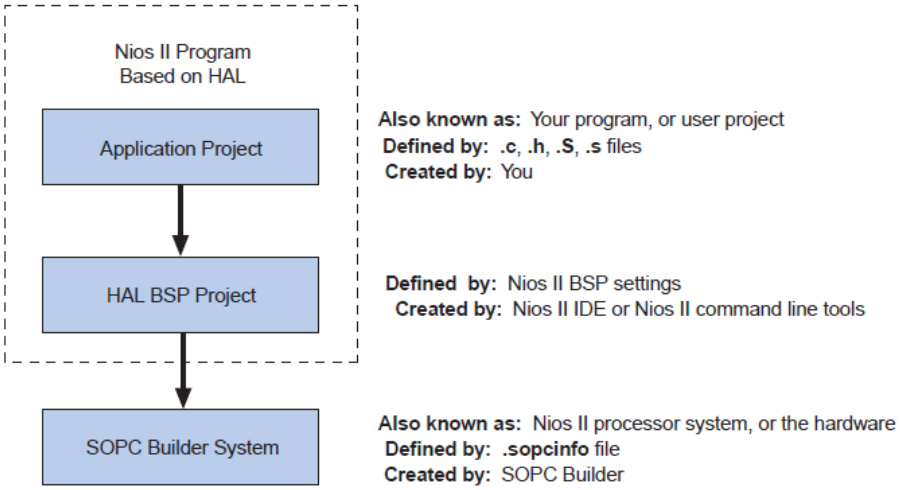
Структура проекта Nios II

Создание и управление программными проектами, использующими HAL, тесно связано с Nios II SBT. В этой секции описываются Nios II проекты в качестве основы понимания HAL.

На рис. 6-1 показаны блоки Nios II программы с акцентом на том, как её компонует HAL BSP. Метка на каждом блоке описывает, как или кто генерирует этот блок, а стрелки показывают зависимости каждого блока.

Figure 6–1. The Nios II HAL Project Structure



Каждая Nios II программа, использующая HAL, состоит из двух проектов Nios II, как показано на рис. 6-1. Ваш код приложения составляет один проект (проект пользовательского приложения), который зависит от другого отдельного BSP проекта (HAL BSP).

Проект приложения состоит из всего разработанного вами кода. Рабочий образ вашей программы напрямую зависит от сборки обоих проектов.

С помощью Nios II SBT на Eclipse, инструменты создают проект HAL BSP, когда вы создаёте ваш проект приложения. В командной строке вы можете создать проект, используя команду **nios2-bsp** или похожий инструмент.

Проект HAL BSP содержит всю необходимую информацию для организации интерфейса вашей программной и аппаратной частей проекта. Драйверы HAL обращаются к вашей системе SOPC Builder, уже включённой в BSP проект.

Проект BSP зависит от системы SOPC Builder, определяемой в файле информации SOPC (**.sopcinfo**). Nios II SBT может сохранять ваш BSP с обновлённой системой SOPC Builder. Эта проектно зависимая структура изолирует вашу программу от изменений в основном устройстве, а вы можете разрабатывать и отлаживать код не обращая внимание на то, будет ли ваша программа работать в выбранной аппаратной части.

Вы можете использовать Nios II SBT для обновления вашего BSP, чтобы создать обновлённую аппаратную часть. Вы контролируете, когда и как происходят эти обновления.

Подробнее о том, как SBT может сохранять ваш BSP с обновлённой аппаратной системой, обратитесь к секции "Изменение вашего BSP" в главе "Инструмент разработки программ Nios II" в настольной книге программиста Nios II.

В общем, когда ваша программа базируется на HAL BSP, вы всегда сможете сохранить её синхронизацию с выбранной аппаратной частью с помощью нескольких простых команд SBT.

Файл описания системы - system.h

Файл **system.h** содержит полное программное описание аппаратной системы Nios II. Не вся информация в файле **system.h** необходима вам как программисту, и она редко востребована для того, чтобы включить её в ваши исходные файлы Си. Однако, файл **system.h** содержит ответ на вопрос: "Какое устройство этой системы?"

В файле **system.h** описана вся периферия системы в следующих подробностях:

- аппаратная конфигурация периферии,
- базовый адрес,
- информация о запросе прерываний (IRQ) (если нужно),
- символьное имя периферии.

Nios II SBT генерирует **system.h** файлы для проектов HAL BSP. Содержание файла **system.h** зависит от аппаратной конфигурации и свойствах HAL BSP.

Ни в коем случае не редактируйте файл **system.h**. SBT предоставляет вам инструменты для управления настройками системы.

За подробной информацией о контроле настроек BSP, обратитесь к секции "Настройки HAL BSP" на странице 6-2.

Код в примере 6-1 из файла **system.h** показывает некоторые опции аппаратной конфигурации, определённые в этом файле.

Example 6–1. Excerpts from a system.h File

```
/*
* sys_clk_timer configuration
*
*/

#define SYS_CLK_TIMER_NAME "/dev/sys_clk_timer"
#define SYS_CLK_TIMER_TYPE "altera_avalon_timer"
#define SYS_CLK_TIMER_BASE 0x00920800
#define SYS_CLK_TIMER_IRQ 0
#define SYS_CLK_TIMER_ALWAYS_RUN 0
#define SYS_CLK_TIMER_FIXED_PERIOD 0

/*
* jtag_uart configuration
*
*/

#define JTAG_UART_NAME "/dev/jtag_uart"
#define JTAG_UART_TYPE "altera_avalon_jtag_uart"
#define JTAG_UART_BASE 0x00920820
#define JTAG_UART_IRQ 1
```

Ширина данных и определение типов в HAL

Для встроенных процессоров, таких как Nios II, очень важно знать точную ширину и тип данных. Поскольку типы данных в ANSI Си не прямо определяют ширину данных, HAL использует набор стандартных определений типов. Типы данных ANSI Си также поддерживаются, но их ширина данных зависит от соглашений переносимости кода в компиляторах.

Заголовочный файл **alt_types.h** определяет типы в HAL; в табл. 6-1 показано определение типов в HAL.

Table 6–1. The HAL Type Definitions

Type	Meaning
alt_8	Signed 8-bit integer.
alt_u8	Unsigned 8-bit integer.
alt_16	Signed 16-bit integer.
alt_u16	Unsigned 16-bit integer.
alt_32	Signed 32-bit integer.
alt_u32	Unsigned 32-bit integer.
alt_64	Signed 64-bit integer.
alt_u64	Unsigned 64-bit integer.

В табл. 6-2 показана ширина данных, которая используется в предлагаемых Altera GNU инструментальных схемах.

Table 6–2. GNU Toolchain Data Widths

Type	Meaning
char	8 bits.
short	16 bits.
long	32 bits.
int	32 bits.

,