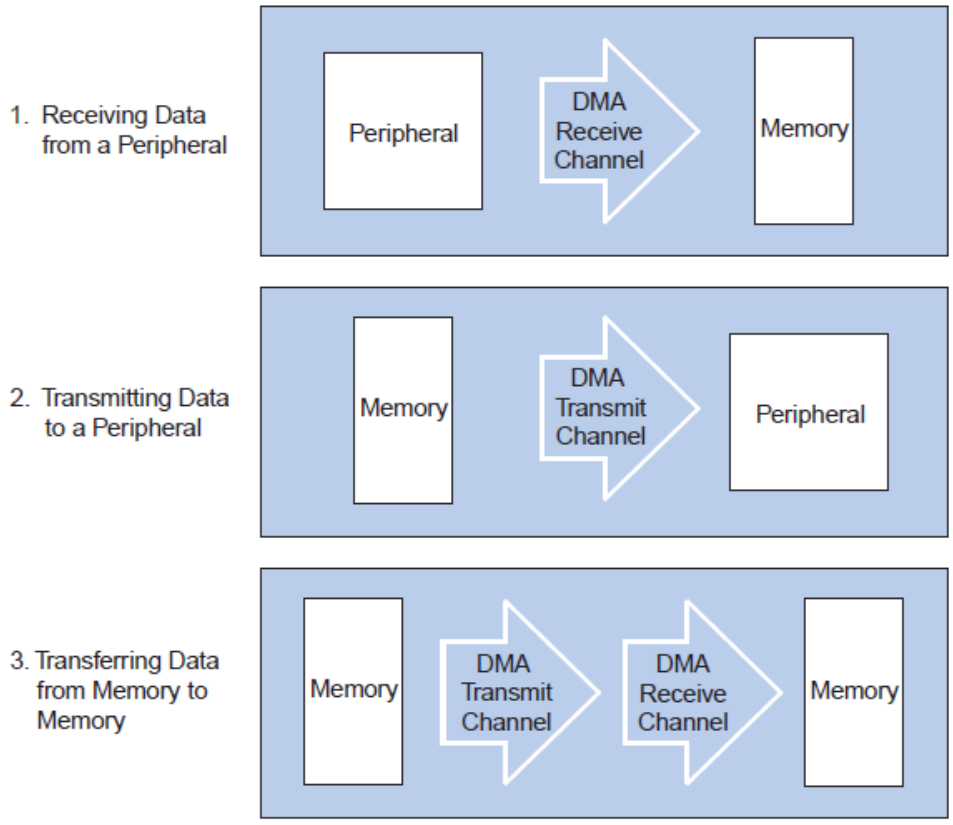
Использование DMA устройств

HAL предоставляет абстрактную модель устройства для устройств с прямым доступом к памяти (DMA). Это такая периферия, которая выполняет тотальные транзакции данных от источника до приёмника данных. Источники и приёмники могут быть памятью или прочими устройствами, например, соединением Ethernet.

В модели устройства HAL DMA существуют два типа DMA транзакций: передача и приём. HAL предоставляет два драйвера устройств для реализации каналов передачи и приёма. Канал передачи забирает данные из буфера передатчика и передаёт их в приёмное устройство. Канал приёма принимает данные от устройства и сохраняет их в буфере приёмника. В зависимости от реализации на аппаратном уровне, программа может иметь доступ только к одной из двух этих оконечных точек.

На рис. 6-2 показаны три основных типа DMA транзакций. Копирование данных из памяти в память вовлекает соответственно оба DMA канала - передачи и приёма.

Figure 6–2. Three Basic Types of DMA Transactions



API для доступа к устройствам DMA определены в **sys/alt_dma.h**.

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

Устройства DMA работают с содержимым физической памяти, поэтому при чтении и записи данных, вы должны учитывать взаимодействие с кэшем.

За дополнительной информацией о кэш памяти, обратитесь к главе "Кэш и точно сопряжённая память" в настольной книге программиста Nios II.

DMA канал передачи

Запросы передачи DMA обрабатываются по очереди, используя устройство обработки передачи DMA. Чтобы осуществить обработку, используйте функцию alt_dma_txchan_open(). Эта функция использует единственный аргумент, имя используемого устройства, определённого в файле **system.h**.

Код в примере 6–12 демонстрирует осуществление обработки для устройства передачи DMA - dma_0.

Example 6–12. Obtaining a File Handle for a DMA Device

```
#include <stddef.h>
#include "sys/alt_dma.h"

int main (void)
{
  alt_dma_txchan tx;

  tx = alt_dma_txchan_open ("/dev/dma_0");
  if (tx == NULL)
  {
    /* Error */
  }
  else
  {
    /* Success */
  }
  return 0;
}
```

Вы можете использовать эту обработку для регистрирования запроса передачи, используя функцию alt_dma_txchan_send(). Её прототип следующий:

```
typedef void (alt_txchan_done)(void* handle);

int alt_dma_txchan_send (alt_dma_txchan   dma,
                         const void*      from,
                         alt_u32          length,
                         alt_txchan_done* done,
                         void*            handle);
```

Вызов alt_dma_txchan_send() регистрирует запрос передачи по каналу **dma**. Аргумент **length** задаёт количество байтов данных для передачи, а аргумент **from** задаёт исходный адрес. Функция возвращает значение раньше, чем закончится полная транзакция DMA. Возвращённое значение показывает, когда запрос успешно поставлен в очередь для обработки, пользовательская функция **done** вызывается с аргументом **handle** для предоставления извещения.

Две дополнительные функции позволяют манипулировать каналами передачи DMA: alt_dma_txchan_space() и alt_dma_txchan_ioctl(). Функция alt_dma_txchan_space() возвращает количество дополнительных запросов, которые поставлены в очередь на обработку устройством. Функция alt_dma_txchan_ioctl() выполняет специфическую манипуляцию с передающим устройством.

Если вы используете устройства Avalon Memory-Mapped® (Avalon-MM) DMA для передачи в устройства (не трансферт между устройствами памяти), вызывайте функцию alt_dma_txchan_ioctl() с установленным аргументом запроса ALT_DMA_TX_ONLY_ON.

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

DMA канал приёма

Каналы приёма DMA функционируют схоже с каналами передачи DMA. Программа может обрабатывать каналы приёма DMA, используя функцию alt_dma_rxchan_open(). Вы можете использовать функцию alt_dma_rxchan_prepare() для регистрирования запросов приёма. Прототип alt_dma_rxchan_prepare() следующий:

```
typedef void (alt_rxchan_done)(void* handle, void* data);

int alt_dma_rxchan_prepare (alt_dma_rxchan   dma,
                            void*            data,
                            alt_u32          length,
                            alt_rxchan_done* done,
                            void*            handle);
```

Вызов этой функции регистрирует запрос приёма по каналу **dma**, вплоть до **length** байт данных, чтобы разместить их по адресу данных Функция возвращает значение раньше, чем закончится полная транзакция DMA. Отрицательное значение означает, что запрос потерян. Когда транзакция завершена, пользовательская функция **done()** вызывается с аргументом **handle** для предоставления извещения и указателем на принятые данные.

Обычные ошибки могут помешать завершению DMA трансферта. Обычно это вызывается внезапным аппаратным отказом; например, если компонент, участвующий в трансферте, потерял ответ на запрос чтения или записи. Если DMA трансферт не завершён (т.е. были переданы меньше чем **length** байт данных), функция **done()** никогда не вызовется.

Две дополнительные функции предоставляются для манипуляции с каналами приёма DMA: alt_dma_rxchan_depth() и alt_dma_rxchan_ioctl().

Если вы используете устройство Avalon-MM DMA для приёма от устройства (не трансферт между устройствами памяти), вызывайте функцию alt_dma_rxchan_ioctl() с установленным аргументом запроса ALT_DMA_RX_ONLY_ON.

Функция alt_dma_rxchan_depth() возвращает максимальное количество запросов приёма, которые поставлены в очередь на обработку устройством. Функция alt_dma_rxchan_ioctl() выполняет специфическую манипуляцию с передающим устройством.

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

Код в примере 6-13 показывает полный пример приложения, которое регистрирует запросы приёма DMA, и блоки в теле **main()**, пока не завершится транзакция.

Example 6–13. A DMA Transaction on a Receive Channel

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

/* flag used to indicate the transaction is complete */
volatile int dma_complete = 0;

/* function that is called when the transaction completes */
void dma_done (void* handle, void* data)
{
  dma_complete = 1;
}

int main (void)
{
  alt_u8 buffer[1024];
  alt_dma_rxchan rx;

  /* Obtain a handle for the device */
  if ((rx = alt_dma_rxchan_open ("/dev/dma_0")) == NULL)
  {
    printf ("Error: failed to open device\n");
    exit (1);
  }
  else
  {
    /* Post the receive request */
    if (alt_dma_rxchan_prepare (rx, buffer, 1024, dma_done, NULL) < 0)
    {
      printf ("Error: failed to post receive request\n");
      exit (1);
    }

    /* Wait for the transaction to complete */
    while (!dma_complete);
    printf ("Transaction complete\n");
    alt_dma_rxchan_close (rx);
  }
  return 0;
}
```

DMA транзакции между устройствами памяти

Копирование данных из одного буфера памяти в другой задействует оба DMA драйвера – приёма и передачи. Код в примере 6-14 показывает процесс постановки в очередь запрос приёма, следующий за запросом передачи, для получения DMA транзакции между устройствами памяти.

Example 6–14. Copying Data from Memory to Memory (Part 1 of 2)

```
#include <stdio.h>
#include <stdlib.h>

#include "sys/alt_dma.h"
#include "system.h"

static volatile int rx_done = 0;

/*
* Callback function that obtains notification that the data
* is received.
*/

static void done (void* handle, void* data)
{
  rx_done++;
}

/*
*
*/

int main (int argc, char* argv[], char* envp[])
{
  int rc;

  alt_dma_txchan txchan;
  alt_dma_rxchan rxchan;

  void* tx_data = (void*) 0x901000;  /* pointer to data to send */
  void* rx_buffer = (void*) 0x902000;  /* pointer to rx buffer */

  /* Create the transmit channel */

  if ((txchan = alt_dma_txchan_open("/dev/dma_0")) == NULL)
  {
   printf ("Failed to open transmit channel\n");
   exit (1);
  }

  /* Create the receive channel */

  if ((rxchan = alt_dma_rxchan_open("/dev/dma_0")) == NULL)
  {
    printf ("Failed to open receive channel\n");
    exit (1);
  }

  /* Post the transmit request */

  if ((rc = alt_dma_txchan_send (txchan,
                                 tx_data,
                                 128,
                                 NULL,
                                 NULL)) < 0)
  {
   printf ("Failed to post transmit request, reason = %i\n", rc);
   exit (1);
  }

/* Continued... */
```

Example 6–14. Copying Data from Memory to Memory (Part 2 of 2)

```
  /* Post the receive request */

  if ((rc = alt_dma_rxchan_prepare (rxchan,
                                    rx_buffer,
                                    128,
                                    done,
                                    NULL)) < 0)
  {
    printf ("Failed to post read request, reason = %i\n", rc);
    exit (1);
  }

  /* wait for transfer to complete */

  while (!rx_done);

  printf ("Transfer successful!\n");

  return 0;
}
```

Использование контроллеров прерываний

HAL поддерживает два типа контроллеров прерываний:

- внутренний контроллер прерываний Nios II
- внешний компонент контроллера прерываний

За дополнительной информацией о работе контроллеров прерываний, обратитесь к главе "Обработка исключений" в настольной книге программиста Nios II.