7. Developing Device Drivers for the Hardware Abstraction Layer

NII52005-10.0.0

7. Разработка драйверов устройств для слоя аппаратной абстракции (HAL)

Введение

Встроенные системы обычно содержат аппаратные средства специального применения, которым требуется собственный драйвер устройства. В этой главе описано, как разрабатывать драйверы устройств и интегрировать их в слой аппаратной абстракции (HAL).

Эта глава также описывает, как разрабатывать пакет программ для использования в HAL пакете поддержки платы (BSP). Процесс интегрирования пакета программ в HAL очень похож на процесс интегрирования драйвера устройства.

Эта глава состоит из следующих секций:

Ограничивайте прямую аппаратную интеграцию для кода драйвера устройства. Лучший способ – это освободить значительную часть кода вашей программы от низкоуровневого доступа к устройству. Старайтесь использовать функции высокоуровневого HAL интерфейса прикладного программирования (API) для доступа к устройству. Это сделает ваш код совместимым с другими Nios II системами, другой аппаратной конфигурации.

Когда вы создаёте новый драйвер, вы можете интегрировать драйвер с HAL структурой (framework) на одном из двух уровней:

- Интеграция в HAL API
- Специальное API для периферии

В качестве альтернативы создания драйвера, вы можете скомпилировать специальный код устройства в качестве пользовательской библиотеки и связать его с приложением. Этот способ будет работать, если специальный код устройства не зависит от BSP, и не требует каких-либо специальных сервисов, доступных в BSP, таких как, способность добавлять определения в файл **system.h**.

Интеграция в HAL API

Интеграция в HAL API – это предпочтительная опция для периферии, которая относится к одному из HAL классов групповой модели устройств, таких как устройства с символьным режимом или устройства прямого доступа к памяти (DMA).

За описанием HAL классов групповой модели устройств обратитесь к главе "Общее представление о слое аппаратной абстракции" в настольной книге программиста под Nios II.

Для интеграции в HAL API вы пишете функции доступа к устройству, как приводится в этой главе, чтобы устройство стало доступным для программы посредством стандартного HAL API. Например, у вас есть новое устройство отображения LCD, которое отображает символы ASCII, вы пишете драйвер устройства с символьным режимом. С помощью этого драйвера, кое-где в программе вы вызываете знакомую функцию printf() для организации потока символов на LCD экран.

Специальное API для периферии

Если периферия не подходит ни под один HAL класс групповой модели устройств, вам необходимо предоставить драйвер устройства с интерфейсом, который подходит под аппаратную реализацию. В этом случае API для устройства отличается от HAL API. Программы получают доступ к устройству посредством вызова вашей (а не HAL API) функции.

Попытка реализации интеграции в HAL API не столько самоцель, сколько реальная выгода для вас в использовании API HAL и стандартной библиотеки Си для управления устройствами.

За подробной информацией об интеграции в HAL API обратитесь к секции "Интегрирование драйвера устройства в HAL" на стр. 7-17.

Во всех остальных секциях этой главы рассказывается об интегрировании драйверов в HAL API и создании драйверов специального API для периферии.

Несмотря на то, что C++ поддерживается программами, основанными на HAL, драйверы HAL не могут быть написаны на C++. Описывайте код вашего драйвера на Си или на ассемблере. Си более подходит для совместимости.

Прежде чем начать

Эта глава подразумевает, что вы знакомы с программированием на Си под HAL.

Обратитесь к главе "Разработка программ с использованием слоя аппаратной абстракции" в настольной книге программиста под Nios II за необходимой информацией перед чтением этой главы.

Эта глава использует переменную *<Altera installation>* для обозначения директории инсталляции Altera® Complete Design Suite. В системе Windows, по умолчанию, это **c:/altera/***<version number>*.

Процесс создания драйверов устройств

Процесс создания нового драйвера под HAL зависит от реализации вашего устройства. Однако следующая группа пунктов применяется ко всем классам устройств:

1. Создайте заголовочный файл устройства, в котором описываются регистры. Этот заголовочный файл служит только для организации интерфейса.
2. Реализуйте функциональную схему драйвера.
3. Протестируйте из main().
4. Направляйтесь по пути окончательной интеграции драйвера в среду HAL.
5. Интегрируйте драйвер устройства в HAL структуру (framework).

Понятия в SOPC Builder

В этой секции обсуждаются основные понятия в инструменте разработки аппаратуры Altera SOPC Builder, которые дадут вам понимание процесса разработки драйвера. Вы можете также разрабатывать драйверы устройств Nios II и без SOPC Builder.

Взаимосвязь между system.h и SOPC Builder

Заголовочный файл **system.h** содержит полное программное описание аппаратной части системы Nios II и является фундаментом разработки драйверов. Поскольку драйверы взаимодействуют с устройствами на аппаратном уровне, стоит упомянуть взаимосвязь между **system.h** и SOPC Builder, который генерирует аппаратную часть системы Nios II. Разработчики аппаратной части используют SOPC Builder для задания архитектуры процессорной системы Nios II, они интегрируют необходимую периферию и память. Поэтому определённые значения в **system.h**, такие как имя и конфигурация каждой периферии, имеют прямую взаимосвязь настройками проекта, сделанными в SOPC Builder.

За дополнительной информацией о заголовочном файле **system.h** обратитесь к главе "Разработка программ с использованием слоя аппаратной абстракции" в настольной книге программиста под Nios II.

Использование SOPC Builder для оптимальной аппаратной конфигурации

Если вы находите менее оптимальные определения в файле **system.h**, помните, что вы можете изменить содержимое **system.h**, изменив базовое аппаратное устройство с помощью SOPC Builder. Прежде чем написать драйвер устройства, приспособив его под несовершенное устройство, рассмотрите возможность более простого пути аппаратного улучшения устройства с помощью SOPC Builder.

Компоненты, устройства и периферия

SOPC Builder использует термин "компонент" для описания аппаратных модулей в составе системы. В контексте разработки программы под Nios II, компоненты SOPC Builder – это устройства, такие как периферия и память. В следующих секциях, "компонент" используется наравне с "устройством" и "периферией", когда рассматриваются понятия, относящиеся только к SOPC Builder.

Аппаратный доступ

Программа имеет доступ к аппаратной части посредством макроса, который абстрагирует интерфейс с распределением в памяти для устройства. В этой секции описывается макрос, который определяет аппаратный интерфейс с каждым устройством.

Все компоненты SOPC Builder находятся в директории, определяющей программную и аппаратную части устройства. Например, каждый компонент в программе Quartus® II имеет свою собственную директорию в директории <*Altera installation*>**/ip/altera/sopc_builder_ip**. Многие компоненты имеют заголовочный файл, в котором определён их аппаратный интерфейс. Заголовочный файла называется <*component name*>_**regs.h**, он содержится в подпапке специальных компонентов **inc**. Например, компонент Altera JTAG UART определяет свой аппаратный интерфейс в файле <*Altera installation*>**/ip/altera/sopc_builder_ip/altera_avalon_jtag_uart/inc/ altera_avalon_jtag_uart_regs.h**.

Заголовочный файл _**regs.h** определяет следующий макрос доступа к компоненту:

- Макрос регистра доступа, который предлагает макрос чтения и (или) записи для каждого регистра компонента, который поддерживает эту операцию. Макрос следующий:
  - IORD_<имя компонента>_<имя регистра>(<базовый адрес компонента>)
  - IOWR_<имя компонента>_<имя регистра>(<базовый адрес компонента>, <*данные*>)

Например, **altera_avalon_jtag_uart_regs.h** определяет следующий макрос:

- IORD_ALTERA_AVALON_JTAG_UART_DATA()
- IOWR_ALTERA_AVALON_JTAG_UART_DATA()
- IORD_ALTERA_AVALON_JTAG_UART_CONTROL()
- IOWR_ALTERA_AVALON_JTAG_UART_CONTROL()

- Макрос регистра адреса, который возвращает физический адрес каждого регистра компонента. Регистр адреса возвращает базовый адрес компонента + заданное значение офсета регистра. Макрос называется **IOADDR**_<*имя компонента*>_<*имя регистра*> (<*базовый адрес компонента*>).

Например, **altera_avalon_jtag_uart_regs.h** определяет следующий макрос:

- IOADDR_ALTERA_AVALON_JTAG_UART_DATA()
- IOADDR_ALTERA_AVALON_JTAG_UART_CONTROL()

Используйте эти макросы только как параметры функции, в которой нужно задать адрес исходных данных или адрес назначения. Например, программе, которая читает поток данных из особого регистра компонента, необходимо задать физический адрес регистра в качестве параметра.

- Маска битового поля и офсет, позволяющий доступ к конкретным битовым полям регистра. Такой макрос имеет следующее название:
  - <имя компонента>_<имя регистра>_<имя поля>_MSK – маска битового поля
  - <имя компонента>_<имя регистра>_<имя поля>_ OFST – офсет бит в начале поля.

Например, доступ к полю PE ALTERA_AVALON_UART_STATUS_PE_MSK и ALTERA_AVALON_UART_STATUS_PE_OFST регистра статуса.

Доступ к регистрам устройств только через макрос, заданный в **_regs.h** файле. Вы должны использовать функции доступа к регистру, чтобы процессор обходил кэш данных во время записи или чтения с устройством. Не используйте жестко запрограммированные константы, поскольку это делает вашу программу восприимчивой к изменениям в основных аппаратных средствах. Если вы пишете драйвер для нового законченного устройства, вы должны изучить заголовочный файл **_regs.h**.

За подробной информацией о разработке драйверов устройств для HAL BSP, обратитесь к *AN 459: Руководство по разработке Nios II HAL драйвера устройства*. Законченные примеры **_regs.h** файла находятся в директории компонентов для любого поставляемого Altera компонента SOPC Builder, например в *<Altera installation>***/ip/ sopc_builder_ip/altera_avalon_jtag_uart/inc.** За дополнительной информацией об эффекте управления кэшем и доступе к устройству, обратитесь к главе "Кэш и прочно сопряжённая память" в настольной книге программиста под Nios II.