Example 7–5. Supporting Driver Settings

```
#include "system.h"
#ifdef MY_CUSTOM_DRIVER_SMALL
int send_data( <args> )
{
// Small implementation
}
#else
int send_data( <args> )
{
// fast implementation
}
#endif
```

В примере 7-5 в Tcl файл вашего драйвера добавлена простая настройка булево выражения. Это средство позволяет BSP пользователю контролировать ваш драйвер в настройках интерфейса BSP. Когда пользователь устанавливает настройку как true или 1, BSP определяет MY_CUSTOM_DRIVER_SMALL либо в **system.h**, либо в BSP **public.mk** файле. Когда пользователь компилирует BSP, ваш драйвер компилируется с соответствующей процедурой, находящейся в объектном файле. Когда пользователь снимает эту настройку, MY_CUSTOM_DRIVER_SMALL не определяется.

Вы добавляете настройку MY_CUSTOM_DRIVER_SMALL в ваш драйвер с помощью Tcl команды add_sw_setting следующим образом:

```
add_sw_setting boolean_define_only system_h_define small_driver
MY_CUSTOM_DRIVER_SMALL false
"Enable the small implementation of the driver for my_peripheral"
```

Каждая Tcl команда должна занимать одну линию в Tcl файле. В этом примере это изменено, по причине ограничений размеров страницы.

Каждый аргумент имеет несколько вариантов. За подробной информацией об использовании и ограничениях, обратитесь к главе "Справка по Nios II SBT" в настольной книге программиста под Nios II.

Уменьшение размера кода

HAL содержит несколько опций для уменьшения размера кода, или кодового покрытия, вашего BSP кода. Некоторым из этих опций требуется непосредственная поддержка от драйверов устройств. Если вам нужно уменьшить размер кода вашей программы, выберите одну или обе следующие методики в вашем собственном драйвере устройства:

- Предоставление драйверов с уменьшенным кодовым покрытием. Эта методика обычно уменьшает функциональность драйвера.
- Поддержка облегчённых драйверов устройств API. Эта методика уменьшает избыточность драйвера. Она может не снизить его функциональность, но должна ограничить гибкость в его использовании.

Эти методики обсуждаются в следующих секциях.

Предоставление драйверов с уменьшенным кодовым покрытием

HAL определяет макрос предпроцессора Си, называемый ALT_USE_SMALL_DRIVERS, который вы можете использовать в исходном коде драйвера для предоставления альтернативного поведения системы, которой необходимо минимальное кодовое покрытие. Если ALT_USE_SMALL_DRIVERS не определено, исходный код драйвера реализует полно функциональную версию драйвера. Если макрос определён, исходный код должен предоставить драйвер ограниченной функциональности. Например, драйвер должен реализовывать операцию направления прерываний по умолчанию, но урезает операцию (и возможно, уменьшается), если определён ALT_USE_SMALL_DRIVERS.

Когда вы пишете драйвер устройства, если вы выбираете игнорировать значение ALT_USE_SMALL_DRIVERS, то используется эта версия драйвера независимо от определений в этом макросе.

За подробной информацией обратитесь к главе "Справка по Nios II SBT" в настольной книге программиста под Nios II.

Поддержка облегчённых драйверов устройств API

Облегчённые API драйверы устройств позволяют вам минимизировать избыточность драйверов устройств с символьным режимом. Это достигается исключением потребности в таблице дескрипторов (описания) alt_fd и в структуре данных alt_dev, необходимой в каждом элементе драйвера.

Если вы хотите поддерживать облегчённые API драйверы устройств для устройств с символьным режимом, вам потребуется написать не менее одной функции облегченного символьного режима, представленной в табл. 7-7. Например, если вы используете устройство только для stdout, вам всего лишь нужно реализовать функцию <*component class*>_*write*().

Для поддержки облегчённых API драйверов устройств, имя вашей функции должно быть основано на имени класса компонента, как показано в табл. 7-7.

Табл. 7-7. Функции драйвера для облегчённого API драйвера устройства

Функция	**Назначение**	**Пример (1)**
<*component class*>_*read*()	Реализует функцию чтения в символьном режиме	altera_avalon_jtag_uart_read()
<*component class*>_*write()*	Реализует функцию записи в символьном режиме	altera_avalon_jtag_uart_write()
<*component class*>_*ioctl()*	Реализует функции, зависимые от устройства	altera_avalon_jtag_uart_ioctl()

(1) На примере модуля **altera_avalon_jtag_uart**

Когда вы собираете ваш BSP с разрешённым ALT_USE_DIRECT_DRIVERS, в зависимости от используемых файловых дескрипторов, HAL становятся доступны следующие макросы:

- ALT_DRIVER_READ(instance, buffer, len, flags)
- ALT_DRIVER_WRITE(instance, buffer, len, flags)
- ALT_DRIVER_IOCTL(instance, req, arg)

Эти макросы определены в <*Nios II EDS install path*>**/components/altera_hal/HAL/inc/sys/alt_driver.h.** Они вместе со специфическими макросами системы, которые создаёт Nios II SBT в **system.h**, генерирую вызовы функций вашего драйвера. Например, при включенном облегчённом драйвере, printf() вызывает HAL функцию write(), которая прямо вызывает вашу функцию драйвера <*component class*>_write(), пропуская файловые дескрипторы.

Вы можете разрешить ALT_USE_DIRECT_DRIVERS в BSP с помощью настройки BSP hal.enable_lightweight_device_driver_api.

За подробной информацией обратитесь к главе "Справка по Nios II SBT" в настольной книге программиста под Nios II.

Вы можете получить преимущества от облегчённого API драйвера устройства, вызывая ALT_DRIVER_READ(), ALT_DRIVER_WRITE() и ALT_DRIVER_IOCTL() в вашем программном приложении. Чтобы использовать эти макросы, включите в заголовочный файл **sys/alt_driver.h**. Замените аргумент элемента именем макроса элемента устройства из **system.h**; или если вы уверены, что имя элемента устройства никогда не будет изменено, вы можете использовать символьную строку, например, custom_uart_0.

Другим способом использовать функции вашего драйвера, это вызывать их напрямую, без макроса. Если ваш драйвер содержит функции, не являющиеся <*component class*>_read(),<*component class*>_write() и <*component class*>_ioctl(), вы должны вызывать эти функции прямо из приложения.

Распределение пространства имён

Чтобы избежать конфликта имён для символов, определённых устройствами в системе SOPC Builder, все глобальные символы должны иметь определённый префикс. Глобальные символы содержат имена глобальных переменных и функций. Для драйверов устройств, префикс – это имя компонента SOPC Builder, следующее после подчёркивания. Поскольку эти имена составляют длинные строки, допускается использовать короткую форму записи. Такая форма записи основана на имени производителя, например, **alt_** - это префикс для компонентов, опубликованных Altera. Подразумевается, что производители тестируют на совместимость все свои компоненты.

Например, для компонента altera_avalon_jtag_uart, правильными являются следующие имена функций:

- altera_avalon_jtag_uart_init()
- alt_jtag_uart_init()

А следующие имена неправильные:

- avalon_jtag_uart_init()
- jtag_uart_init()

Исходные файлы локализуются с использованием путей поиска, такие ограничения для пространства имён также применяются для имён файлов драйверов устройств и заголовочных файлов.

Перезапись драйверов устройств по умолчанию

Всем компонентам SOPC Builder могут поставляться драйверы устройств HAL. Обратитесь к секции "Интегрирование драйвера устройства в HAL" на стр. 7-17. Однако, если драйвер, поставляемый для компонента не подходит для вашего приложения, вы можете перезаписать драйвер по умолчанию другим драйвером.

В Nios II SBT на Eclipse, вы можете использовать редактор BSP для установки своего собственного драйвера.

За информацией о выборе драйверов устройств, обратитесь к секции "Использование BSP редактора" в главе "Начало работы в графической оболочке" в настольной книге программиста Nios II.

В командной строке, вы можете устанавливать собственный драйвер с помощью следующей BSP Tcl команды:

set_driver <*driver name*> <*component name*>

Например, если вы используете команду **nios2-bsp**, вы заменяете драйвер по умолчанию для uart0 драйвером, называемым custom_driver, следующим образом:

```
nios2-bsp hal my_bsp --cmd set_driver custom_driver uart0
```