



1 Core Overview

The IrDA UART Core implements a method for communication of serial data. The core provides a simple register-mapped Avalon[®] interface. Master peripherals (such as a Nios[®] II processor) communicate with the core by reading and writing control and data registers.

2 Instantiating the Core in SOPC Builder


Designers use the IrDA UART Core's configuration wizard in the SOPC Builder to specify the required features. The following section describes the available options in the configuration wizard.

2.1 Configuration Settings

This section describes the configuration settings.

2.1.1 Baud Rate Options

The IrDA UART Core can implement any of the standard baud rates for RS-232 connections. The baud rate is fixed at system generation time and cannot be changed via the Avalon slave port.

 The baud rate is calculated based on the clock frequency provided by the Avalon interface. Changing the system clock frequency in hardware without regenerating the IrDA UART Core hardware will result in incorrect signaling.

2.1.2 Baud Rate (bps) Setting

The Baud Rate setting determines the default baud rate after reset. The Baud Rate option offers standard preset values (e.g., 9600, 57600, 115200 bps).

The baud rate value is used to calculate an appropriate clock divisor value to implement the desired baud rate. Baud rate and divisor values are related as follows:

$$\text{Divisor} = \text{int}((\text{clock frequency}) / (\text{baud rate}) + 0.5)$$

$$\text{Baud rate} = (\text{clock frequency}) / (\text{divisor} + 1)$$

2.1.3 Data Bits, Stop Bits, Parity

The UART core's parity, data bits and stop bits are configurable. These settings are fixed at system generation time; they cannot be changed via the core's registers. The available settings are shown in Table 1.

Table 1. Bit Settings		
Settings	Allowed Values	Description
Data Bits	7, 8, 9	This setting determines the widths of the txdata, rxdata, and endofpacket registers.
Stop Bits	1, 2	This setting determines whether the core transmits 1 or 2 stop bits with every character. The core always terminates a receive transaction at the first stop bit, and ignores all subsequent stop bits, regardless of the Stop Bits setting.
Parity	None, Even, Odd	This setting determines whether the UART transmits characters with parity checking, and whether it expects received characters to have parity checking. See below for further details.

2.1.3.1 Parity Setting When Parity is set to None, the transmit logic sends data without including a parity bit, and the receive logic presumes that the incoming data does not include a parity bit. When parity is None, the data register's PE (parity error) bit is not implemented; it always reads 0.

When Parity is set to Odd or Even, the transmit logic computes and inserts the required parity bit into the outgoing TXD bit stream, and the receive logic checks the parity bit in the incoming RXD bit stream. When parity is Even, the parity bit is 1 if the data has an even number of 1 bits; otherwise the parity bit is 0. Similarly, when parity is Odd, the parity bit is 1 if the data has an odd number of 1 bits.

3 Software Programming Model

3.1 Register Map

Table 2 shows the register map for the IrDA UART core. Device drivers control and communicate with the core through the two 32-bit memory-mapped registers, shown in Table 2.

Table 2. IrDA UART Core Register Map												
Offset in bytes	Register Name	R/W	Bit Description									
			31...16	15	14...11	10	9	8	7	6...2	1	0
0	data	RW	RAVAIL	RVALID	(1)		PE	(2)	(2)	DATA		
4	control	RW	WSPACE	(1)			WI	RI	(1)		WE	RE

Notes on Table 2:

- (1) Reserved. Read values are undefined. Write zero.
- (2) These bits may or may not exist, depending on the **Data Width** hardware options.
If they do not exist, they read zero and writing has no effect.

3.1.1 Data Register

The read and write FIFOs are accessed via the data register.

A read from the data register returns the first character from the FIFO (if one is available) in the

Table 3. Data Register Bits

Bit Number	Bit/Field Name	Read/Write/Clear	Description
8...0	DATA	R/W	The value to transfer to/from the IrDA UART core. When writing, the DATA field is a character to be written to the write FIFO. When reading, the DATA field is a character read from the read FIFO.
9	PE	R	Indicates whether the DATA field had a parity error.
15	RVALID	R	Indicates whether the DATA field is valid. If RVALID=1, then the DATA field is valid, else DATA is undefined.
32...16	RAVAIL	R	The number of characters remaining in the read FIFO (after this read).

DATA field. Reading also returns information about the number of characters remaining in the FIFO in the RAVAIL field. A write to the data register stores the value of the DATA field in the write FIFO. If the write FIFO is full, then the character is lost.

3.1.2 Control Register

IrDA UART core's interrupt generation and read status information are controlled by the control register. Table 4 describes the function of each bit.

Table 4. Control Register Bits

Bit Number	Bit/Field Name	Read/Write/Clear	Description
0	RE	R/W	Interrupt-enable bit for read interrupts
1	WE	R/W	Interrupt-enable bit for write interrupts
8	RI	R	Indicates that the read interrupt is pending
9	WI	R	Indicates that the write interrupt is pending
32...16	WSPACE	R	The number of spaces available in the write FIFO.

A read from the control register returns the status of the read and write FIFOs. Writes to the register can be used to enable/disable interrupts.

The RE and WE bits enable interrupts for the read and write FIFOs, respectively. The WI and RI bits indicate the status of the interrupt sources, qualified by the values of the interrupt enable bits (WE and RE). RI and WI can be examined to determine what condition generated the interrupt request.

3.2 Programming with the IrDA UART Core

The IrDA UART core is packaged with C-language functions accessible through the SOPC software development kit (SDK) libraries. These functions implement basic operations that users need for the IrDA UART core. When using the Altera Debug Client, these functions are automatically provided for use in a C-language application program, as listed in Section 3.3. To use the functions, the C code must include the statement:

```
#include "alt_up_irda.h"
```

3.3 IrDA Functions

3.3.1 int enable_read_interrupt ()

Enable the read interrupts for IrDA UART Core.

Returns:

0 for success

3.3.2 int disable_read_interrupt ()

Disable the read interrupts for IrDA UART Core.

Returns:

0 for success

3.3.3 int get_parity_checking ()

Get whether the DATA field has a parity error.

Returns:

0 for no errors, -1 for parity error occurs

3.3.4 unsigned get_num_chars_in_FIFO ()

Get the number of characters remaining in the read FIFO (after this read).

Returns:

the number of characters remaining

3.3.5 int write_data (alt_u16 data)

Write data to the IrDA UART Core.

Parameters:

data – the character to be transferred to the IrDA UART Core

Returns:

0 for success or -1 on error

3.3.6 int read_data (alt_u16 * *data*)

Read data from the IrDA UART Core.

Parameters:

data – pointer to the memory where the character read from IrDA UART Core should be stored

Returns:

0 for success or -1 on error

Note:

This function will use the `get_parity_checking` to check the parity for the DATA field

