Key Design Features

- Synthesizable, technology independent VHDL IP Core
- UART compatible serial interface controller
- Receive input FIFO with configurable depth
- Transmit output FIFO with configurable depth
- Supports all standard data rates from 9600 to 921600 baud
- Fully custom data rates also supported – limited only by system clock frequency
- 5, 6, 7 or 8-bit data payload width with 1 or 2 stop bits
- Even, odd, mark, space or no parity
- Receive and transmit interrupt flags
- Receive and transmit FIFO full flags

Applications

- UART communications using a variety of electrical standards such as RS232, RS422 and RS485 etc.
- Control in industrial, commercial and lab environments
- Basic PC-to-board interfacing and debug

Block Diagram

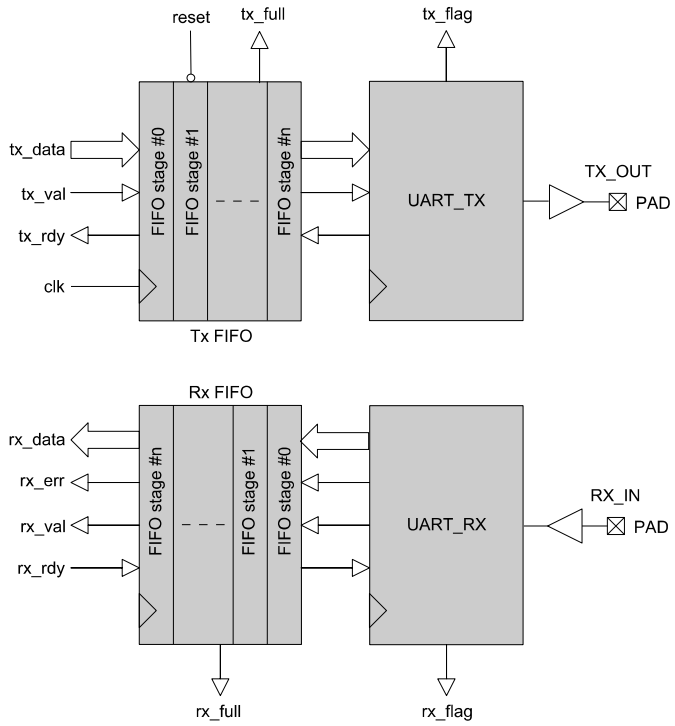

Figure 1: UART serial interface controller architecture

Pin-out Description

Pin name	*I/O*	*Description*	*Active state*
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
rx_flag	out	Byte received flag	high
rx_full	out	Receive FIFO full flag	high
tx_flag	out	Byte transmitted flag	high
tx_full	out	Transmit FIFO full flag	high
rx_in	in	Serial bits in	serial data
tx_out	out	Serial bits out	serial data
rx_data[7:0]	out	Received data	data
rx_err	out	Parity error flag (qualified by rx_val)	high
rx_val	out	Received data valid	high
rx_rdy	in	Received data ready handshake	high
tx_data[7:0]	in	Transmit data	data
tx_val	in	Transmit data valid	high
tx_rdy	out	Transmit data ready handshake	high

Generic Parameters

Generic name	*Description*	*Type*	*Valid range*
baudrate	Baud rate in bits per second	integer	9600 to 921600 (custom rates also supported)
sclkfreq	System clock frequency in Hz	integer	Ratio (sclkfreq / baurate) < 65536
databits	Number of data bits	integer	5,6,7 or 8
stopbits	Number of stop bits	integer	1,2
parity	Enable parity bit after data payload in bitstream	integer	0: none 1: even 2: odd 3: mark 4: space
rxfifo_depth	Receive data FIFO depth	integer	≥ 2
rxfifo_depth_log2	Receive data FIFO depth log2	integer	log2 (rxfifo_depth)
txfifo_depth	Transmit data FIFO depth	integer	≥ 2
txfifo_depth_log2	Transmit data FIFO depth log2	integer	log2 (txfifo_depth)

General Description

UART_CONT is a robust UART-compliant serial interface controller capable of receiving and transmitting bits serially. It has a configurable data payload from 5 to 8-bits (with or without parity) and supports either 1 or 2 stop bits.

Both the receiver and transmitter circuits have a configurable FIFO which may be used to buffer the parallel input and output data as required. In addition, the controller features a number of flags to indicate that a byte has been received, a byte has been sent, and that the FIFOs are full.

Under normal conditions, the controller will support baud rates of 9600 to 921600 baud - although higher and lower rates may be supported depending on the choice of system clock frequency. Fully custom baud rates are also permitted. Internally, the controller uses a 16-bit clock divider for UART serial timing. As long at the ratio: (*sclkfreq* / *baudrate*) is in the range 16 to 65535 then the UART timing will be valid.

The UART controller is comprised of four main blocks as described by Figure 1. These blocks are the Receiver (De-serializer), the Transmitter (Serializer) and the receive and transmit FIFOs.

Both the receive and transmit FIFOs use the standard Zipcores valid-ready handshake protocol. This means that data is written or read from the FIFOs on the rising-edge of *clk* when *val* and *rdy* are both high[1].

The transmit FIFO may be used to 'queue up' a sequence of bytes to be sent via the UART interface. Likewise, the receive FIFO may be used to buffer incoming words. When the receive FIFO is full the flag *rx_full* is asserted and will remain high until the FIFO is emptied. If the receive FIFO is full, then any further bytes received will be lost until the the FIFO has sufficient capacity.

Functional Timing

Figure 2 shows the format of the bit stream at the UART receiver. The example demonstrates the timing waveform at 9600 baud in which the duration of a bit is approximately 104 us.

A frame begins with a START bit (logic '0') then the bits are read starting with the LSB and ending with the MSB. The frame terminates with a STOP bit (Logic '1'). The design may be configured to use 5, 6, 7 or 8 data bits, an optional parity bit and 1 or 2 stop bits.

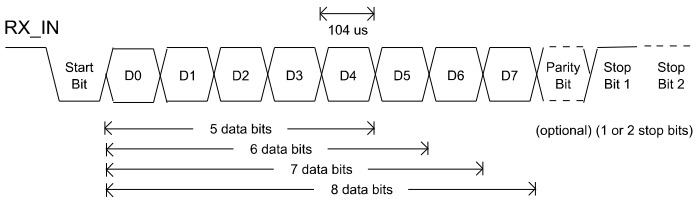

Figure 2: UART serial bitstream format

Figure 3 demonstrates the corresponding receiver output for the input bit-stream in Figure 2. Note that the interface is initially stalled with *rx_rdy* asserted low. Once the rdy handshake is asserted high, the data is transferred.

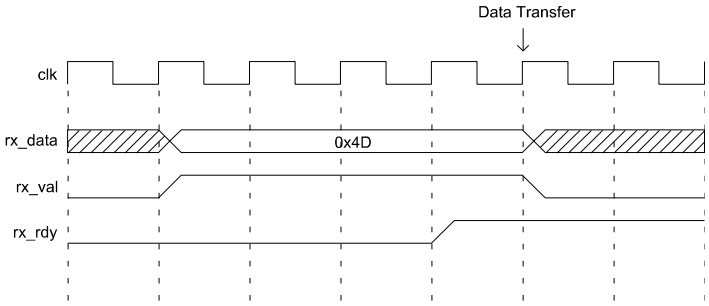

Figure 3: UART Receiver valid-ready handshake

The UART transmitter timing is exactly the same, with the exception that the data direction is reversed.

The additional signals *rx_flag*, *tx_flag* and *rx_err* are not shown in the timing diagrams. The rx and tx flags function as simple strobes that are asserted for one system clock cycle when data is written to and read from and to the Rx and Tx FIFOs. The *rx_err* signal is a flag that is asserted high when the receiver detects a parity error in the received data. This signal is qualified by the *rx_val* signal and and when *rx_val* is low it should be ignored.

Source File Description

All source files are provided as text files coded in VHDL. The following table gives a brief description of each file.

Source file	Description
uart_fifo.vhd	Transmit and Receive FIFOs
uart_tx.vhd	UART Transmitter
uart_rx.vhd	UART Receiver
uart_cont.vhd	Top-level block
uart_cont_bench.vhd	Top-level test bench
uart_file_reader.vhd	File reader for transmit data
uart_tx_in.txt	Transmit data text file

Functional Testing

An example VHDL test bench is provided for use in a suitable VHDL simulator. The compilation order of the source code is as follows:

1. uart_fifo.vhd
2. uart_rx.vhd
3. uart_tx.vhd
4. uart_cont.vhd
5. uart_file_reader.vhd
6. uart_cont_bench.vhd

The VHDL test bench instantiates the UART_CONT component in a loop-back configuration with the serial data transmit stream feeding into the serial receive stream. The baud rate, parity, number of stop bits and system clock frequency may be adjusted by the user as required.

1 See Zipcores application note: app_note_zc001.pdf for more examples of the valid-ready protocol and it's implementation

The input stimulus for the test is provided by the file *uart_tx_in.txt.* This stimulus file should be put in the current top-level VHDL simulation directory. The input text file is a sequence of 8-bit data words (in hex) on consecutive lines that represent the data to be transmitted.

In the default set up, the simulation must be run for around 100 ms during which time the file-reader module will read the 8-bit data to be transmitted. The controller is set up in a loop-back configuration with parity set to 'none'.

The simulation generates an output text file called: *uart_rx_out.txt.* This file contains the 8-bit data captured at the receiver outputs during the course of the simulation. At the end of the test, the input and output files may be compared to verify correct operation. Both these files should be identical as the UART is configured in loop-back.

Development Board Testing

The UART Serial Controller was implemented on a Xilinx® 2V3000 FPGA running at a system clock frequency of 65MHz. An Intersil® ICL3232CPZ line-driver chip was used to provide the RS232-level interfacing.

A simple serial communications program was written for a PC to allow a series of characters to be sent and received using the standard (COM1) serial port. The port was configured to use 8 data bits, 1 stop bit and no parity. Various baud rates were tested to ensure correct operation at different frequencies.

The first series of tests were carried out at 115200 baud. Figure 4 demonstrates the result of sending the character 'U' (0x55 in hex) to the UART controller. The FPGA-based controller was set up in a loop-back configuration so that the received bits were were re-transmitted. The upper trace shows the serial bits on the Rx pin. The bottom trace shows the resulting data on the Tx pin.

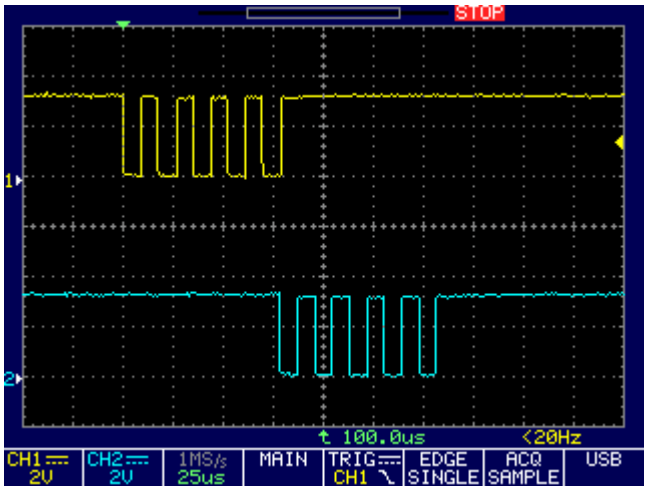

Figure 4: Receive/Transmit 0x55 at 115200 baud

Figure 5 shows detail of the mark-space ratio of the same transmitted output bits. For a baud rate of 115200, then the width of each bit should be around 8.7 us.

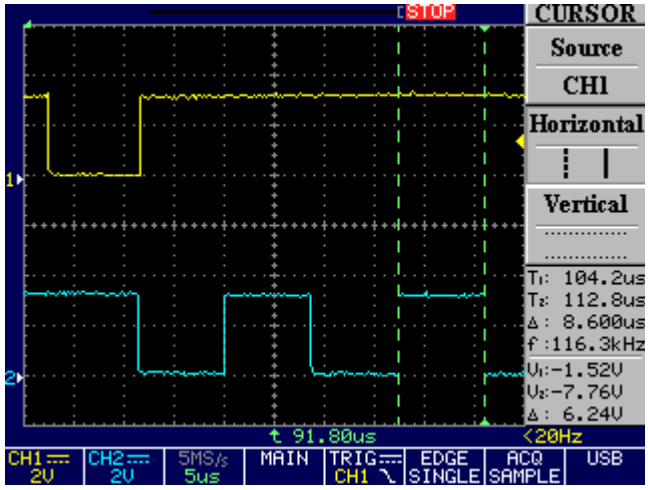

Figure 5: Timing detail at 115200 baud

Figure 6 shows the UART controller in the same loop-back configuration, but this time working at 921600 baud. At this data rate, the PC serial controller card was working at the limit of it's specified performance. The resulting mark space ratio was quite poor. However, the FPGA-based UART Controller was still capable of decoding the bit-stream correctly.

Note that the FPGA-based UART Controller resends the data 0x55 with a correctly proportioned mark-space ratio.

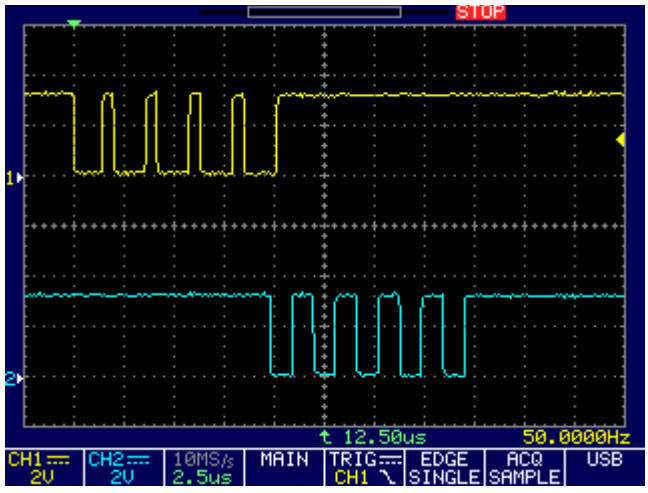

Figure 6: Receive/Transmit 0x55 at 921600 baud

Figure 8 on the following page shows a detailed view of the transmitted bits. For a baud rate of 921600 then the duration of a bit should be around 1.085 us.

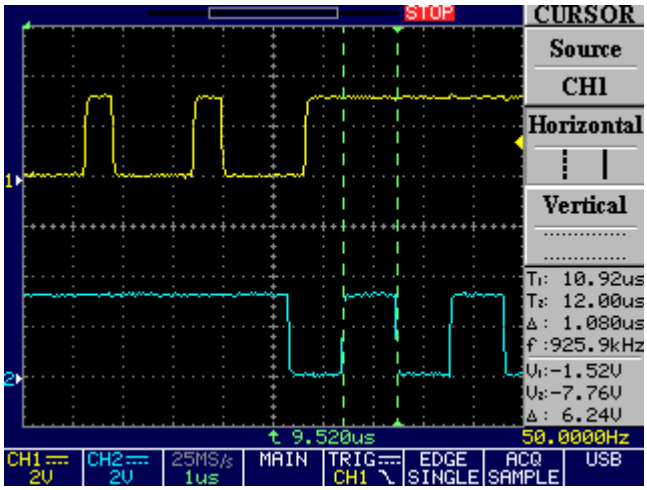

Figure 7: Timing detail at 921600 baud

Synthesis

The files required for synthesis and the design hierarchy is shown below:

- uart_cont.vhd
  - uart_fifo.vhd
  - uart_rx.vhd
  - uart_tx.vhd

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx® Virtex 6 and Spartan 6 FPGA devices. Synthesis results for other FPGAs and technologies can be provided on request.

Note that in order to achieve the fastest and most area efficient designs the size of the FIFOs should be kept to a minimum.

Trial synthesis results are shown with the generic parameters set to: baudrate = 115200, sclkfreq = 3686400, databits = 8, stopbits = 1, parity = 0, rxfifo_depth = 16, rxfifo_depth_log2 = 4, txfifo_depth = 16, txfifo_depth_log2 = 4.

Resource usage and timing is specified after Place and Route.

VIRTEX 6

Resource type	***Quantity used***
Slice register	75
Slice LUT	107
Block RAM	0
DSP48	0
Occupied Slices	35
Clock frequency (approx)	430 MHz

SPARTAN 6

Resource type	***Quantity used***
Slice register	75
Slice LUT	108
Block RAM	0
DSP48	0
Occupied Slices	37
Clock frequency (approx)	320 MHz

Revision History

Revision	***Change description***	***Date***
1.0	Initial revision	03/11/2008
1.1	Added parity bit support	18/10/2011
1.2	Updated synthesis results for Xilinx® 6 series FPGAs	23/10/2012
1.3	Updated synthesis results in line with minor source-code changes. Added the tx_full flag	04/05/2014
1.4	Added generic to allow cofigurable number of data bits the the payload	07/07/2014