Использование файловой подсистемы

Общая модель устройств HAL для файловой подсистемы позволяет доступ к данным, хранящимся в ассоциированном устройстве памяти, используя I/O функции файла стандартной библиотеки Си. Например, доступная только для чтения файловая подсистема Altera zip предоставляет доступ только чтение к файлам, хранящимся во флеш памяти.

Файловая подсистема отвечает за управления всеми файлами I/O доступа ниже установленной верхней точки. Например, если файловая подсистема зарегистрирована с верхней точкой **/mnt/rozipfs**, то все файлы доступны ниже этой директории, например fopen("/mnt/rozipfs/myfile", "r") направляется прямо к этой файловой подсистеме.

В случае с устройствами с символьным режимом, вы можете манипулировать файлами в файловой подсистеме, используя I/O функции файла Си, определённые в **file.h**, такие как fopen() и fread().

За дополнительной информацией об использовании I/O функции файла обратитесь к документации на библиотеку Си newlib, установленной вместе с Nios II EDS. В Windows в меню Пуск, кликните **Programs** > **Altera** > **Nios II <*version*>** > **Nios II EDS <*version*> Documentation**.

Файловая система, расположенная в хост-машине

Файловая система, расположенная в хост-машине, позволяет исполнять программы на рабочей плате, читая и записывая файлы, хранящиеся на хост-машине. Nios II SBT для Eclipse передаёт файлы данных через загрузочный кабель Altera. Ваша программа получает доступ к файловой системе, расположенной в хост-машине, используя I/O функции стандартной библиотеки ANSI Си, таких как fopen() и fread().

Файловая система, расположенная в хост-машине, это пакет программ, который вы добавляете в свой BSP. Следующие средства и ограничения применяются к файловой системе, расположенной в хост-машине:

- Файловая система, расположенная в хост-машине, создаёт директорию приложения Nios II C/C++ и поддиректории, доступные для файловой системы HAL на рабочей плате.
- Выбранный процессор имеет доступ только файлам в директории проекта. Будьте осторожны, не повреждайте исходные файлы проекта.

- Файловая система, расположенная в хост-машине, работает только на стадии отладки проекта. Она не может быть использована на стадии прогона.
- Файлы данных перемещаются между хост-машиной и рабочей платой последовательно через загрузочный кабель Altera, поэтому время доступа к файлу медленное. В зависимости от конфигурации вашей хост системы и системы в рабочей плате, обращение к хост-машине может занимать несколько миллисекунд. Для улучшения характеристик, используйте буферизированные I/O функции, такие как fread() и fwrite(), и увеличьте размер буфера для длинных файлов.

Вы конфигурируете файловую систему, расположенную в хост-машине, используя редактор Nios II BSP. Файловая система, расположенная в хост-машине, имеет одну настройку: верхнюю точку, которая задаёт верхнюю точку внутри файловой системы HAL. Например, если имя вашей верхней точки **/software/project1**, то HAL программа использует следующий код для открытия файла **/software/project1/datafile.dat.**:

```
fopen("/mnt/host/datafile.dat", "r");
```

Использование устройств таймеров

Устройства таймеры – это аппаратная периферия, которая подсчитывает тактовые импульсы и может генерировать периодический запрос на прерывание. Вы можете использовать устройства таймеры для предоставления некоторого количества связанного с таймерами оборудования, такого как системный такт HAL, сигнал тревоги, время суток и измерение времени. Для использования оборудования таймеров, процессорная система Nios II должна иметь аппаратную периферию таймеров.

HAL API предлагает два вида драйверов устройств таймеров:

- Драйвер системного тактового сигнала – поддерживает сигнал тревоги, который может быть использован, например, в планировщике.
- Драйвер временной метки – поддерживает измерение времени с высоким разрешением.

Индивидуальная периферия таймера может быть использована в качестве системного тактового сигнала или в качестве таймера временной метки, но никак не для обоих таймеров.

Специальные API функции HAL для доступа к устройствам таймерам определены в файлах **sys/alt_alarm.h** и **sys/alt_timestamp.h**.

Драйвер системного тактового сигнала

Драйвер системного тактового сигнала HAL предлагает периодические тактовые импульсы, вызывающие приращение системного такта на каждый такт. Программа может использовать оборудование системного тактового сигнала для выполнения функций за определённое время, и для получения информации о времени. Вы выбираете определённую аппаратную периферию таймера в качестве устройства системного тактового сигнала, манипулируя настройками BSP.

Подробная информация о контроле настроек BSP находится в секции "Настройки HAL BSP" на стр. 6-2.

HAL предлагает реализацию следующих стандартных функций UNIX: gettimeofday(), settimeofday() и times(). Время, возвращаемое этими функциями, основано на системном тактовом сигнале HAL.

Системный тактовый сигнал измеряет время в тактовых импульсах. Системным инженерам, которые занимаются аппаратными и программными средствами, не нужно путать системный тактовый сигнал HAL с тактовым сигналом, поступающим на аппаратную часть процессора Nios II. Период тактовых импульсов HAL системы в основном много длиннее, чем аппаратный системный такт. Файл **system.h** определяет частоту тактовых импульсов.

На стадии прогона вы можете получить значение истекшего системного тактового сигнала, вызвав функцию alt_nticks(). Эта функция возвращает текущее время системных тактовых импульсов с момента сброса. Вы можете получить системную тактовую частоту (импульсы в секунду), вызвав функцию alt_ticks_per_second(). Драйвер таймера HAL инициализирует тактовую частоту, когда создаёт элемент системного такта.

Стандартная UNIX функция gettimeofday() доступна для получения текущего времени. Вы должны сначала откалибровать время суток, вызвав settimeofday(). Дополнительно вы можете использовать функцию times() для получения информации о количестве истекших тактов. Прототипы этих функций доступны в **times.h**.

За дополнительной информацией об использовании этих функций, обратитесь к главе "Справка по HAL API" в настольной книге программиста Nios II.

Сигнал тревоги

Вы можете регистрировать функции, которые будут выполняться в определённое время, используя сигнал тревоги HAL. Регистры, запрограммированные в программе на сигнал тревоги, вызывают функцию:

alt_alarm_start():

```
int alt_alarm_start (alt_alarm* alarm,
                     alt_u32    nticks,
                     alt_u32    (*callback) (void* context),
                     void*      context);
```

Функция callback() вызывается после прохождения функции nticks. Входной аргумент context сохраняется как входной аргумент в callback(), когда происходит вызов функции. HAL не может использовать параметр context. Он может использоваться только в качестве параметра для функции callback().

Ваш код должен быть распределён в структуре alt_alarm, с указанием на входной аргумент alarm. Такая структура данных должна иметь время существования не менее чем alarm. Лучший способ распределить эту структуру – это декларировать её как static или global. alt_alarm_start() инициализирует *alarm.

Функция обратного вызова может сбросить сигнал тревоги. Возвращаемое значение регистрированной функции обратного вызова – это количество тактов до следующего вызова функции обратного вызова. Возвращаемое значение нуль показывает, что сигнал тревоги остановлен. Вы можете вручную сбросить сигнал тревоги вызовом alt_alarm_stop().

Один сигнал тревоги создаётся для каждого вызова alt_alarm_start(). Несколько сигналов тревоги могут быть запущены последовательно.

Функции обратного вызова работают в контексте исключения. Это предполагает функциональные ограничения, которым вы должны следовать при написании функции обратного вызова сигнала тревоги.

За дополнительной информацией об использовании этих функций, обратитесь к главе "Обработка исключений" в настольной книге программиста Nios II.

Фрагмент кода в примере 6-8 показывает регистрирование сигнала тревоги в периодической функции обратного вызова каждую секунду.

Example 6–8. Using a Periodic Alarm Callback Function

```
#include <stddef.h>
#include <stdio.h>
#include "sys/alt_alarm.h"
#include "alt_types.h"

/*
* The callback function.
*/

alt_u32 my_alarm_callback (void* context)
{
  /* This function is called once per second */
  return alt_ticks_per_second();
}

...

/* The alt_alarm must persist for the duration of the alarm. */
static alt_alarm alarm;

...

  if (alt_alarm_start (&alarm,
                       alt_ticks_per_second(),
                       my_alarm_callback,
                       NULL) < 0)
  {
    printf ("No system clock available\n");
  }
```

Драйвер временной метки

Зачастую вы хотите измерять временные интервалы с большей точностью, чем предоставляется с помощью системных тактовых импульсов HAL. HAL предлагает функции с высоким разрешением по времени, использующие драйвер временной метки. Драйвер временной метки представляет собой монотонный счётчик прямого счёта, который мы можете использовать для получения информации о времени. HAL поддерживает только один драйвер временной метки в системе.

Вы задаёте периферию аппаратного таймера как устройство временной метки, манипулируя настройками BSP. Предлагаемый Altera драйвер временной метки использует заданный вами таймер.

Если предустановлен драйвер временной метки, доступны следующие функции:

- alt_timestamp_start()
- alt_timestamp()

Вызов alt_timestamp_start() запускает счётчик. Последовательный вызов alt_timestamp() возвращает текущее значение счётчика временной метки. Повторный вызов alt_timestamp_start() сбрасывает счётчик в нуль. Поведение драйвера временной метки невозможно предугадать, когда счётчик достигает значения (2^{32} - 1).

Вы можете получить частоту инкремента счётчика временной метки, вызвав функцию alt_timestamp_freq(). Эта частота является, по обыкновению, аппаратной частотой процессорной системы Nios II – обычно миллион циклов в секунду. Драйвер временной метки определяется в заголовочном файле **alt_timestamp.h**.

За дополнительной информацией об использовании этих функций, обратитесь к главе "Справка по HAL API" в настольной книге программиста Nios II.

Фрагмент кода в примере 6-9 показывает, как использовать средство временной метки в коде измерения времени исполнения.

Example 6–9. Using the Timestamp to Measure Code Execution Time

```
#include <stdio.h>
#include "sys/alt_timestamp.h"
#include "alt_types.h"

int main (void)
{
  alt_u32 time1;
  alt_u32 time2;
  alt_u32 time3;

  if (alt_timestamp_start() < 0)
  {
    printf ("No timestamp device available\n");
  }
  else
  {
    time1 = alt_timestamp();
    func1(); /* first function to monitor */
    time2 = alt_timestamp();
    func2(); /* second function to monitor */
    time3 = alt_timestamp();

    printf ("time in func1 = %u ticks\n",
            (unsigned int) (time2 - time1));
    printf ("time in func2 = %u ticks\n",
            (unsigned int) (time3 - time2));
    printf ("Number of ticks per second = %u\n",
            (unsigned int)alt_timestamp_freq());
  }
  return 0;
}
```