Создание драйверов для HAL классов устройств

HAL поддерживает набор классов групповых моделей устройств. При написании драйвера устройства, как будет описано в этой секции, вы описываете для HAL элемент определённого устройства, который попадает в один из известных классов устройств. В этой секции определяется совместимый интерфейс для функций драйвера, с помощью которого HAL может получить постоянный доступ к функциям драйвера.

Классы групповых моделей устройств определены в главе "Общее представление о слое аппаратной абстракции" в настольной книге программиста под Nios II.

В следующих секциях определены API для следующих классов устройств:

- Устройства с символьным режимом
- Файловая подсистема
- Устройства DMA
- Устройства таймеры системного тактового сигнала
- Устройства таймеры временной метки
- Устройства флеш памяти
- Устройства Ethernet

В следующих секциях описано, как реализовать драйверы устройств для каждого класса устройств, и как зарегистрировать их для использования в HAL системе.

Драйверы устройств с символьным режимом

В этой секции описывается, как создавать модуль устройства и регистрировать символьное устройство.

Создание нового элемента устройства

Чтобы сделать устройство доступным в системе как устройство с символьным режимом, его необходимо представить в качестве элемента структуры alt_dev. Код в примере 7-1 определяет alt_dev структуру.

Структура alt_dev, расположенная в *<Nios II EDS install path>***/components/ altera_hal/HAL/inc/sys/alt_dev.h,** по существу является коллекцией указателей функции. Эти функции вызываются в ответ на запрос приложения к файловой системе HAL. Например, если вы вызываете функцию open() с именем файла, отвечающего за это устройство, то в результате вы вызываете функцию open(), предоставленную в этой структуре.

Example 7–1. alt_dev Structure

```
typedef struct {
  alt_llist    llist;     /* for internal use */
  const char*  name;
  int (*open)  (alt_fd* fd, const char* name, int flags, int mode);
  int (*close) (alt_fd* fd);
  int (*read)  (alt_fd* fd, char* ptr, int len);
  int (*write) (alt_fd* fd, const char* ptr, int len);
  int (*lseek) (alt_fd* fd, int ptr, int dir);
  int (*fstat) (alt_fd* fd, struct stat* buf);
  int (*ioctl) (alt_fd* fd, int req, void* arg);
} alt_dev;
```

За дополнительной информацией о функциях open(), close(), read(), write(), lseek(),fstat() и ioctl() обратитесь к главе "Справка по HAL API" в настольной книге программиста под Nios II.

Никакие из этих функций напрямую не модифицируют глобальный статус ошибки, errno. Вместо этого, возвращаемое значение есть отрицание соответствующего кода ошибки, предлагаемого в **errno.h**.

Например, функция ioctl() возвращает -ENOTTY, если она не может обработать запрос раньше, чем errno установится прямо в ENOTTY. Программы системы HAL, вызывающие эти функции, следят за тем, чтобы errno устанавливалось именно таким образом.

Прототипы этих функций отличаются от их эквивалента на уровне приложения в том, что каждая из них имеет аргумент входного файлового дескриптора типа alt_fd*, а не int.

Создаётся новая структура alt_fd при обращении к open(). Этот элемент структуры считается входным аргументом для всех вызовов функций, сделанных к ассоциированному файловому дескриптору.

В следующем коде определена структура alt_fd:

```
typedef struct
{
    alt_dev* dev;
    void*    priv;
    int      fd_flags;
} alt_fd;
```

где:

- dev – это указатель на структуру устройства, для использованного устройства.
- fd_flags – это значение флагов, относящихся к open().

- priv – зарезервированый для различной реализации аргумент, определяемый драйвером. Если драйверу требуется специальное, не относящееся к HAL, значение для обслуживания каждого файла или потока, вы можете сохранить его в структуре данных и использовать поддержку указателя структуры priv. HAL игнорирует priv.

Распределите память для структуры данных в вашей функции open() (указатель в структуре alt_dev). Высвобождайте память в вашей функции close().

Чтобы избежать утечки памяти, следите за тем, чтобы функция close() вызывалась только тогда, когда файл или поток больше не понадобятся.

Драйверу не нужно использовать все функции структуры alt_dev. Если некоторый указатель функции установлен на NULL, взамен её используется действие по умолчанию. В табл. 7-1 показаны действия по умолчанию для каждой доступной функции.

Табл. 7-1. Поведение по умолчанию функций alt_dev

Функция	Поведение по умолчанию
open	Если вызов open() не успешен, устройство имеет предыдущее сохранённое значение по вызову ioctl() с req = TIOCEXCL.
close	Вызывает close() с правильным файловым дескриптором для этого устройства, всегда успешно.
read	Вызов read() для этого устройства всегда не успешен.
write	Вызов write() для этого устройства всегда не успешен.
lseek	Вызов lseek() для этого устройства всегда не успешен.
fstat	Устройство идентифицирует fstat в качестве устройства с символьным режимом.
ioctl	ioctl() не возможно обработать без ссылки на не успешное обращение к устройству.

В дополнении к указателям функций, структура alt_dev содержит два других поля: llist и name. Поле llist необходимо для внутреннего использования, оно должно всегда иметь значение ALT_LLIST_ENTRY. Поле name – это место устройства в файловой системе HAL, это имя устройства, определённое в **system.h**.

Регистрирование символьного устройства

После того, как вы создадите элемент структуры alt_dev, устройство нужно сделать доступным в системе, зарегистрировав его в HAL, и вызвав следующую функцию:

```
int alt_dev_reg (alt_dev* dev)
```

Эта функция имеет один входной аргумент, который является регистрируемой структурой устройства. При успешной регистрации, возвращаемое значение – нуль. Отрицательное возвращаемое значение означает, что устройство не зарегистрировано.

После того, как устройство зарегистрировано в файловой системе HAL, вы получаете доступ к нему через HAL API и стандартную библиотеку ANSI Си. Имя узла устройства – это имя, заданное в структуре alt_dev.

За дополнительной информацией обратитесь к главе "Разработка программ с использованием слоя аппаратной абстракции" в настольной книге программиста под Nios II.