Использование флеш устройств

HAL предлагает общие модели устройств для чипов энергонезависимой флеш памяти. Флеш память использует специальный программный протокол для сохранения данных. HAL API предлагает функции для записи данных во флеш память. Например, вы можете использовать эти функции для реализации файловой подсистемы на флеш.

HAL API также предлагает функции для чтения флеш, несмотря на то, что они в основном не нужны. Для большинства флеш чипов, программы во время чтения могут интерпретировать пространство флеш памяти как простую память, и им не требуется вызов специальных функций HAL API. Если флеш чип имеет специальный протокол для чтения данных, как в стираемом программируемом последовательно конфигурированном (EPCS) чипе конфигурации Altera, вы должны использовать HAL API для чтения и записи данных.

В этой секции описывается HAL API для моделей флеш устройств. Следующие два API предлагают два уровня доступа к флеш:

- Простой доступ к флеш – функции, которые записывают в буферы для флеш, и затем читают обратно на уровне блока. Во время записи, если буфер меньше чем полный блок, эти функции стирают предыдущие данные во флеш выше и пишут новые данные ниже.
- Тонкоструктурный доступ к флеш - функции, которые записывают в буферы для флеш, и затем читают обратно на уровне буфера. Во время записи, если буфер меньше чем полный блок, эти функции сохраняют предыдущие данные во флеш выше и пишут новые данные ниже.

Эти функции необходимы для управления файловой подсистемой.

Функции API для доступа к флеш устройствам определены в файле **sys/alt_flash.h**.

За дополнительной информацией об использовании этих функций, обратитесь к главе "Справка по HAL API" в настольной книге программиста. Вы можете детально изучить общий интерфейс с флеш (CFI), включая организацию стирания регионов и блоков в общем интерфейсе с флеш, от JEDEC (www.jedec.org). Вы можете найти стандарт CFI, отыскав документ JESD68.

Простой доступ к флеш

Этот интерфейс состоит из функций alt_flash_open_dev(), alt_write_flash(), alt_read_flash() и alt_flash_close_dev(). Код "Использование простых API функций с флеш" на стр. 6-22 показывает использование всех этих функций в одном примере кода. Вы открываете флеш устройство, вызовом alt_flash_open_dev(), который возвращает обработчик файла во флеш чипе. Эта функция имеет единственный аргумент, который имеет имя флеш чипа, определённое в **system.h**.

После завершения обработки, вы можете использовать функцию alt_write_flash(), для записи данных во флеш чип.

```
int alt_write_flash( alt_flash_fd* fd,
                     int           offset,
                     const void*   src_addr,
                     int           length )
```

Вызов этой функции записывает во флеш чип, идентифицированный обработчиком **fd**. Драйвер записывает данные, начиная с **offset** байтов базы флеш чипа. Данные записываются, начиная указателя адреса **src_addr**, а длина записываемых данных – это **length**.

Здесь есть функция alt_read_flash() для чтения данных из флеш чипа. Её прототип:

```
int alt_read_flash( alt_flash_fd* fd,
                    int           offset,
                    void*         dest_addr,
                    int           length )
```

Вызов функции alt_read_flash() читает из флеш чипа обработчиком **fd**, начиная с **offset** байтов базы флеш чипа. Данные читаются, начиная указателя адреса **dest_addr**, а длина читаемых данных – это **length**. Для большинства флеш чипов, вы можете получить доступ как к стандартной памяти, делая ненужной функцию alt_read_flash().

Функция alt_flash_close_dev() принимает обработку файла и закрывает чип. Её прототип:

```
void alt_flash_close_dev(alt_flash_fd* fd )
```

Код в примере 6-10 показывает, как использовать простые API функции флеш для доступа к флеш чипам под именем **/dev/ext_flash**, как это определено в **system.h**.

Стирание или повреждение блока

В основном флеш память делится на блоки. Функции alt_write_flash() необходимо выполнить стирание содержимого блока перед тем, как она сможет записать в него данные. В таком случае, она не пытается сохранить существующее содержимое блока. Это действие может привести к непредсказуемым повреждениям данных (стирание), если вы осуществляете запись, не попадающую в границы блока. Если вы хотите сохранить существующее содержимое флеш памяти, используйте тонкоструктурные флеш функции. Они обсуждаются в следующей секции.

Табл. 6-7 на стр. 6-23 показывает, как вы можете вызвать непредсказуемое повреждение данных, используя простые функции доступа к флеш. Табл. 6-7 показывает пример 8-килобайтной (КБ) флеш памяти, поделённой на два 4-килобайтных блока. Сначала пишется 5 КБ из 0xAA во флеш память по адресу 0x0000, а затем пишется 2 КБ из 0xBB по адресу 0x1400. После завершения первой записи (по времени t(2)), флеш память состоит из 5 КБ из 0xAA, её остаток пуст (это значит 0xFF). Перед началом второй записи во второй блок он стирается. На точке t(3) флеш состоит из 4 КБ из 0xAA и 4 КБ из 0xFF. После завершения второй записи, время t(4), 2 КБ из 0xFF по адресу 0x1000 повреждены.

Тонкоструктурный доступ к флеш

Три дополнительные функции предоставляют полный контроль за записью содержимого во флеш память с наивысшей глубиной детализации:

- alt_get_flash_info()
- alt_erase_flash_block()
- alt_write_flash_block()

По природе устройства флеш памяти, вы не сможете стереть одиночный адрес в блоке. Вы должны стереть одновременно весь блок (состоящий из набора одиночных адресов). Запись во флеш память может только изменить биты от 1 в 0; чтобы изменить любой бит с 0 на 1, вы должны стереть весь блок, включающий этот бит.

Поэтому, чтобы передвинуть заданную позицию в блоке, предохраняя от изменения окружающее содержимое, вы должны прочитать это содержимое блока в буфер, сдвинуть значение в буфере, стереть флеш блок и, затем, записать буфер размером с блок обратно во флеш память. Функции тонкоструктурного доступа к флеш памяти автоматизируют этот процесс на уровне флеш блока.

Example 6–10. Using the Simple Flash API Functions

```
#include <stdio.h>
#include <string.h>
#include "sys/alt_flash.h"
#define BUF_SIZE 1024

int main ()
{
  alt_flash_fd* fd;
  int           ret_code;
  char          source[BUF_SIZE];
  char          dest[BUF_SIZE];

  /* Initialize the source buffer to all 0xAA */
  memset(source, 0xAA, BUF_SIZE);

  fd = alt_flash_open_dev("/dev/ext_flash");
  if (fd!=NULL)
  {
    ret_code = alt_write_flash(fd, 0, source, BUF_SIZE);
    if (ret_code==0)
    {
      ret_code = alt_read_flash(fd, 0, dest, BUF_SIZE);
      if (ret_code==0)
      {
        /*
         * Success.
         * At this point, the flash is all 0xAA and we
         * have read that all back to dest
         */
      }
    }
    alt_flash_close_dev(fd);
  }
  else
  {
    printf("Cannot open flash device\n");
  }
  return 0;
}
```

Функция alt_get_flash_info() получает количество стираемых регионов, количество стираемых блоков в каждом регионе и размер каждого стираемого блока. Прототип функции следующий:

```
int alt_get_flash_info (
    alt_flash_fd*  fd,
    flash_region** info,
    int*           number_of_regions )
```

Если вызов был удачен, она возвращает указатель адреса **number_of_regions** содержащий количество стираемых регионов во флеш памяти, и ***info points** в виде массива структур **flash_region**. Этот массив является частью дескриптора файла.

Table 6–7. Example of Writing Flash and Causing Unexpected Data Corruption

Address	Block	Time t(0)	Time t(1)	Time t(2)	Time t(3)	Time t(4)
			First Write		Second Write	
		Before First Write	After Erasing Block(s)	After Writing Data 1	After Erasing Block(s)	After Writing Data 2
0x0000	1	??	FF	AA	AA	AA
0x0400	1	??	FF	AA	AA	AA
0x0800	1	??	FF	AA	AA	AA
0x0C00	1	??	FF	AA	AA	AA
0x1000	2	??	FF	AA	FF	FF *(1)*
0x1400	2	??	FF	FF	FF	BB
0x1800	2	??	FF	FF	FF	BB
0x1C00	2	??	FF	FF	FF	FF

Примечание к табл. 6-7:
(1) Ненарочное сбрасывание в FF во время стирания для второй записи.

Структура flash_region определена в файле **sys/alt_flash_types.h**. Структура данных представлена так:

```
typedef struct flash_region
{
  int offset;         /* Offset of this region from start of the flash */
  int region_size;    /* Size of this erase region */
  int number_of_blocks;  /* Number of blocks in this region */
  int block_size;     /* Size of each block in this erase region */
}flash_region;
```

Вместе с информацией, получаемой при вызове alt_get_flash_info(), вы получаете позицию для стирания или программирования отдельных блоков флеш памяти. Функция alt_erase_flash() стирает один блок в флеш памяти, она имеет следующий прототип:

```
int alt_erase_flash_block ( alt_flash_fd* fd, int offset, int length )
```

Флеш память идентифицируется указателем **fd**. Блок идентифицируется как **offset** байт от начала флеш памяти, а размер блока сохраняется в **length**.

Функция alt_write_flash_block() записывает в один блок флеш памяти, её прототип следующий:

```
int alt_write_flash_block( alt_flash_fd* fd,
                           int           block_offset,
                           int           data_offset,
                           const void    *data,
                           int           length)
```

Эта функция записывает во флеш памяти, идентифицируемую указателем **fd**. Она записывает в блок, находящийся в **block_offset** байтах относительно стартового адреса флеш чипа. Функция записывает **length** байт данных от места, указанного данными в **data_offset** байтах относительно стартового адреса флеш чипа.

Эти функции программирования и стирания не выполняют проверку адреса и не могут верифицировать, попала ли операция записи в следующий блок. Вы должны задавать правильную информацию о блоках перед программированием и стиранием.

Код в примере 6-11 на стр. 6-24 показывает использование функция тонкоструктурного доступа к флеш памяти.

Example 6–11. Using the Fine-Grained Flash Access API Functions

```
#include <string.h>
#include "sys/alt_flash.h"
#include "stdtypes.h"
#include "system.h"#define BUF_SIZE 100

int main (void)
{
  flash_region* regions;
  alt_flash_fd* fd;
  int           number_of_regions;
  int           ret_code;
  char          write_data[BUF_SIZE];

  /* Set write_data to all 0xa */
  memset(write_data, 0xA, BUF_SIZE);

  fd = alt_flash_open_dev(EXT_FLASH_NAME);

  if (fd)
  {
    ret_code = alt_get_flash_info(fd, &regions, &number_of_regions);

    if (number_of_regions && (regions->offset == 0))
    {
      /* Erase the first block */
      ret_code = alt_erase_flash_block(fd,
                                       regions->offset,
                                       regions->block_size);
      if (ret_code == 0)      {
         /*
          * Write BUF_SIZE bytes from write_data 100 bytes to
          * the first block of the flash
          */
         ret_code = alt_write_flash_block (
            fd,
            regions->offset,
            regions->offset+0x100,
            write_data,
            BUF_SIZE );
      }
    }
  }
  return 0;
}
```