsdrv_intel_L18_PSRAM (refer section 2.1.1) 18
4.2.3 # defines directives need to add with hardware 18
4.3 Functions need to modify/Create : 18
References 19

Figures

Figure 1. The FFS architecture is split in three layers **..... 6**
Figure 2. FFS reading vs. modify functions **..... 7**
Figure 3. Single-bank flash **..... 8**
Figure 4. Multi-bank flash **..... 9**
Figure 5. Typical configurations of driver **..... 10**
Figure 6. PSRAM loading configurations registers using software access (Taken from Intel® Wireless Flash Memory (L18) with Synchronous PSRAM data sheet). **..... 17**
Figure 7. PSRAM reading registers using software access (Taken from Intel® Wireless Flash Memory (L18) with Synchronous PSRAM data sheet). **..... 17**

Tables

Table 1. Presently Supported Different NOR Flash devices. **..... 3**
Table 2. Presently Supported Intel Flash devices. **..... 3**

1 Introduction

The purpose of this App-Note is to identify porting requirement of the existing Flash driver to support Intel Wireless Flash (L18) memory with synchronous PSRAM. The scope of this document is to describe changes required for porting new hardware on to Locosto hardware with minimum changes and adhere to software architecture.

The following tables list the supported flash parts.

Table 1. Presently Supported Different NOR Flash devices.

Device	Characteristic
AMD Single bit	Fast byte write.
AMD mirror bit	2 bits per flash cell. Fast byte write. Re-Programming possible. Fast page program.
Intel single bit	Fast byte write.
Intel Strata (multi-level)	2 bits per flash cell. Re-Programming after Power Loss not possible. Slow byte write. Fast page program
SST version 1*	Do not support erase suspend and resume. Fast erase.
SST version 2**	Support erase suspend and resume.

* Approx before 2003 ** Approx after 2003

Table 2. Presently Supported Intel Flash devices.

Device ID	**Characteristic**
0x88C2	Intel® Advanced Boot Block Flash Memory (C3) - 16-Mbit Top Boot Device.
0x88C3	Intel® Advanced Boot Block Flash Memory (C3) - 16-Mbit Bottom Boot Device.
0x88C4	Intel® Advanced Boot Block Flash Memory (C3) - 32-Mbit Top Boot Device.
0x88C5	Intel® Advanced Boot Block Flash Memory (C3) - 32-Mbit Bottom Boot Device.
0x88CC	Intel® Advanced Boot Block Flash Memory (C3) - 64-Mbit Top Boot Device.
0x88CD	Intel® Advanced Boot Block Flash Memory (C3) - 64-Mbit Bottom Boot Device.
0x8854	Intel® Wireless Flash Memory (W30) - 64-Mbit Top Boot Device.
0x8855	Intel® Wireless Flash Memory (W30) - 64-Mbit Bottom Boot Device.
0x8812	Intel Strata Flash® Wireless Memory (L30) – 128-Mbit Top Boot Device.

1.1 Porting in steps

Porting of new flash part, **Intel Wireless Flash® Memory (L18) with PSRAM** can be simplified into the following steps.

1. Understand the Flash File System architecture.
2. Understand the key features of the new flash part
3. Identify the nearest matching one from the existing list of supported flash parts and
4. Identify the changes needed to support the new flash part.

This document addresses the above mentioned steps in detail.

1.2 An Introduction to the FLASH Memory

Flash is a form of Electrically Erasable Read Only Memory. Flash chips are arranged into blocks/sectors. Some manufactures call them blocks and others call them sectors. The term used in this document is blocks. A typical flash device contains N number of 8kBytes blocks (so-called boot blocks) and N number of 64kBytes blocks. The boot blocks can be at the bottom or at the top of the flash thus the device is respectively called a bottom or top boot device. A flash block is a group of bits that share erase circuitry and all bits are erased simultaneously. Reading is performed like reading from any traditional memory such as RAM or ROM. The writing, erasing and other flash specific operations require specific sequences of operations. Some flash products come with two or more flash banks on chip, so-called multi-bank (MB) devices. A bank is a group of flash blocks.

Flash characteristic:

- Random read and write access per byte basis.
- High access time (typical 70ns).

Flash characteristic that is addressed as constraints:

- Relative slow write performance (typical 8 μs/word).
- Bits can be changed to zero by a write operation but can only be changed back to one by erasing a whole block.
- Resetting bits from zero to one cannot be done individually, but only by resetting (or 'erasing') a complete block.
- The lifetime of a flash chip is measured in erase cycles, with the typical lifetime being 100,000 erases per block.
- Flashes can only perform one operation at a time within a bank.
- Typical block erase duration is 1 sec (worst-case can get above 15 sec!).

In order to have higher density the manufactures have developed technologies that stores 2 bit per flash cell. The 2 technologies are called respectively "multi-level-cell" (MLC) and "mirror-bit". Each technology has special characteristics. The MLC technology is introduced by "Intel" where as the "mirror-bit" technology is introduced by "Spansion".

The "MLC" technology lowers the cost by enabling the storage of multiple bits of data per memory cell thereby reducing the consumption of silicon area. The 2 bit/cell Intel Strata Flash memory technology provides a cost structure equivalent to the next generation of process technology while using the current generation of process technology equipment.

"Mirror-Bit" flash technology is fundamentally different and more advanced than conventional multi-level cell (MLC) and single-level cell (SLC) floating gate technology. The Mirror-Bit cell doubles the

intrinsic density of a Flash memory array by storing two physically distinct bits on opposite sides of a memory cell. Each bit within a cell serves as a binary unit of data (e.g. either 1 or 0) that is mapped directly to the memory array.

TI-Flash driver is supporting following superior command sets.
1. M28 (Intel alike).
2. M29 (AMD alike).

The flash characteristic information given in the above section is generic. In case of Intel L30, which is the nearest match for L18, the worst case block erase timeout will be 4 sec.

2 Locosto FFS

This section gives an overview of the Flash File System Architecture in the Locosto Platforms.

2.1 FFS Architecture

The FFS architecture consists of three layers, as defined below:

1. **API layer**: This layer is used for file and directory level operations and is loosely modeled after the POSIX file I/O interface.
2. **Core layer**: This layer takes care of operations related to chunks and nodes management.
3. **Driver layer:** This layer covers all the flash related operations.

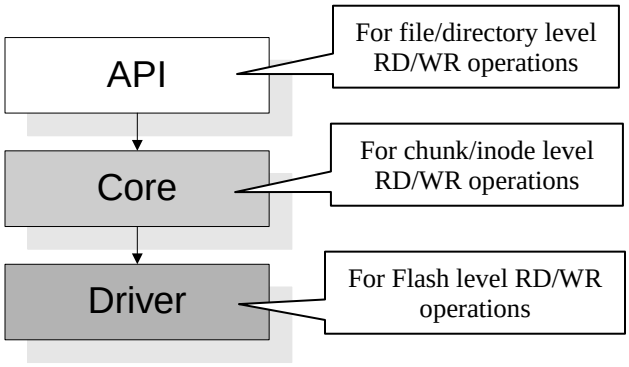

Figure 1. The FFS architecture is split in three layers

2.1.1 API Layer

The API's are not completely POSIX file I/O interface. Beside the POSIX inspired functions, the following functions exist:

- ffs_file_write() - added to simplify the creating of files.
- ffs_file_read() - added to simplify the reading of files.
- ffs_is_modifiable() - this function is to be implemented by the user. It is called by FFS when an application tries to modify an object, which has the read-only flag set. The user can use the input argument to determine if the object being modified should indeed be considered read-only.

Two types of FFS functions exist:

- The FFS reading functions are functions that do NOT change contents or state of FFS, such as ffs_fread(), ffs_stat() etc.
- The FFS modify functions or modify operations are contents/state changing functions that add,

change or remove object(s) or change state in FFS - e.g. ffs_fwrite(), ffs_mkdir(), ffs_remove() etc.

An FFS reading function executes in the user context, but FFS modify functions like write/erase is executed by FFS process/tasks. This is because writing/erasing flash memories is a time-consuming process, taking anywhere from some hundreds of microseconds up to several seconds.
There are two types of modify functions: blocking and non-blocking.

2.1.1.1 Non-blocking function

When the application calls a modify function the API function sends a mail to the FFS background task. The API function then returns and the caller's task continue to run. This means that the FFS modify operation has been scheduled and has not begun execution. Later, when the FFS task gets the CPU it will read the mail and execute the operation requested. When it is done, it will return the result of the operation to the caller by means of either a mail or a callback function as specified at the time of calling. All exceptions are returned through this callback mechanism.

2.1.1.2 Blocking function

The Blocking function works almost like the non-blocking function except that it does not use callback but automatic waits for the operation to complete before returning from the call thus blocks the calling task for the time it takes the FFS function to access the physical flash.

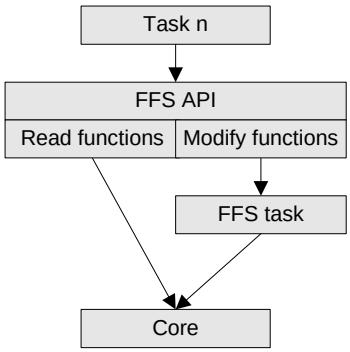

Figure 2. FFS reading vs. modify functions

2.1.2 Core Layer

Core layer takes care of managing flash sector using inodes and data chunks. This layer abstracts the flash driver interface to a structured file system interface so that the API layer can use Flash memory like files & directories. Each inode contains file-system meta-data for the chunk, such as the data chunk *location* (encoded as an offset), file size, flags, parent directory etc. Data chunks are used to contain the objects name and the actual data. Chunks are stored in the data blocks.

2.1.3 Driver Layer

The driver Layer abstracts the Flash Interface specific code from the rest of the FFS modules. Two superior command sets are used for the flash manufactures, M28 (Intel alike) and M29 (AMD alike). By supporting these two sets Locosto supports most of the NOR flash devices in the market.

The driver is responsible for identifying the flash device by Querying the manufacture ID and device ID. The ID's are later used to choose the right configuration. Besides identifying the flash devices the driver also needs support for both writing and erasing the flash device (no read driver is needed). As mentioned in section 2, there are two different NOR Flash architectures with different usage restrictions. To resolve these restrictions and take advantage of the MB flash, Locosto has driver for each type.

2.1.3.1 Single-bank (SB)

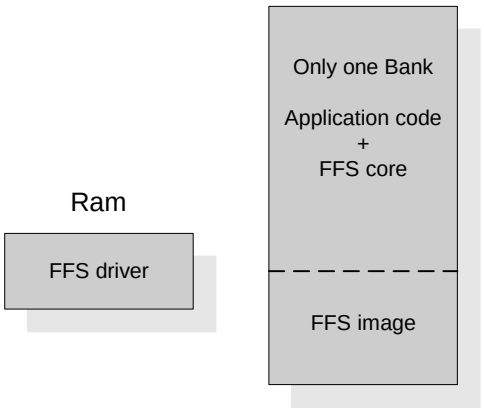

Figure 3. Single-bank flash

The SB device has the same usage restriction as the multi bank (MB) device; however the restriction is bigger because there is only one flash bank available. It is not possible to execute code while the flash is busy thus it is necessary to run the driver code from another device. By copying the driver to ram we can execute the code while erasing or writing to the flash device.
Another issue which has to be taken care is of interrupts, which will cause the CPU to run code from the flash and that would crash if the flash device are in write or erase mode. Interrupts can be disabled but the application code must not be blocked for the complete erase duration thus it is necessary to poll on the interrupts while they are disabled. If an interrupt is detected while the erase procedure has disabled it, the erase will be suspended and the application code will be allowed to run. When FFS gets the CPU again, it will disable the interrupts and resume the erase.
The big disadvantage in the SB drivers is that the block erase will be very often suspended thus the erase time will be increased. The write performance in the SB drivers is better than the MB driver because the driver is executed faster in ram than flash.

2.1.3.2 Multi-bank (MB)

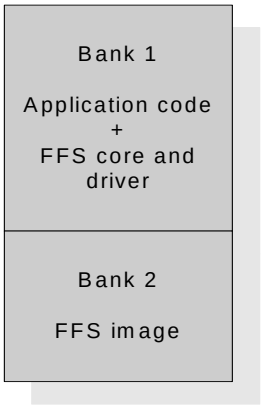

Figure 4. Multi-bank flash

The MB device has a usage restriction. It is not able to read from a busy block (erasing or writing). All FFS read functions are blocking calls and it is not allowed to block these reads while waiting for the erase to finish. To resolve this problem the erase suspend and resume commands are used. The erase operation is suspended while reading from the same bank and when the reading is finished then the erase operation is resumed.
Concerning the MB performance compared to the SB device, the erase is only interrupted while reading from FFS not while the application code is running. Thus the MB driver is faster when it erases flash blocks.
The application code is not allowed to reside in the bank, which is used for the FFS blocks.

2.1.3.3 Driver configuration

FFS can auto detect the flash device by querying the manufacture and device ID from the flash device and depending of the queried information configuration.

The configuration consists of the following information:

- Manufacture ID (Available from device datasheet)
- Device ID (Available from device datasheet)
- Base address (Absolute address of the first block to be used by/for FFS)
- Driver type (Intel, AMD or SST, also SB or MB)
- Device memory map (Number of blocks and there size)

Typically the application code will be placed at the beginning of the flash and the remaining part of the flash will be used for FFS.

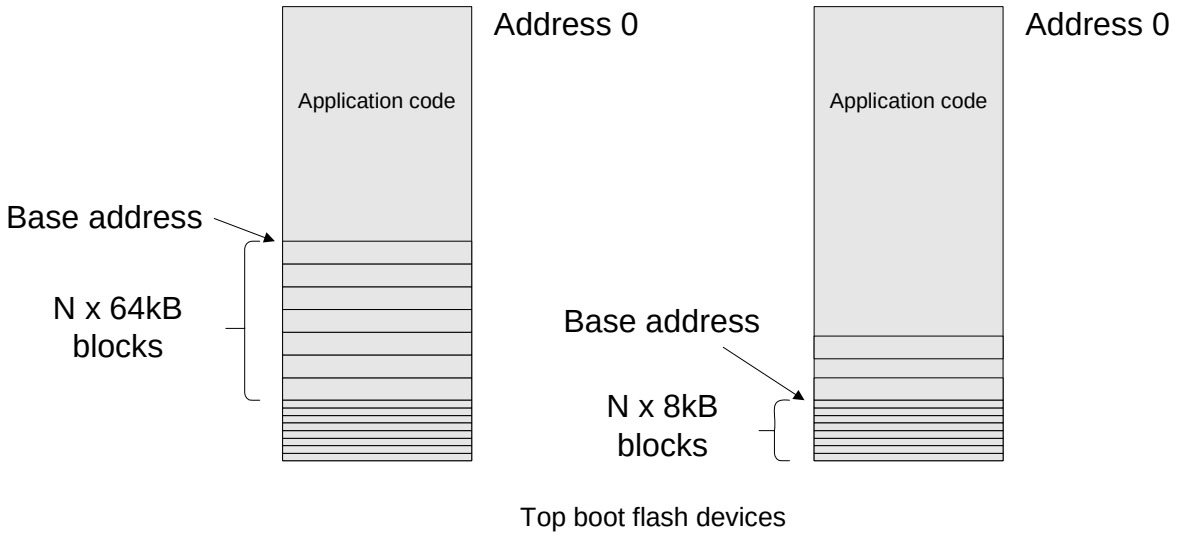

Figure 5. Typical configurations of driver.

3 Locosto FFS Driver interface

The porting of new Flash part takes impact on the Flash Driver implementation and this section focus more on understanding the Locosto FFS driver Interface functions and Data structures.

3.1 Locosto FFS driver Data Structures

3.1.1 ffsdrv_s

ffsdrv_s	Flash low level driver function pointers

Synopsis:

```
struct ffsdrv_s {
        int  (* init)(void);
        void (* erase)(uint8 block);
        void (* write_buffer)(volatile uint16 *dst, volatile uint16 *src,
                uint16 num_words );
        void (* write_halfword)(volatile uint16 *dst, uint16 value);
        void (* write)(void *dst, const void *src, uint16 size);
        void (* write_suspend)(void);
        void (* write_resume)(void);
        void (* erase_suspend)(void);
        void (* erase_resume)(void);
        void (* erase_sector)(void *dst);

    };
```

Example:

```
const struct ffsdrv_s ffsdrv_intel_bw = {
                                        ffsdrv_null_init,
                                        ffsdrv_intel_erase,
                                        ffsdrv_intel_write_buffer_mode,
                                        ffsdrv_intel_write_halfword,
                                        ffsdrv_generic_buffer_write,
                                        ffsdrv_intel_write_suspend,
                                        ffsdrv_intel_write_resume,
                                        ffsdrv_intel_erase_suspend,
                                        ffsdrv_intel_erase_resume,
                                        ffsdrv_intel_erase_sector
                    };
```

3.1.2 flash_info_s

flash_info_s	General flash information for one flash device

Synopsis:

```
struct flash_info_s {
   const struct block_info_s *binfo; // block info array for this device
   char   *base;      // base flash address of ffs blocks
   uint16 manufact;   // read with flash A0 = 0
   uint16 device;     // read with flash A0 = 1
   uint8  driver;     // flash driver type
   uint8  numblocks;  // number of blocks defined for use by ffs
};
```

Example:

```
const struct flash_info_s flash_info[ ] =
{

        // Intel 28F640W30-B, 64Mb. (DSample). Using top-most 15x64kB sectors
        { &flash_16x64[0], (char *) 0x700000, MANUFACT_INTEL,   0x88FF,
        FFS_DRIVER_INTEL_SB, 15 },

        // Intel StrataFlash (ESample).
        { &flash_8x128[0], (char *) 0x4700000, MANUFACT_INTEL,   0x8812,
        FFS_DRIVER_INTEL_BW, 8 }
}
```

3.1.3 block_info_s

block_info_s	Flash block information for one ffs block (flash sector). Note that the ffs block definition might be of a smaller size than the physical flash sector. The ffs blocks must be defined in ascending order of addresses.

Synopsis:

```
struct block_info_s {
        uint32 offset;
        uint8  size_ld;   // log2 of block size
        uint8  unused1;
        uint8  unused2;
        uint8  unused3;
        };
```

Example:

```
// 8x128kB
static const struct block_info_s flash_8x128[ ] =
        {
                { 0x00000, 17 },
                { 0x20000, 17 },
```

```
                                        { 0x40000, 17 },
                                        { 0x60000, 17 },
                                        { 0x80000, 17 },
                                        { 0xA0000, 17 },
                                        { 0xC0000, 17 },
                                        { 0xE0000, 17 }
                        };
```

3.1.4 FFS_DRIVER

FFS_DRIVER Flash driver identifiers

Synopsis:

```
enum FFS_DRIVER
 {
FFS_DRIVER_NULL         =  0, // Null driver
FFS_DRIVER_AMD          =  2, // AMD dual/multi-bank driver
FFS_DRIVER_AMD_SB       =  3, // AMD single-bank driver
FFS_DRIVER_AMD_NOR      =  4,  // for locosto NOR flash driver, Spansion
                               // multibank driver
FFS_DRIVER_AMD_MIRROR_BIT = 5,       // for locosto Mirror bit NOR flash
                                     //driver, Spansion multibank driver

 //OMAPS62129
FFS_DRIVER_AMD_NOR_PSEUDO_SB =  6,   // for locosto NOR flash driver,
                // Spansion multibank driver in pseudo SB mode to allow
                // reuse of the 3Mb flash in same bank as FFS for code /data

FFS_DRIVER_INTEL        = 16, // Intel dual/multi-bank driver
FFS_DRIVER_INTEL_SB     = 17, // Intel single-bank driver
FFS_DRIVER_INTEL_BW     = 18, // Intel buffer write driver

FFS_DRIVER_AMD_PSEUDO_SB = 32, // Test driver
FFS_DRIVER_TEST         = 34, // Test driver
FFS_DRIVER_TEST_BW      = 35, // Test driver (use buffer write)

FFS_DRIVER_RAM          = 64  // Ram driver
};
```

3.1.5 FFS_MANUFACTURER

FFS_MANUFACTURER Manufacturer identifiers. These should never have to be changed.They are ordered in alphabetically ascending order.

Synopsis:

```
enum FFS_MANUFACTURER {
                MANUFACT_AMD     = 0x01,
                MANUFACT_ATMEL   = 0x1F,
                MANUFACT_FUJITSU = 0x04,
```

```
MANUFACT_INTEL   = 0x89,
MANUFACT_MXIC    = 0xC2,
MANUFACT_SAMSUNG = 0xEC,
MANUFACT_SHARP   = 0xB0,
MANUFACT_ST      = 0x20,
MANUFACT_SST     = 0xBF,
MANUFACT_TOSHIBA = 0x98,
MANUFACT_RAM     = 0xFE, // Ram
MANUFACT_TEST    = 0x54  // 'T'est manufacturer
};
```

3.1.6 flash_base_addr

flash_base_addr	base address of FLASH

Synopsis:

```
extern volatile uint32 flash_base_addr;
// 0x6000000 for I sample
```

3.1.7 #define directives used

```
// Commands for Intel flash memory devices
#define INTEL_READ_ARRAY          (0xFF)
#define INTEL_READ_STATUS         (0x70)
#define INTEL_CLR_STATUS          (0x50)
#define INTEL_WORD_PRG            (0x40)
#define INTEL_BUFFER_PRG          (0xE8)
#define INTEL_BUFFER_PRG_CONFIRM  (0xD0)
#define INTEL_LOCK_SETUP          (0x60)
#define INTEL_UNLOCK_BLK          (0xD0)
#define INTEL_ERASE_CONFIRM       (0xD0)
#define INTEL_BLOCK_ERASE         (0x20)
#define INTEL_SUSPEND             (0xB0)
#define INTEL_RESUME              (0xD0)
```

4 Porting to “Intel Wireless Flash (L18) with PSRAM” FLASH Device

This section will focus on following topics:

- Key features of the new Flash Part
- Difference in data sheets.
- Data structure needs to modify to adapt new hardware.
- Functions need to be modified to adapt new hardware.

Given below are the key features of “L18 with PSRAM”:

- 64Kw (128Kb) Block size – present Locosto Flash driver supports.
- Multi Bank (refer section 2.1.3.2) memory map - present Locosto Flash driver supports.
- Asymmetrically Blocked Array Architecture - present Locosto Flash driver supports.
- Low-power buffered programming - present Locosto Flash driver supports.
- Buffered Enhanced Factory Programming - present Locosto Flash driver doesn’t support.
- PSRAM interface – 32 or 64 M bit – present Locosto Flash driver doesn’t support.

Considering the Locosto supported Intel flash parts (refer Table 2), the nearest match to “L18 with PSRAM” is “Intel Strata Flash® Wireless Memory (**L30**) – 128-Mbit Top Boot Device” with device id 0x8812 (refer Table 2).

4.1 Difference in data sheets.(Intel ‘L30’ & ‘L18 with PSRAM’)

4.1.1 Based on Memory Map

The present Flash (based on nearest match to “L18 with PSRAM” in supported FLASH devices) array has asymmetrical physical partitioning and blocking architecture; partitions are 8-Mbit in size for 64-Mbit and 128-Mbit devices. The 64-Mbit device has 8 partitions, while the 128-Mbit device has 16 partitions. The partition size for 256-Mbit device is 16Mbit and the device has 16 partitions. The parameter partition contains four 16-Kword parameter blocks and seven 64-Kword main blocks. The remaining main partition contains eight 64-Kword main blocks each. Based on the above given memory map it matches with the top

The memory map of “L18 with PSRAM” Flash is based on stacking individual 128-Mbit and 256-Mbit flash die density options, and it matches with the memory map of L30 128Mbit & 256Mbit flash.

4.1.2 Based on RAM interface

The present Flash (based on nearest match to “L18 with PSRAM” in supported FLASH devices) doesn’t support PSRAM feature.

The synchronous PSRAM (Non-Mux I/O interface) is a high-performance volatile memory operating at speeds up to 54 MHz with configurable burst lengths. The PSRAM lower sixteen addresses can be

routed to the data pins on the PCB board to enable a flexible flash and PSRAM A/D-Mux I/O interface device design.

4.1.2.1 PSRAM Operating Modes

The PSRAM can be used in three different modes:

• SRAM (full asynchronous) mode: In this mode the PSRAM applies the standard asynchronous SRAM protocol to perform read and write accesses. In additions, reads may be performed in page mode. In this mode the clock must always remain static low.
• Fully Synchronous mode: In this mode, both read and write accesses are performed synchronously with respect to the clock.
• NOR-Flash mode: In this mode, reads are performed synchronously with respect to the clock and writes are performed asynchronously. The asynchronous write operation requires that the clock remain static low during the entire write.

4.1.2.2 PSRAM Control Registers

The two control registers define the PSRAM device operation. The Bus Control Register (BCR) defines how the PSRAM interacts with the system memory busy, and the Refresh Control Register (RCR) defines low-power refresh modes. Both these registers are loaded with default values on power-up and can be updated at any time using hardware or software access method. Please refer the data sheet for the description of these registers.

4.1.2.3 Programming PSRAM Registers

Software access of the control registers uses a sequence of asynchronous read and asynchronous write operations.

Loading the control registers is a four-step sequence, consisting of two asynchronous read operations followed by two asynchronous write operations (see Fig 6). The read sequence is virtually identical, except that an asynchronous read is performed during the fourth operation (see Fig 7). The address used during all read and write operations is the highest address being accessed (7FFFFFh for 128 Mb, 3FFFFFh for 64 Mb, and 1FFFFFh for 32 Mb). This sequence does not change the contents of this address.

The data value presented during the third operation (write) in the sequence defines whether to access the BCR or the RCR.

• If the data is 0000h, the sequence accesses the RCR.
• If the data is 0001h, the sequence accesses the BCR.

The fourth operation uses the data bus to transfer data in to or out of the control registers.

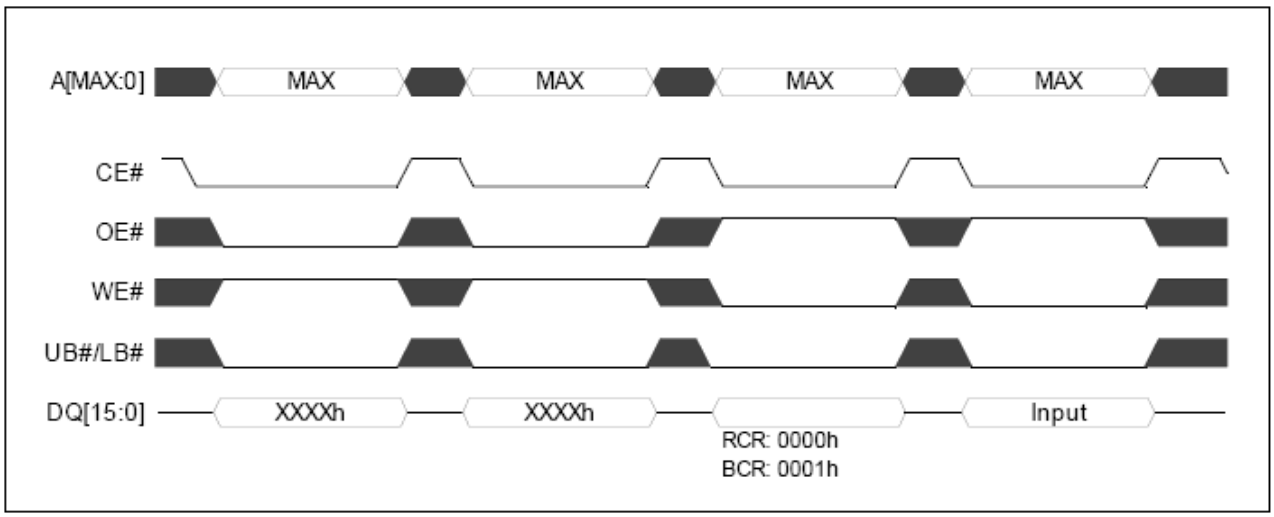

Figure 6. PSRAM loading configurations registers using software access (Taken from Intel® Wireless Flash Memory (L18) with Synchronous PSRAM data sheet).

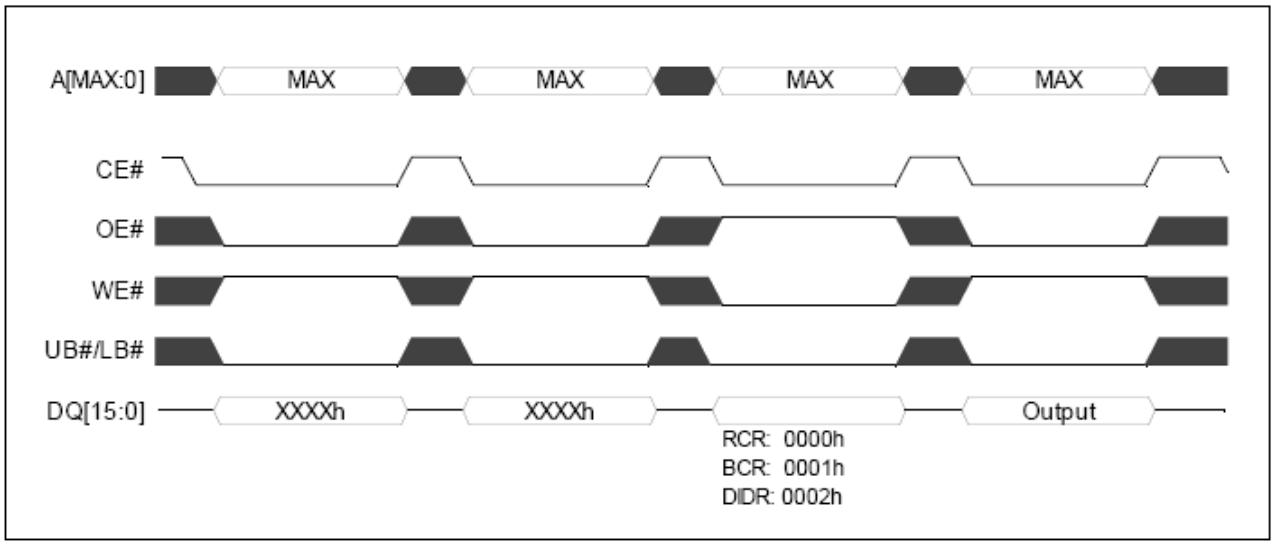

Figure 7. PSRAM reading registers using software access (Taken from Intel® Wireless Flash Memory (L18) with Synchronous PSRAM data sheet).

NOTE: - We have covered software aspects for the FFS porting in the above given section.

4.2 Data Structure & defines need to modify/create:

4.2.1 flash_info (refer section 2.1.2)

flash_info	General flash information for one flash device

We need to add one more element in this array of structure to add support of “L18 with PSRAM” in the code.

4.2.2 *ffsdrv_s* ffsdrv_intel_L18_PSRAM (refer section 2.1.1)

Ffsdrv_intel_L18_PSRAM	Flash low level driver function pointers

We need to add one function pointer structure to support “L18 with PSRAM” driver functions.

Note: *The structure “ffsdrv_s” exposes the supported functionalities of the driver to the upper layer; any new features/functionality would need a modification in this structure.*

4.2.3 # defines directives need to add with hardware

- #define **INTEL_PSRAM_RCR_SELECT** (0x0000) // to select RCR
- #define **INTEL_PSRAM_BSR_SELECT** (0x0001) // to select BCR
- #define **INTEL_PSRAM_RCR_CFG** (0xPQ) // **PQ** = depends on initial settings
- #define **INTEL_PSRAM_BSR_CFG** (0xRS) // **RS** = depends on initial settings

4.3 Functions need to modify/Create :

```
1. void ffsdrv_intel_L18_PSRAM_init(void)
```

Description

This function performs the initialization of PSRAM for the intel FLASH in this driver.

Parameters

- **Void**

Immediate Return

- **Void**

Possible changes required

Need to add this function which will do the initial setting in BCR & RCR for PSRAM initialization. or the L30 initialization is not required, So the ffsdrv_null_init has been kept in the structure initialization. This function need to be replaced with “*ffsdrv_intel_L18_PSRAM*” (refer section 4.2.2).

References

- Intel® Wireless Flash Memory (L18) 768-Mbit L18 Family with Synchronous PSRAM data sheet.
- Locosto Design Specification Flash File System.
- Intel StrataFlash® Wireless Memory (L18 with A/D-Multiplexed I/O SCSP) data sheet.
- Intel StrataFlash® Wireless Memory (L30) data sheet.
- http://www.spansion.com/flash_memory_technology/mirrorbit.html
- “Intel StrataFlashTM Memory Technology Overview” application note.

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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