Nios UART

January 2003, Version 3.0 Data Sheet

General Description

The Nios® UART module is an Altera® SOPC Builder library component included in the Nios development kit. The UART module is a common serial interface with variable baud rate, parity, stop and data bits, and optional control signals. The SOPC Builder UART library component has available system choices to define device logic and interface signals on the Nios development kit. The UART's Verilog HDL or VHDL source code is available for development and includes the necessary software subroutines for easy system integration.

Functional Description

The Nios UART implements simple RS-232 asynchronous transmit and receive logic inside an Altera device. The UART sends and receives serial data over two external pins (`RxD` and `TxD`). Software controls and communicates with the UART through five memory-mapped, 16-bit registers.

To comply with RS-232 voltage-signaling specifications, an external level shifting buffer is required (e.g., Linear Technology LTC-1386) between the `TxD/RxD` I/O pins and the corresponding RS-232 external connections. The acceptable input voltage range and the output voltage level depends on how the Altera device's I/O pins are configured. The UART uses a logic 0 for mark, and a logic 1 for space. The UART runs on a single synchronous clock input, `clk`.

The UART peripheral can be used in conjunction with the DMA peripheral to allow streaming data transfers between the UART and memory. See the *Nios DMA Data Sheet* for details.

You can customize the UART peripheral specifically for simulation. For example, during high-speed system simulation (e.g., 100 MHz), UART simulation can be painstakingly slow because the UART transmits approximately 11,500 characters per second at 115,200 bps. At this baud rate, an 11-bit character (8 bits, 1 start-bit, 2 stop bits) takes greater than 9,500 clock cycles to simulate at 100 MHz. To speed functional simulation, you can run the UART with a small baud divisor, which allows the UART to run at half the system clock speed. In this mode, one bit is transmitted every two clock cycles, or roughly one character per 10 clock cycles.

Additionally, you can customize the data stream transmitted to the UART, which is useful for simulating operation of an application that normally requires a user to type text that is sent to the Nios application (e.g., via the GERMS monitor).

You specify how data streams transmit to the UART in one of the following ways:

- Using the **Simulated RXD-Input Character Stream** dialog box
- Using the optional **Interactive** windows in the ModelSim software

See *AN 189: Simulating Nios Embedded Processor Designs* for details on the Nios development kit simulation flow and process using an example Nios design and the ModelSim simulator.

Transmitter Logic

The UART transmitter consists of a 7-, 8-, or 9-bit `txdata` holding register and a 7-, 8-, or 9-bit transmit shift register (the number of data bits is determined by the `data_bits` PTF assignment). The `txdata` holding register is directly written by software. The transmit shift register is automatically loaded from the `txdata` register when a serial transmit shift operation is not currently in process. The transmit shift register directly feeds the `TxD` data pin. Data is shifted out to `TxD` LSB first.

These two registers provide double buffering; software can write a new value into the `txdata` register while the previously written character is being shifted out. Software can monitor the transmitter's status by reading the `status` register's transmitter ready (`trdy`), transmitter shift register empty (`tmt`), and transmitter overrun error (`toe`) bits.

The transmitter logic automatically inserts the correct number of start, stop, and parity bits in the serial `TxD` data stream as required by the RS-232 specification and determined by the PTF assignments.

Receiver Logic

The UART receiver consists of a 7-, 8-, or 9-bit receiver-shift register and a 7-, 8-, or 9-bit `rxdata` holding register (the number of data bits is determined by the `data_bits` PTF assignment). The `rxdata` holding register can be read directly by software. The `rxdata` holding register is loaded from the receiver shift register automatically every time a new character is fully received.

These two registers provide double buffering. The rxdata register can hold a previously received character while the subsequent character is being shifted into the receiver shift register.

Software can monitor the receiver's status by reading the status register's read-ready (rrdy), receiver-overrun error (roe), break detect (brk), parity error (pe), and framing error (fe) bits. The receiver logic automatically detects the correct number of start, stop, and parity bits in the serial RxD stream as required by the RS-232 specification, and as determined by the hardware compile time UART configuration. The receiver logic checks for four exceptional conditions in the received data, and sets corresponding status register bits (fe, pe, roe, or brk).

Baud Rate Generation

The UART's internal baud clock is derived from the UART's master clock input, which is the same as the Avalon-bus system clock. The internal baud clock is generated by a clock divider. The divisor value can come from one of the following sources:

- A constant value set by the UART's baud PTF assignment and the system's clock_freq PTF assignment
- The host-settable 16-bit value in the divisor register

The UART uses a host-settable baud rate divisor register when the fixed_baud PTF parameter is set to 0. The UART uses a fixed baud rate when fixed_baud is set to 1.

UART Registers

Table 1 lists and describes the UART registers.

Table 1. UART Register Map

A2..A0	Register Name	R/W	Description/Register Bits															
			15	. . .	12	11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata	RO								RxData								
1	txdata	WO								TxData								
2	status *(1)*	RW			eop	cts	dcts	–	e *(2)*	rrdy	trdy	tmt	toe	roe	brk	fe	pe
3	control	RW			ieop	rts	idcts	trbk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe
4	divisor	RW	Baud Rate Divisor (optional)															
5	endofpacket	RW								End-packet value								

Notes to Table 1:
(1) A write operation to the status register clears the dcts, e, toe, roe, brk, fe, and pe bits.
(2) status register bit 8 (e) is the logical OR of the toe, roe, brk, fe, and pe bits.

rxdata Register

Software reads received characters from the rxdata register. When a new character is fully received via the RxD input, it is transferred into the rxdata register, and the status register's rrdy bit is set to 1. When software reads a value from the rxdata register, the status register's rrdy bit is set to 0. If a character is transferred into the rxdata register when the rrdy bit is set (that is, software has not retrieved the previous character), a receiver-overrun error occurs and the status register's roe bit is set to 1. New characters are always transferred into the rxdata register, whether or not software retrieved the previous character. Writing data to the rxdata register has no effect.

txdata Register

Software writes characters to be transmitted directly into the txdata register. Characters should not be written to the txdata register until the transmitter is ready for a new character, as indicated by the trdy bit in the status register. If a character is written to the txdata register when trdy is 0, the results are undefined. The trdy bit is set to 0 when software writes a character into the txdata register. The trdy bit is set to 1 when a character is transferred from the txdata register into the transmitter shift register.

For example, assume the UART is idle and software writes a first character into the txdata register. The trdy bit is set to 0, then set to 1 when the character is transferred into the transmitter shift register. Software can then write a second character into the txdata register, and the trdy bit is set to 0 again. However, this time the first character still occupies the transmitter shift register and is still in the slower process of being transmitted over the TxD output pin. The trdy bit is not set to 1 until the first character is fully shifted out and the second character is automatically transferred into the transmitter shift register. Reading data from the txdata register produces an undefined result.

status Register

The status register consists of individual bits that indicate particular conditions inside the UART. The status register can be read at any time by software. Reading the status register does not change the value of any of the bits. Each status bit is associated with a corresponding interrupt-enable bit in the control register. When a status bit's corresponding interrupt-enable equals 1, a true (1) condition on that status bit causes an interrupt request to be sent to software.

Most status bits are set to 0 when software performs a write operation to the status register (the written data value is ignored). See the bit descriptions for the bits that are not effected.

The status register bits are shown in Table 2:

Table 2. status Register Bits

Bit Number	Bit Name	Description
0	pe	Parity error
1	fe	Framing error
2	brk	Break detect
3	roe	Receive overrun error
4	toe	Transmit overrun error
5	tmt	Transmit empty
6	trdy	Transmit ready
7	rrdy	Receive char ready
8	e	Exception
10	dcts	Change in clear to send (CTS) signal
11	cts	CTS signal
12	eop	End of packet encountered

pe Bit

A parity error occurs when the received parity bit has an unexpected (incorrect) logic level.

The pe bit is set to 1 when the UART receives a character with an incorrect parity bit. When the PTF parameter parity is set to "N", no parity checking is performed and the pe bit is always 0 (see "parity" on page 19). The pe bit is persistent—it stays set to 1 until it is explicitly cleared by a software write operation to the status register.

fe Bit

A framing error occurs when the receiver fails to detect a correct stop bit.The fe bit is set to 1 when the UART receives a character with an incorrect stop bit. The fe bit is persistent—it stays set to 1 until it is explicitly cleared by a software write operation to the status register.

When the pe or fe bit is set, reading from the rxdata register produces an undefined result.

brk Bit

The receiver logic detects a break when the RxD pin is held low (logic 0) continuously for longer than a full-character time (7, 8, or 9 bit-cycles, plus start, stop, and parity bits). When a break is detected, the brk bit is set to 1. The brk bit is persistent—it stays set to 1 until it is explicitly cleared by a software write operation to the status register.

roe Bit

A receiver-overrun error occurs when a newly received character is transferred into the rxdata holding register before the previous character is read by software (that is, while the rrdy bit is 1). In this case, the roe bit is set to 1, and the rxdata register's previous contents are overwritten with the newly received character. The roe bit is persistent—it stays set to 1 until it is explicitly cleared by a software write operation to the status register.

toe Bit

The toe bit is set to 1 when software writes a new character into the txdata holding register while the trdy bit is 0 (that is, before the previous character is transferred into the transmitter shift register). The toe bit is persistent—it stays set to 1 until it is explicitly cleared by a software write operation to the status register.

tmt Bit

The tmt bit indicates the transmitter shift register's current state. When the transmitter shift register is in the process of shifting a character out the TxD pin, tmt is set to 0. When the transmitter shift register is idle (that is, a character is not being transmitted) the tmt bit is 1. Software can determine if a transmission is completed (and should be received at the other end of a serial link) by checking the tmt bit.

The tmt bit is not changed by a write operation to the status register.

trdy Bit

The trdy bit indicates the txdata holding register's current state. When the txdata holding register is empty (that is, its contents are transferred to the transmitter shift register), it is ready for a new character and trdy is 1. When the value in the txdata register is not transferred into the transmitter shift register (because the transmitter shift register is busy shifting out the previous character), trdy is 0. Software must always wait for trdy to equal 1 before writing a new character into the txdata register.

The trdy bit is not changed by a write operation to the status register.

rrdy Bit

The rrdy bit indicates the rxdata holding register's current state. When the rxdata holding register is empty (that is, the UART has not received any new characters), it is not ready to be read and rrdy is 0. When a newly received value is transferred into the rxdata register, rrdy is set to 1. The rrdy bit is set to 0 when software performs a read operation on the rxdata register. Software must always wait for rrdy to equal 1 before reading a character from the rxdata register.

The rrdy bit is not changed by a write operation to the status register.

e Bit

The exception condition e bit is a simple logical-OR of the status register's toe, roe, brk, fe, and pe bits. The e bit indicates to software that an exception condition—other than the normal character transactions—is occurring. The e bit and corresponding interrupt-enable exception condition (ie) bit also provide a convenient method to enable/disable interrupts for all error conditions.

The e bit is set to 0 by a write operation to the status register because all of its constituent bits are set to 0 by this action.

dcts Bit

When the PTF parameter use_cts_rts = 1, the status register includes the dcts bit (see "use_cts_rts" on page 18). This bit is set to 1 whenever a logic-level transition (that is, an edge) is detected on the synchronously-sampled cts_n input pin. This bit is set by both falling (1 to 0) and rising (0 to 1) transitions on the cts_n input. The dcts bit is persistent—it stays set to 1 until it is explicitly cleared when software performs a write operation to the status register.

If the control register's idcts interrupt-enable bit is 1, the UART generates an interrupt when dcts is 1.

When the PTF parameter use_cts_rts = 0, the dcts bit is always 0.

cts Bit

When the PTF parameter use_cts_rts = 1, the status register includes the read-only cts bit (see "use_cts_rts" on page 18). This bit reflects the CTS input signal's instantaneous, synchronously-sampled logic state. The UART hardware has a logic-negative input pin, cts_n. Thus, cts = 1 when a 0 logic-level is applied to the external cts_n input, and cts = 0 when a 1 logic-level is applied to the external cts_n input.

When the PTF parameter use_cts_rts = 0, the cts bit is always 0.

The cts_n input has no effect on the UART's transmit or receive logic. The only visible effect of the cts_n input is the state of the cts and dcts bits, and a potential interrupt that can be generated when the control register's idcts bit is enabled.

eop Bit

When the PTF parameter use_eop_register = 1, the status register includes the eop bit (see "use_eop_register" on page 18). This bit is set to 1 by one of the following events:

- Software writes an EOP character to the transmitter
- Software reads an EOP character from the receiver

Specifically, the eop bit is set to 1 when software executes a write transaction to the txdata register and the data value written to txdata is the same as the endofpacket register's current value. This bit is also set to 1 when software executes a read transaction from the rxdata register and the data value read by software is the same as the endofpacket register's current value. The eop bit is persistent—it stays set to 1 until it is explicitly cleared when software performs a write operation to the status register.

When the PTF parameter use_eop_register = 0, the eop bit always reads as 0.

control Register

The control register is composed of individual bits, each controlling the UART's internal operation. Each bit in the control register enables an interrupt bit for the corresponding bit in the status register. The value in the control register can be read at any time by software.

The control register bits are shown in Table 3:

Table 3. control Register Bits

Bit Number	Bit Name	Description
0	ipe	Enable interrupt for a parity error
1	ife	Enable interrupt for a framing error
2	ibrk	Enable interrupt for a break detect
3	iroe	Enable interrupt for a receiver overrun error
4	itoe	Enable interrupt for a transmitter overrun error
5	itmt	Enable interrupt for a transmitter shift register empty
6	itrdy	Enable interrupt for a transmission ready
7	irrdy	Enable interrupt for a read ready
8	ie	Enable interrupt for an exception
9	trbk	Transmit break
10	idcts	Enable interrupt for a change in CTS signal
11	rts	Request to send (RTS) signal
12	ieop	Enable interrupt for an end of packet encountered

The control register bits allow software to determine which, if any, internal conditions of the UART result in an interrupt request to software.

Each bit in the status register has a corresponding interrupt-enable bit at the same bit position in the control register. For example, the pe bit is bit 0 of the status register, and the corresponding ipe bit is bit 0 of the control register. For each status register bit, an interrupt request is generated when both the status bit and its corresponding interrupt-enable bit both equal 1.

trbk Bit

The transmit break (trbk) bit, allows software to transmit a break character over the UART's TxD pin under software control. The TxD pin is set to 0 when the trbk bit equals 1. The trbk bit overrides any normal logic level that the transmitter logic may have otherwise driven on the TxD pin. The trbk bit interferes with any transmission in process. The software must set the trbk bit back to 0 after an appropriate break period elapses.

rts Bit

When the PTF parameter use_cts_rts = 1, the control register includes the rts bit (see "use_cts_rts" on page 18). This readable and writable bit directly feeds the logic-negative rts_n output pin. Software can write rts at any time. The value of rts has no effect on the UART's transmit or receive hardware. The rts bit's only purpose is to determine the rts_n output pin's value. When rts is 1, a low logic-level (0) is driven on the rts_n output pin. When rts is 0, a high logic-level (1) is driven on the rts_n output pin.

When the PTF parameter use_rts_cts = 0, the rts bit always reads as 0 and writing to rts has no effect.

divisor Register

The divisor register is only implemented when the PTF parameter fixed_baud is set to 0 (see "fixed_baud" on page 17). When fixed_baud is set to 1, the divisor register does not exist—the write operations to the divisor register have no effect, and the result of a read operation from the divisor register is undefined.

When fixed_baud is set to 0, the divisor register's contents are used to generate the UART's baud rate clock. The UART's final baud rate is computed by the formula:

$$\text{baud rate} = \frac{\texttt{clock\_freq}}{(\texttt{divisor} + 1)}$$

Software can read back the value in the divisor register at any time.

endofpacket Register

The end of packet character is determined by the value of the endofpacket register. The default value is zero. For more information, see "eop Bit" on page 8.

Interrupt Outputs

The UART produces a single IRQ output signal as part of its Avalon bus interface. The UART asserts its interrupt-request output when one or more internal conditions occurs and the condition's corresponding interrupt-enable bit in the `control` register is also set. At reset, all interrupt-enable bits are set to 0; therefore, the UART cannot assert its interrupt-request output until one or more of its interrupt-enable bits are set to 1 by software.

Each possible interrupt condition has an associated bit in the `status` register and an associated interrupt-enable bit in the `control` register. When any of the interrupt conditions occurs, the associated `status` bit is set to 1 and remains set until the `status` register is cleared by software. Software clears the `status` register by writing it with any value (the value is ignored).

All possible interrupt conditions are listed with their associated `status` and `control` (interrupt-enable) bits in Table 2 and Table 3. Details of each condition are provided in the `status` bit descriptions. The IRQ output is asserted when any of these `status` bits are set while the corresponding interrupt-enable bit is 1.

Software Data Structure

Below is the UART software data structure.

```
typedef volatile struct{
  int np_uartrxdata;       // Read-only, 8-bit
  int np_uarttxdata;       // Write-only, 8-bit
  int np_uartstatus;       // Read-only, 9-bit
  int np_uartcontrol;      // Read/Write, 9-bit
  int np_uartdivisor;      // Read/Write, 16-bit, optional
  int np_uartendofpacket;  // Read/Write, end of packet character
} np_uart;
```

Software Subroutines

Table 4 lists the UART software subroutines available in the Nios library (**lib** folder in the custom software development kit (SDK) when one or more UART peripherals are present in the Nios system. These functions are declared in the include file **nios.h**.

Table 4. UART Software Subroutines

Subroutine	Description
nr_uart_rxchar	Reads a character from the UART whose address is passed as an argument.
nr_uart_txcr	Sends a carriage return and line feed to the UART at address `nasys_printf_UART`.
nr_uart_txchar	Sends a single character to the UART whose address is passed as an argument.
nr_uart_txhex	Prints an integer value, in hexadecimal, to the UART at address `nasys_printf_UART`.
nr_uart_txhex16	Prints the value of a short integer, in hexadecimal, to the UART at address `nasys_printf_UART`.
nr_uart_txhex32	Prints the value of a long integer, in hexadecimal, to the UART at address `nasys_printf_UART`.
nr_uart_txstring	Prints a null-terminated string to the UART at address `nasys_printf_UART`.

nr_uart_rxchar

This subroutine reads a character from the UART peripheral whose address is passed in uartBase. If no character is waiting, nr_uart_rxchar returns -1. If zero is passed for the peripheral address, nr_uart_rxchar reads a character from the UART at location `nasys_printf_uart` (**nios.h**).

Syntax

```
int nr_uart_rxchar(np_uart *uartBase);
```

Parameter

The `uartBase` parameter is a pointer to the UART peripheral.

Figure 1 shows an example of nr_uart_rxchar subroutine.

Figure 1. nr_uart_rxchar Example

```
#include "nios.h"

void main(void)
{
   int c;

   printf("Please enter a character:\n");

   while((c = nr_uart_rxchar(nasys_printf_UART)) == -1)
      ; // wait for valid input

   printf("Your character is:\t%c\n", c);
}
```

nr_uart_txchar

This subroutine sends a single character, c, to the UART peripheral whose address is passed as uartBase. If zero is passed for the peripheral address, nr_uart_txchar sends a character to the UART at location nasys_printf_uart (defined in **nios.h**).

Syntax

```
int nr_uart_txchar(int c, np_uart *uartBase);
```

Parameters

Parameter Name	Description
c	Character to be sent.
uartBase	Pointer to the UART peripheral.

Figure 2 shows an example of the nr_uart_txchar subroutine.

Figure 2. nr_uart_txchar Example

```
#include "nios.h"

#define kLineWidth 77
#define kLineCount 100

void SendLots(void)
{
    char c;
    int i,j;
    int mix;

    printf("\n\nPress character, or <space> for mix: ");
    while((c = nr_rxchar(0)) < 0);

    printf("%c\n\n",c);

    // Don't show unprintables

    if(c < 32)
        c = '.';

    mix = c==' ';

    for(i = 0; i < kLineCount; i++)
    {
        for(j = 0; j < kLineWidth; j++)
        {
            if(mix)
            {
                c++;
                if(c >= 127)
                   c = 33;
            }
            nr_uart_txchar(c,nasys_printf_UART);
            // send character to UART
        }
        nr_uart_txcr();
        // send carriage return and new line

    }
    printf("\n\n");
}
```

nr_uart_txcr

This subroutine sends a carriage return and line feed to the UART at location nasys_printf_uart (defined in **nios.h**).

Syntax

```
int nr_uart_txcr(void);
```

nr_uart_txhex

This subroutine prints the integer value of x in hexadecimal to the UART at location nasys_printf_uart (defined in **nios.h**). The range for a 16-bit Nios CPU is 0000 to FFFF, and for a 32-bit Nios CPU is 00000000 to FFFFFFFF.

Syntax

```
int nr_uart_txhex(int x);
```

Parameter

The x parameter is an integer value to be sent to UART.

nr_uart_txhex16

This subroutine prints the 16-bit value of x in hexadecimal to the UART at location nasys_printf_uart (defined in **nios.h**). The range is from 0000 to FFFF.

Syntax

```
int nr_uart_txhex16(short x);
```

Parameter

The x parameter is a 16-bit integer value to be sent to UART.

nr_uart_txhex32

This subroutine prints the 32-bit value of x in hexadecimal to the UART at location nasys_printf_uart (defined in **nios.h**). The range is from 00000000 to FFFFFFF. This subroutine is not available on a 16-bit Nios CPU.

Syntax

```
int nr_uart_txhex32(long x);
```

Parameter

The x parameter is a 32-bit integer value to be sent to UART.

nr_uart_txstring

This subroutine prints the null-terminated string *s* to the UART at location nasys_printf_uart (defined in **nios.h**).

Syntax

```
int nr_uart_txstring(char *s);
```

Parameter

The `s` parameter is a pointer to a null-terminated character string.

PTF Assignments

Table 5 lists the UART's PTF parameters followed by a description of each parameter.

Table 5. UART PTF File Parameters

Parameter	Section *(1)*	Type	Allowed Values	Default	Units
clock_freq	S/WSA	Integer	≥ 1	33333000	Hz
fixed_baud	M/WSA	Boolean	1, 0	1	–
use_cts_rts	M/WSA	Boolean	1, 0	0	–
use_eop_register	M/WSA	Boolean	1, 0	0	–
data_bits	M/WSA	Integer	7, 8, 9	8	bits
stop_bits	M/WSA	Integer	1, 2	1	bits
parity	M/WSA	String	"N", "E", "O"	"N"	–
baud	M/WSA	Integer	clock_freq/65536 to clock_freq/2	115200	bps
sim_true_baud	M/WSA	Boolean	1, 0	0	–
sim_char_stream	M/WSA	String	any printable alphanumeric character, plus "\n" (newline), "\r" (carriage return), and "\"" (double-quote character)	–	–

Note to Table 5:
(1) The **Section** column describes the parameter's location in the PTF:
S/WSA = `SYSTEM/WIZARD_SCRIPT_ARGUMENTS`
M/WSA = `MODULE/WIZARD_SCRIPT_ARGUMENTS`

clock_freq

The `clock_freq` assignment is the global system clock frequency. This setting is derived from the system PTF, and is not set by the wizard.

fixed_baud

When `fixed_baud` is set to 1, the UART is implemented using a constant (unchangeable) baud divisor. Thus, UARTs constructed with `fixed_baud` set to 1 always run at the same baud rate, given by the `baud` assignment (see "baud" on page 20). When `fixed_baud` is set to 1, the hard coded baud divisor value is computed according to the formula:

$$\texttt{divisor} = \text{int}\left(\frac{\texttt{clock\_freq}}{\texttt{baud}} + 0.5\right)$$

The `clock_freq` assignment determines the clock frequency. When `fixed_baud` is set to 1, software cannot change the UART's baud rate, and the UART does not implement a `divisor` register at address offset 4. When `fixed_baud` is set to 1, writing to address offset 4 has no effect, and reading from address offset 4 produces an undefined result.

When `fixed_baud` is set to 0, the UART includes a 16-bit `divisor` register at address offset 4 (see "divisor Register" on page 10). The `divisor` register's value determines the baud rate according to the formula:

$$\text{baud rate} = \frac{\texttt{clock\_freq}}{(\texttt{divisor} + 1)}$$

Software can write the `divisor` register to any 16-bit value, which is treated as an unsigned integer. At reset, the `divisor` register is initialized to a value that depends on the `baud` assignment, according to the formula:

$$\text{divisor reset value} = \text{int}\left(\frac{\texttt{clock\_freq}}{\texttt{baud}} + 0.5\right)$$

Thus, when `fixed_baud` is set to 0, the `baud` assignment determines the UART's baud rate, until and unless software writes a different value into the `divisor` register.

use_cts_rts

When use_cts_rts is set to 1, the UART includes:

- A cts_n (logic-negative CTS) input pin
- An rts_n (logic-negative RTS) output pin
- A cts bit in the status register
- A dcts bit in the status register
- An rts bit in the control register
- An idcts bit in the control register

When use_cts_rts is set to 1, software can detect CTS and transmit RTS flow control signals. The cts-input and rts-output pins are entirely under the control of software, and have no direct effect on any other part of the UART hardware, other than the associated control and status bits.

When use_cts_rts is set to 0, the UART does not include cts_n and rts_n pins, and the control/status bits cts, dcts, idcts, and rts are not implemented (always read as 0).

use_eop_register

When use_eop_register is set to 1, the UART includes:

- A 7-, 8-, or 9-bit endofpacket register at address-offset 5 (size given by data_bits assignment)
- An eop bit in the status register
- An ieop bit in the control register
- An end of packet signal on its Avalon bus interface to be used in conjunction with streaming data transfers

When use_eop_register is set to 1, the UART can be programmed to automatically terminate streaming data transactions when used with a streaming-capable Avalon master (such as a DMA controller).

The eop detection feature can be used with a DMA, for example, to implement a UART that automatically fills a buffer until a specified character is encountered in the incoming rxdata or txdata stream. The terminating (end of packet) character's value is taken from the endofpacket register.

When use_eop_register is set to 0, the UART does not include:

- endofpacket register
- end of packet Avalon interface signal
- eop bit in the status register
- ieop bit in the control register

Also, when use_eop_register is set to 0, writing to address offset 5 has no effect, and reading from address offset 5 produces an undefined result.

data_bits

The UART can be constructed to transmit and receive 7-, 8-, or 9-bit data values, as determined by the value of data_bits. The txdata, rxdata, and endofpacket register widths are all determined by the data_bits assignment.

stop_bits

The UART can be constructed to transmit either 1 or 2 stop bits with every character, as determined by the stop_bits assignment. The UART always terminates a receive transaction at the first stop bit, and ignores all subsequent stop bits, regardless of this parameter's value.

parity

When parity is set to N, the UART transmit-logic sends data without including a parity bit, and the UART receive-logic presumes the incoming data does not include a parity bit. When parity is set to N, the status register's pe bit is not implemented (always reads 0).

For the other parity values, the UART transmit-logic computes and inserts the required parity bit into the outgoing TxD bitstream, and the UART receive-logic checks the incoming parity bit in the RxD bitstream. If the receiver finds data with incorrect parity, the status register's pe bit is checked and an interrupt is asserted when the corresponding mask bit in the control register (ipe) is enabled.

The outgoing parity bit inserted into the TxD stream and the received parity bit's expected correct value is determined by the parity assignment:

parity Assignment	Description
E	Parity bit is set to 1 if txdata has an even number of 1-bits, otherwise parity bit is set to 0
O	Parity bit is set to 1 if txdata has an odd number of 1-bits, otherwise parity bit is set to 0

baud

When fixed_baud is set to 1, the baud assignment determines the UART's baud-rate. When fixed_baud is set to 0, the baud assignment determines the divisor register's initial (reset) value. The formulas for computing the resultant divisor constant are described in "fixed_baud" on page 17.

sim_true_baud

When the UART logic is generated, a simulation model is also constructed. When sim_true_baud is set to 1, the UART simulation faithfully models the transmit- and receive baud divisor logic. This often leads to long simulation runtimes, since serial transmission rates are often slower than any other process in the system. Accurately modeling the UART's divisor logic is time-consuming and seldom useful.

When sim_true_baud is set to 0, the simulated UART's baud divisor is overridden by the fixed value 4. This assignment only affects the simulation model—the generated UART logic as such is not changed. The smaller baud divisor tends to accelerate simulation runtimes with little loss of authenticity in the simulated system results. When fixed_baud is set to 0 and sim_true_baud is set to 0, the software-writable divisor register is included in the simulation model, but its initial (reset) value is 4 instead of the actual (non-simulation) value determined by the baud assignment.

sim_char_stream

The sim_char_stream value consists of simulation information entered in the wizard. Based on this stream, the SOPC Builder generates a simulation vector file that inputs the sim_char_stream into the UART. By default, sim_char_stream contains no characters.

See the *SOPC Builder PTF File Reference Manual* for comprehensive information about the internal workings of the SOPC Builder tool including details on the PTF file structure and parameters.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com