Программная модель

В следующих секциях описывается программная модель для ядра UART, включая карту регистров и программное декларирования для доступа к аппаратной части. Для пользователей процессором Nios II, Altera предоставляет драйверы системной библиотеки слоя аппаратной абстракции (HAL), которые разрешают вам доступ к ядру UART с использованием стандартных функций библиотеки ANSI Си, таких как printf() и getchar().

Поддержка системной библиотеки HAL

Поставляемый Altera драйвер реализовывает HAL драйвер устройства с символьным режимом, который интегрируется в системную библиотеку HAL для систем Nios II. Пользователи HAL могут иметь доступ к UART через знакомые HAL API и ANSI Си стандартные библиотеки, вместо того, чтобы иметь доступ к регистрам UART. Запрос ioctl() определён таким образом, чтобы позволить HAL пользователям контролировать определённые аппаратные аспекты UART.

Если ваша программа использует HAL драйвер устройства для доступа к аппаратной части UART, то прямое обращение к регистрам устройства мешает корректной работе драйвера.

Для пользователей процессора Nios II, системная библиотека HAL API предлагает полный доступ к средствам ядра UART. Программы Nios II рассматривают ядро UART как устройство с символьным режимом, поэтому они посылают и принимают данные, используя стандартные функции библиотеки ANSI Си.

Драйвер поддерживает контрольные сигналы CTS/RTS , которые были разрешены в SOPC Builder. Обратитесь к секции "Опции драйвера: реализация быстрой и малой версий" на стр. 6-9.

В следующем коде показано наипростейшее использование, вывод сообщения в stdout с использованием printf(). В этом примере, система SOPC Builder ограничивает ядро UART, а системная библиотека HAL сконфигурирована для использования этого устройства под stdout.

Example 6–1. Example: Printing Characters to a UART Core as stdout

```
#include <stdio.h>
int main ()
{
   printf("Hello world.\n");
   return 0;
}
```

В следующем коде показано чтение символов из и отправка сообщений в устройство UART, используя стандартную библиотеку Си. В этом примере система SOPC Builder имеет ядро UART с именем uart1, которое не нужно конфигурировать как устройство stdout. В этом случае программа рассматривает устройство схоже с другими узлами файловой системы HAL.

Example 6–2. Example: Sending and Receiving Characters

```
/* A simple program that recognizes the characters 't' and 'v' */
#include <stdio.h>
#include <string.h>
int main ()
{
   char* msg = "Detected the character 't'.\n";
   FILE* fp;
   char prompt = 0;

   fp = fopen ("/dev/uart1", "r+"); //Open file for reading and writing
   if (fp)
   {
      while (prompt != 'v')
      {  // Loop until we receive a 'v'.
         prompt = getc(fp);  // Get a character from the UART.
         if (prompt == 't')
         {  // Print a message if character is 't'.
            fwrite (msg, strlen (msg), 1, fp);
         }
      }

      fprintf(fp, "Closing the UART file.\n");
      fclose (fp);
   }

   return 0;
}
```

За дополнительной информацией о системной библиотеке HAL обратитесь к Настольной книге программиста под Nios II.

Опции драйвера: реализация быстрой и малой версий

Чтобы соответствовать требованиям различных типов систем, драйвер UART имеет два варианта: быстрая и малая версии. Быстрая версия устанавливается по умолчанию. Оба варианта драйверов (быстрый и малый) полностью поддерживают функции стандартной библиотеки Си и HAL API.

Быстрый драйвер имеет реализацию источника прерываний, которая позволяет процессору исполнять другие задачи, пока устройство не готово для отправки или приёма данных. Поскольку скорость обмена данных по UART сравнительно меньше процессора, быстрый драйвер имеет больше пользы для характеристики системы, которая может выполнять другие задачи в этот промежуток времени.

Малый драйвер реализован с периодическим опросом, он ожидает, пока устройство UART пошлёт или примет каждый символ. Существует два способа разрешения драйвера с малым кодовым покрытием:

- С помощью настройки HAL системной библиотеки – разрешить малое покрытие кода. Эта опция влияет на все драйверы устройств для всех устройств системы.
- Задать опцию предпроцессора -DALTERA_AVALON_UART_SMALL. Вы можете использовать эту опцию, когда хотите иметь малый драйвер UART с периодическим опросом, но не хотите влиять на другие драйверы устройств.

Обратитесь к разделу помощи в Nios II IDE за подробной информацией о том, как устанавливать свойства HAL и опции предпроцессора.

Если аппаратно разрешены сигналы контроля над процессом CTS/RTS, быстрый драйвер автоматически их использует. Малый драйвер игнорирует их.

Операции ioctl()

Драйвер UART поддерживает функцию ioctl(), чтобы позволить HAL программам запрашивать специальные аппаратные операции. В табл. 6-2 определены запросы операций, поддерживаемые драйвером UART.

Табл. 6-2. Операции ioctl()

Запрос	Описание
TIOCEXCL	Закрывает устройство под эксклюзивный доступ. Дальнейший вызов open() для этого устройства не исполняется, поскольку этот файловый дескриптор закрыт, устройство открывается с использованием запроса TIOCNXCL ioctl. Этот запрос удаётся, поскольку для этого устройства не существует других файловых дескрипторов. Параметр arg игнорируется.
TIOCNXCL	Открывает закрытое под эксклюзивный доступ устройство. Параметр arg игнорируется.

Дополнительные запросы операций опционально доступны только для быстрого драйвера, как показано в табл. 6-3. Чтобы разрешить эти операции в вашей программе, вы должны установить предпроцессорную опцию -DALTERA_AVALON_UART_USE_IOCTL.

Табл. 6-3. Дополнительные операции UART ioctl() только для быстрого драйвера

Запрос	Описание
TIOCMGET	Возвращает текущую конфигурацию устройства, заполняя содержимым вход структуры termios(1). Указатель на эту структуру поддерживается как значение параметра opt.
TIOCMSET	Устанавливает конфигурацию устройства в соответствии со значением, хранящемся на входе структуры termios (1). Указатель на эту структуру поддерживается как значение параметра arg.

Примечание: (1) Структура termios определяется в стандартной библиотеке Newlib C. Вы можете найти эти определения в файле *<Nios II EDS install path>***/ components/altera_hal/HAL/inc/sys/termios.h**.

За подробной информацией о функции ioctl() обратитесь к Настольной книге программиста под Nios II.

Ограничения

Драйвер HAL для ядра UART не поддерживает регистр endofpacket. Обратитесь к секции "Карта регистров" за подробной информацией.

Программные файлы

Ядро UART сопровождается следующими программными файлами. Эти файлы определяют низкоуровневый интерфейс с устройством, и предлагают HAL драйверы устройств. Программисты не должны изменять эти файлы.

- **altera_avalon_uart_regs.h**— этот файл определяет карту регистров, предлагает символьные константы для аппаратного доступа к устройству. Символы этого файла используются в функциях драйвера.
- **altera_avalon_uart.h**, **altera_avalon_uart.c**— эти файлы реализуют драйвер устройства UART для системной библиотеки HAL.

Карта регистра

Программисты, использующие HAL API, никогда не имеют прямого доступа к ядру UART через его регистры. В общем, карта регистров полезна только для программистов, разрабатывающих драйвер для ядра.

Драйвер устройства HAL, поставляемый Altera, имеет прямой доступ к регистрам. Если вы пишете драйвер устройства, а HAL драйвер активен для этого же устройства, ваш драйвер будет конфликтовать и не будет работать.

В табл. 6-4 показана карта регистров ядра UART. Драйверы устройства контролируют и обмениваются с ядром посредством регистров с распределением в памяти.

Табл. 6-4. Карта регистров ядра UART

Офсет	Имя регистра	R/W	Описание/ биты регистра													
			15:13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata	RO	резервировано					*(1)*	*(1)*	принятые данные						
1	txdata	WO	резервировано					*(1)*	*(1)*	передаваемые данные						
2	status*(2)*	RW	резерв.	eop	cts	dcts	*(1)*	e	rrdy	trdy	tmt	toe	roe	brk	fe	pe
3	control	RW	резерв.	ieop	rts	idcts	trbk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe
4	divisor*(3)*	RW	делитель скорости обмена													
5	endofpacket *(3)*	RW	резервировано					*(1)*	*(1)*	значение окончания пакета						

Примечания к табл. 6-4:

(1) Этого бита может не быть, в зависимости от аппаратной опции **Data Width**. Если его нет, то чтение его – нуль, а запись не имеет значения.

(2) Запись нуля в регистр status сбрасывает биты dcts, e, toe, roe, brk, fe и pe.

(3) Этот регистр может быть или не быть, в зависимости от опций аппаратной конфигурации. Если его нет, то чтение приводит к непредвиденным значениям, а запись не имеет эффекта.

Некоторые регистры и биты опциональны. Эти регистры и биты существуют аппаратно, только если они разрешены на стадии генерирования системы. Опциональные регистры и биты помечены в следующих секциях.

Регистр rxdata

Регистр rxdata хранит данные принятые по входу RXD. Когда новый символ полностью принят по входу RXD, он перемещается в регистр rxdata, а бит регистра status – rrdy - устанавливается в 1. Бит регистра status – rrdy – устанавливается в 0, когда регистр rxdata прочитан. Когда символ перемещается в регистр rxdata при установленном бите rrdy (другими словами, предыдущий символ не забран из регистра), возникает ошибка переполнения приёмника, и бит регистра status – roe - устанавливается в 1. Новые символы всегда перемещаются в регистр rxdata независимо от того, прочитаны ли предыдущий символ. Запись данный в регистр rxdata не имеет эффекта.

Регистр txdata

Мастер периферия Avalon-MM записывает символы для отправки в регистр txdata. Символы не могут быть записаны в txdata, пока передатчик не будет готов для нового символа, это отображается в бите TRDY регистра status. Бит TRDY устанавливается в 0, когда символ записан в регистр txdata. Бит TRDY устанавливается в 1, когда символ передан из регистра txdata в передающий сдвиговый регистр. Если символ записывается в регистр txdata, когда бит TRDY = 0, результат не определён. Чтение регистра txdata возвращает неопределённое значение.

Например, представим, что логика передатчика бездействует, и мастер периферия Avalon-MM записывает первый символ в регистр txdata. Бит TRDY устанавливается в 0, затем устанавливается в 1, когда символ передан в передающий сдвиговый регистр. Мастер может записать второй символ в регистр txdata, а бит TRDY снова устанавливается в 0. Однако теперь сдвиговый регистр занят сдвигом первого символа в выход TXD. Бит TRDY не устанавливается в 1, пока первый символ полностью не сдвинется, а второй символ автоматически переместится в передающий сдвиговый регистр.

Регистр status

Регистр status состоит из отдельных битов, которые отображают соответствующие состояния внутри ядра UART. Каждый бит status ассоциирован с соответствующим битом разрешения прерывания регистра control. Регистр status может быть прочитан в любое время. Чтение не приводит к изменению значения любого из битов. Запись нуля в регистр status сбрасывает биты DCTS, E, TOE, ROE, BRK, FE и PE.

Биты регистра status показаны в табл. 6-5.

Табл. 6-5. Биты регистра status (часть 1 из 2)

Бит	Имя	Доступ	Описание
0(*1*)	PE	RC	Ошибка паритета. Ошибка паритета возникает, когда принятый бит паритета имеет несуществующий (некорректный) логический уровень. Бит PE устанавливается в 1, когда ядро передаёт символы с некорректным битом паритета. Бит PE остаётся в 1, пока он специально не сбрасывается записью в регистр status. Когда PE установлен, чтение из регистра rxdata приводит к неожиданному результату. Если аппаратная опция **Parity** не разрешена, нет проверки паритета, и бит PE всегда 0. Обратитесь к секции "Настройка Data Bits, Stop Bits, Parity" на стр. 6-5.
1	FE	RC	Ошибка кадрирования. Ошибка кадрирования происходит, когда приёмник не может определить корректный бит stop. Бит FE устанавливается в 1, когда ядро принимает символ с некорректным stop битом. Бит FE остаётся в 1, пока он специально не сбрасывается записью в регистр status. Когда FE установлен, чтение из регистра rxdata приводит к неожиданному результату.
2	BRK	RC	Детектор сбоя. Логика приёмника детектирует сбой, когда вывод RXD удерживает логический 0 время, большее, чем полная передача сивола (data+start+stop+parity биты). Когда сбой обнаружен, бит BRK устанавливается в 1. Бит BRK остаётся в 1, пока он специально не сбрасывается записью в регистр status.
3	ROE	RC	Ошибка переполнения приёмника. Ошибка переполнения приёмника возникает, когда новый принятый символ перемещён в регистр удержания rxdata, до того, пока предыдущий символ был прочитан (другими словами, пока бит RRDY в 1). В этом случае, бит ROE устанавливается в 1, а предыдущее содержимое регистра rxdata перезаписывается новым символом. Бит ROE остаётся в 1, пока он специально не сбрасывается записью в регистр status.

Табл. 6-5. Биты регистра status (часть 2 из 2)

Бит	Имя	Доступ	Описание
4	TOE	RC	Ошибка переполнения передатчика. Ошибка переполнения передатчика возникает, когда новый символ записывается в регистр удержания txdata прежде, чем предыдущий символ перемещается в сдвиговый регистр (другими словами, бит TRDY в 0). В этом случае, бит TOE устанавливается в 1. Бит TOE остаётся в 1, пока он специально не сбрасывается записью в регистр status.
5	TMT	R	Передатчик пуст. Бит TMT показывает текущее состояние сдвигового регистра передатчика. Когда сдвиговый регистр в процессе сдвига символа на вывод TXD, TMT установлен в 0. Когда сдвиговый регистр бездействует (другими словами, символ не передаётся) бит TMT в 1. Мастер периферия Avalon-MM может определять, когда отправка завершена (и принята на другом конце последовательной связи), проверяя бит TMT.
6	TRDY	R	Отправка готова. Бит TRDY отображает текущее состояние регистра удержания txdata. Когда регистр txdata пуст, он готов для нового символа, и TRDY в 1. Когда регистр txdata полон, TRDY в 0. Мастер периферия Avalon-MM должна ждать, пока TRDY перейдёт в 1, прежде чем записывать новые данные.
7	RRDY	R	Принятый символ готов. Бит RRDY отображает текущее состояние регистра удержания rxdata. Когда регистр rxdata пуст, он готов для нового символа, и RRDY в 0. Когда новое принятое значение перемещается в регистр rxdata, RRDY устанавливается в 1. Чтение регистра rxdata сбрасывает бит RRDY в 0. Мастер периферия Avalon-MM должна ждать, пока RRDY перейдёт в 1, прежде чем читать регистр rxdata.
8	E	RC	Исключение. Бит E отображает, когда происходит исключительное событие. Бит E является "логическим или" битов TOE, ROE, BRK, FE и PE. Бит E и соответствующий бит разрешения прерывания (IE) в регистре control предлагают удобный способ разрешения/запрещения IRQ для всех состояний ошибок. Бит E устанавливается в путём записи в регистр status.
10*(1)*	DCTS	RC	Сигнал изменения готовности к приёму (CTS). Бит DCTS устанавливается в 1, когда детектируется логический переход во входном порте CTS_N (защёлкивается синхронно с тактом Avalon-MM). Этот бит устанавливается по растущему и спадающему переходу CTS_N. Бит DCTS остаётся в 1, пока он специально не сбрасывается записью в регистр status. Если аппаратная опция **Flow Control** не разрешена, бит DCTS всегда 0. Обратитесь к секции "Настройка Flow Control" на стр. 6-5.
11*(1)*	CTS	R	Сигнал готовности к приёму (CTS). Бит CTS связан с мгновенным состоянием входа CTS_N (защёлкивается синхронно с тактом Avalon-MM). Вход CTS_N не влияет на процесс передачи или приёма. Единственным видимым эффектом входа CTS_N являются состояния битов CTS и DCTS, а IRQ может генерироваться, когда разрешён бит idcts регистра control. Если аппаратная опция **Flow Control** не разрешена, бит CTS всегда 0. Обратитесь к секции "Настройка Flow Control" на стр. 6-5.
12*(1)*	EOP	R	Вычисление окончания пакета. Бит EOP устанавливается в 1 в одном из следующих случаев: ▪ Символ EOP был записан в txdata ▪ Символ EOP был прочитан из rxdata Символ EOP определяется содержимым регистра endofpacket. Бит EOP остаётся в 1, пока он специально не сбрасывается записью в регистр status. Если опция **Include End-of-Packet Register** не разрешена, бит EOP всегда 0. Обратитесь к секции "Опция Streaming Data (DMA) Control" на стр. 6-6.

Примечание к табл. 6-5:
(1) Это опция, она может быть не реализована аппаратно.

Регистр control

Регистр control состоит из отдельных битов, каждый из которых контролирует определённый аспект работы ядра UART. Значение в регистре control может быть прочитано в любое время. Каждый бит в регистре control разрешает IRQ для соответствующего бита регистра status. Когда оба бита – status и соответствующий ему бит разрешения прерывания равны 1 – ядро генерирует IRQ.

Биты регистра control представлены в табл. 6-6.

Табл. 6-6. Биты регистра control

Бит	Имя	Доступ	Описание
0	IPE	RW	Разрешение прерывания при ошибке паритета.
1	IFE	RW	Разрешение прерывания при ошибке кадрирования.
2	IBRK	RW	Разрешение прерывания при детектировании сбоя
3	IROE	RW	Разрешение прерывания при ошибке переполнения приёмника.
4	ITOE	RW	Разрешение прерывания при ошибке переполнения передатчика.
5	ITMT	RW	Разрешение прерывания при опустошении сдвигового регистра передатчика.
6	ITRDY	RW	Разрешение прерывания при готовности передатчика.
7	IRRDY	RW	Разрешение прерывания при готовности чтения.
8	IE	RW	Разрешение прерывания при исключении.
9	TRBK	RW	Сбой передачи. Бит TRBK позволяет мастер периферии Avalon-MM передавать символ сбоя через выход TXD. Сигнал TXD форсировано переключается в 0, когда бит TRBK установлен в 1. Бит TRBK изменяет любые логические уровни, которые логика передатчика сообщает на выход TXD. Бит TRBK вмешивается в любой процесс отправки. Мастер периферия Avalon-MM должна установить бит TRBK обратно в 0, после завершения соответствующего периода сбоя.
10(1)	IDCTS	RW	Разрешение прерывания при изменении сигнала CTS.
11	RTS	RW	Сигнал запроса передачи (RTS). Бит RTS прямо поступает на выход RTS_N. Мастер периферия Avalon-MM может записывать в бит RTS в любое время. Значение бита RTS влияет только на выход RTS_N, это не влияет на логику передатчика или приёмника. Поскольку логика выхода RTS_N негативная, когда бит RTS в 1, логический уровень 0 подаётся на выход RTS_N. Если аппаратная опция **Flow Control** не разрешена, бит RTS всегда 0. Обратитесь к секции "Настройка Flow Control" на стр. 6-5.
12	IEOP	RW	Разрешение прерывания для состояния окончания пакета.

Примечание к табл. 6-6:
(1) Это опция, она может быть не реализована аппаратно.

Регистр divisor (опционально)

Значение в регистре divisor используется для генерирования тактовой частоты скорости обмена. Действительная скорость обмена определяется по формуле:

Скорость обмена = (Тактовая частота) / (divisor + 1)

Регистр divisor опциональное аппаратное средство. Если аппаратная опция **Baud Rate Can Be Changed By Software** не разрешена, регистр divisor не существует. В этом случае, запись в регистр не имеет эффекта, а чтение регистра divisor приводит к непредвиденным результатам.

За дополнительной информацией обратитесь к секции "Опция Baud Rate" на стр. 6-4.

Регистр endofpacket (опционально)

Значение в регистре endofpacket определяет символ окончания посылки для DMA транзакций переменной длительности. После сброса, значение по умолчанию – нуль, которое является ASCII символом null (\0). За дополнительной информацией обратитесь к табл. 6-5 на стр. 6-12 за описанием бита EOP.

Регистр endofpacket опциональное аппаратное средство. Если аппаратная опция **Include end-of-packet register** не разрешена, регистр endofpacket не существует. В этом случае, запись в регистр endofpacket не имеет эффекта, а чтение регистра приводит к непредвиденным результатам.

Поведение прерывания

Ядро UART инициирует один сигнал IRQ для интерфейса Avalon-MM, который может быть подключен к любой мастер периферии системы, например к процессору Nios II. Мастер периферия должна читать регистр status для определения источника прерывания.

Каждое состояние прерывания имеет ассоциированный бит в регистре status и бит разрешения прерывания в регистре control. Когда происходит одно из событий прерываний, ассоциированный бит status устанавливается в 1, и остаётся в этом состоянии, пока не будет явно опознан. Выход IRQ назначается, когда любой из битов status установлен, притом, что соответствующий бит разрешения прерывания установлен в 1. Мастер периферия признаёт прерывание, сбрасывая регистр status.

После сброса, все биты разрешения прерывания установлены в 0; поэтому ядро не может вызывать IRQ, пока мастер периферия не установит один или несколько битов разрешения прерывания в 1.

Все возможные состояния прерываний перечислены в ассоциированных битах status и control (разрешения прерывания) в табл. 6-5 на стр. 6-16 и в табл. 6-6 на стр. 6-18. Подробное описание каждого состояния прерывания предоставляется в описании битов status.