IrDA UART for Altera's DE-Series Boards

For Quartus II 15.0

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

The IrDA UART core can be instantiated in a system using Qsys or as a standalone component from the IP Catalog within the Quartus II software. Designers use the core's configuration wizard to specify the desired 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 Interface Settings

The IrDA UART Core can either have a Avalon Memory-Mapped port or two Avalon Streaming ports. It is recommended to select Memory Mapped when connecting to a processor, otherwise set it to Streaming. The Incoming clock rate must be set to the value of the frequency of the clock that will be driving the IrDA UART.

2.1.2 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.

2.1.3 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:

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

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

2.1.4 Data Width, Stop Bits, and 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 Width	7, 8, 9 bits	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.

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 when Memory-Mapped Avalon Type is selected for the 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...24	23...16	15	14...11	10	9	8	7	6...2	1	0
0	`data`	RW	*(1)*	RAVAIL	RVALID	*(1)*		PE	*(2)*	*(2)*	DATA		
4	`control`	RW	*(1)*	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 specified **Data Width**.
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.

Table 3. Data Register Bits			
Bit Number	**Bit/Field Name**	**Read/Write**	**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 and PE fields contain valid data.
23...16	RAVAIL	R	The number of characters remaining in the read FIFO (including this read).

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

A read from the control register returns the status of the read and write FIFOs. Writing to the register is 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 setting of the interrupt enable bits (WE and RE). Bits RI and WI can be examined to determine what condition generates the interrupt request.

Table 4. Control Register Bits			
Bit Number	**Bit/Field Name**	**Read/Write**	**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
23...16	WSPACE	R	The number of spaces available in the write FIFO.

3.2 Device Driver for the Nios II Processor

The IrDA UART core is packaged with C-language functions accessible through the hardware abstraction layer (HAL). These functions implement basic operations that users need for the IrDA UART core. To use the functions, the C code must include the statement:

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

3.2.1 alt_up_irda_enable_read_interrupt

Prototype: `void alt_up_irda_enable_read_interrupt(alt_up_irda_dev *irda)`
Include: `<altera_up_avalon_irda.h>`
Parameters: `irda` – the IRDA device structure
Description: Enable the read interrupts for the IRDA UART core.

3.2.2 alt_up_irda_disable_read_interrupt

Prototype: `void alt_up_irda_disable_read_interrupt(alt_up_irda_dev *irda)`
Include: `<altera_up_avalon_irda.h>`
Parameters: `irda` – the IRDA device structure
Description: Disable the read interrupts for the IRDA UART core.

3.2.3 alt_up_irda_check_parity

Prototype: `int alt_up_irda_check_parity(alt_u32 data_reg)`
Include: `<altera_up_avalon_irda.h>`
Parameters: `data_reg` – the date register
Returns: 0 for no errors, −1 for parity error.
Description: Check whether the DATA field has a parity error.

3.2.4 alt_up_irda_get_used_space_in_read_FIFO

Prototype: `unsigned alt_up_irda_get_used_space_in_read_FIFO(alt_up_irda_dev *irda)`
Include: `<altera_up_avalon_irda.h>`
Parameters: `irda` – the IrDA device structure
Returns: The number of data words remaining.
Description: Gets the number of data words remaining in the read FIFO.

3.2.5 alt_up_irda_get_available_space_in_write_FIFO

Prototype: `unsigned alt_up_irda_get_available_space_in_write_FIFO(alt_up_irda_dev *irda)`
Include: `<altera_up_avalon_irda.h>`
Parameters: `irda` – the IrDA device structure
Returns: The amount of available space remaining.
Description: Gets the amount of available space remaining in the write FIFO.

3.2.6 alt_up_irda_write_data

Prototype: `int alt_up_irda_write_data(alt_up_irda_dev *irda, alt_u8 data)`
Include: `<altera_up_avalon_irda.h>`
Parameters: `irda` – the IRDA device structure
`data` – the character to be transferred to the IRDA UART Core.
Returns: 0 for success or −1 on error.
Description: Write data to the IRDA UART core.
Notes: User should ensure the write FIFO is not full before writing, otherwise the character is lost.

3.2.7 alt_up_irda_read_data

Prototype: `int alt_up_irda_read_data(alt_up_irda_dev *irda, alt_u8 *data, alt_u8 *parity_error)`
Include: `<altera_up_avalon_irda.h>`
Parameters: `irda` – the IRDA device structure
`data` – pointer to the memory where the character read from the IRDA UART core should be stored.
`parity_error` – pointer to the memory where the parity error should be stored.
Returns: 0 for success or −1 on error.
Description: Read data from the IRDA UART core.
Notes: This function will clear the DATA field of the data register after reading and it uses the `alt_up_irda_check_parity` function to check the parity for the DATA field.

3.2.8 alt_up_irda_read_fd

Prototype: `int alt_up_irda_read_fd(alt_fd *fd, char *ptr, int len)`
Include: `<altera_up_avalon_irda.h>`
Parameters: –
Description:

3.2.9 alt_up_irda_write_fd

Prototype: `int alt_up_irda_write_fd(alt_fd *fd, const char *ptr, int len)`
Include: `<altera_up_avalon_irda.h>`
Parameters: –
Description:

3.2.10 alt_up_irda_open_dev

Prototype: `alt_up_irda_dev* alt_up_irda_open_dev(const char *name)`
Include: `<altera_up_avalon_irda.h>`
Parameters: `name` – the device name in Qsys
Returns: the device structure
Description: Open the IRDA device according to device name.