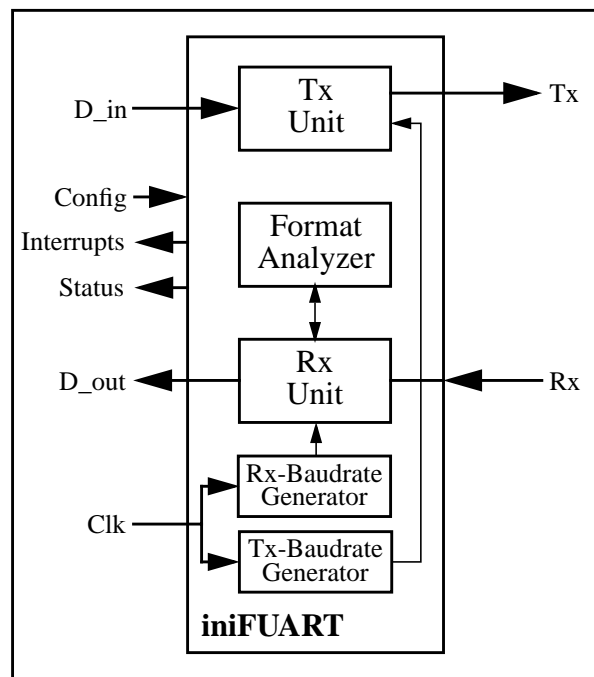# iniFUART

Features:

- Configurable Transfer Rate: 1200bps to 115.2kbps with Accuracy Better than 0.15% from 1MHz Clock!
- Data Format: 7, 8 Bits
- Parity Enable, Odd/Even, Error Detection
- Stop Bit: 1, 2 Bits
- Format Check
- Glitch Rejection
- Parallel Interface with Event Control
- Structured, Synchronous VHDL Design
- Flexible Interfaces

Structure of iniFUART:

INICORE - the reliable Core and System Provider.

We provide high quality IP, design expertise and

leading edge silicon to the industry.

The **iniFUART** is an innovative, flexible implementation of a Fast Universal Asynchronous Receiver Transmitter (FUART) device. iniFUART, which uses the RS-232 serial protocol, provides the interface between a microprocessor and a serial port or between the system and a standard serial port. The core contains a highly accurate programmable baudrate generator, serial receiver and transmitter, communication channels, and interrupt control signals.

The iniFUART core may be used as a data link layer with parallel interfaces and event communication. Application-specific blocks (e.g., interrupt controller, special interfaces, status reporting circuits) can then be built around the iniFUART and will not affect the main functionality.

INICORE's in-depth know how in serial communication and frequency synthesizers gives you the best benefit for the FUART of your needs.

INICORE offers the structural VHDL FUART simulation/synthesis model for the target technology of your choice.

US Sales Office:
INICORE INC.
5600 Mowry School Road, Suite 180,
Newark, CA 94560
Tel: 510 445 1529 Fax: 510 656 0995
E-mail: ask_us@inicore.com
Web: www.inicore.com

INICORE AG
Mattenstrasse 6a,CH-2555 Brügg, Switzerland
Tel: ++41 32 374 32 00, Fax: ++41 32 374 32 01
E-mail: ask_us@inicore.ch
Web: www.inicore.ch

1 Overview

The iniFUART core may be used as a data link layer with parallel interfaces and event communication. Microprocessor specific interfaces must be built around the iniFUART, as well as queues, interrupt controllers and status reporting circuits. The following picture shows all inputs and outputs:
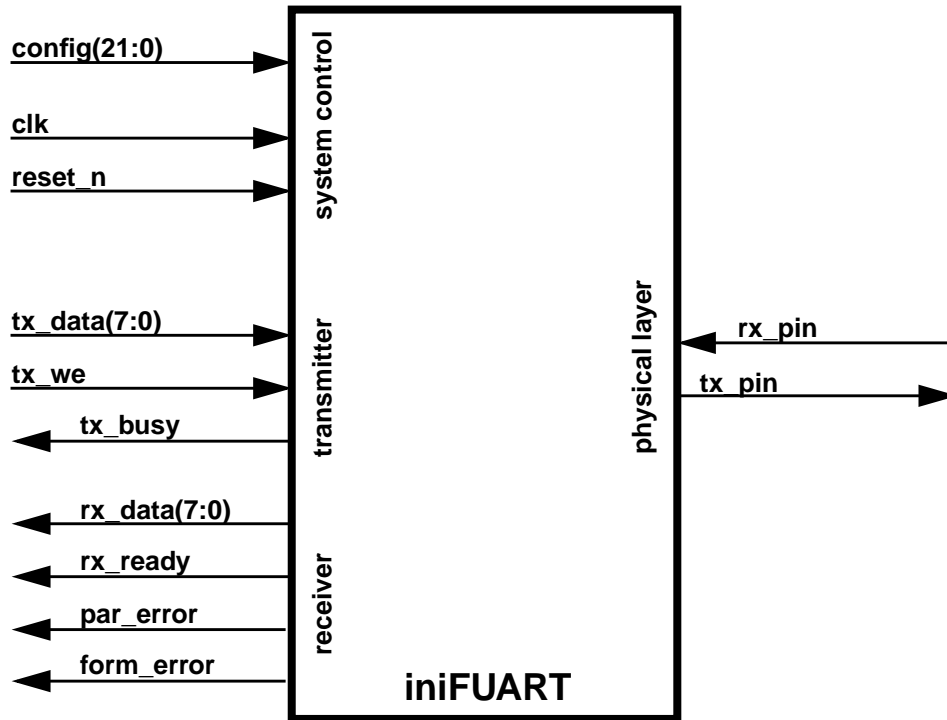
1.1 Event communication

For communicating events, the iniFUART core uses or produces always active '1' pulses, which are activated for only one clk cycle. In the inactive state, they remain low with respect to the rising clk edge, so glitches may occur. For communicating over clock domains, these events must be synchronized first!
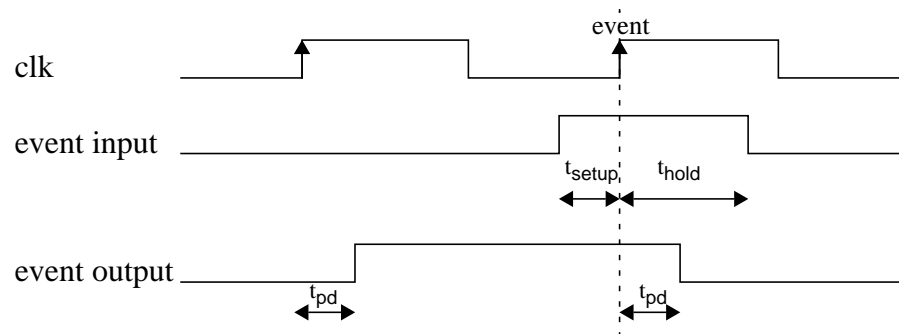
The parameters t_{setup}, t_{hold} and t_{pd} are technology dependent and must be determined according to the chosen technology.

2 IO description   The following part lists the input and output ports of the iniFUART core and gives a short overview of their functionality.

2.1 General inputs   These pins are used to clock and initialize the whole iniFUART core. There are no other clocks in this core.

pin name	type	description
clk	in	system clock for the whole iniFUART
reset_n	in	asynchronous system reset, active low

2.2 Configuration   The configuration pins are used to set the bitrate, bit timing and output format. They're static inputs. and used for both receiver and transmitter in common.

pin name	type	description
baudrate[15:0]	in	Defines the baudrate. For formulas, resolution etc. see 'Baudrate generator'
data_78	in	'0': use 7 bit data '1': use 8 bit data
par_ebl	in	'0': no parity check, no parity bit transmitted and received '1': use parity check, parity bit inserted and checked
par_pol	in	'0': use even parity (number of ones in a byte, including parity bit is even) '1': use odd parity (number of ones in a byte, including parity bit is odd) ignored when par_ebl is inactive!
stop_12	in	'0': use and check 1 stop bit '1': use and check 2 stop bits
tx_run	in	'0': inactive transmitter, ignores all inputs, outputs are inactive '1': transmitter is working
rx_run	in	'0': inactive receiver, ignores all inputs, outputs are inactive '1': receiver is working

2.3 Serial interface   The serial interface includes the receive and transmit path separately. It is a full duplex solution, receive and transmit is possible at the same time.

pin name	type	description
rx_pin	in	Pin for the incoming bit stream. The inactive state is logic 1
tx_pin	out	Pin for the outgoing bit stream. The inactive state is logic 1

2.4 Transmitter interface

For transmitting data, a parallel event controlled interface is used. It is an efficient way to embed the iniFUART in systems as well as connecting simple or complex specific interfaces, including queues etc., to it.

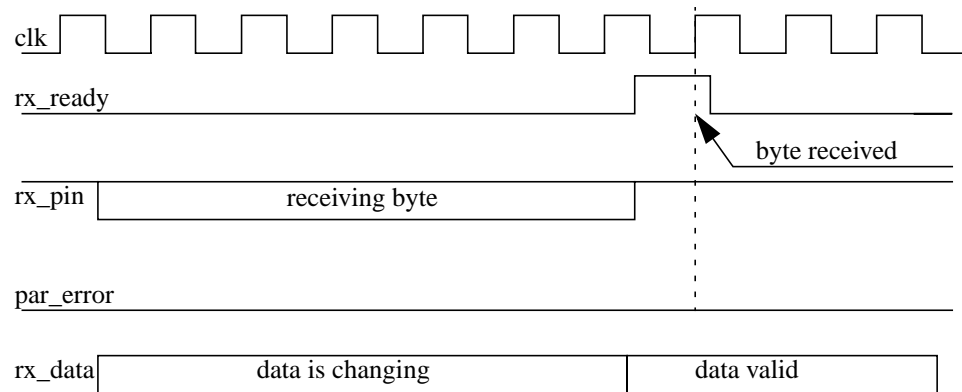| pin name	type	description
tx_data[7:0]	in	8bit data to be transmitted. For 7bit configuration, bit[7] is ignored (use only tx_data[6:0]). The data must be valid when tx_we is active.
tx_we	in	Event for storing the tx_data in the transmit shift register. It's up to the system to not activate this input when the iniFUART is busy.
tx_busy	out	When the transmitter is sending a byte, this status output remains active (logic 1) until it is ready to send a new byte. While tx_busy is 1, tx_we mustn't be activated.

The following diagram shows a typical case:
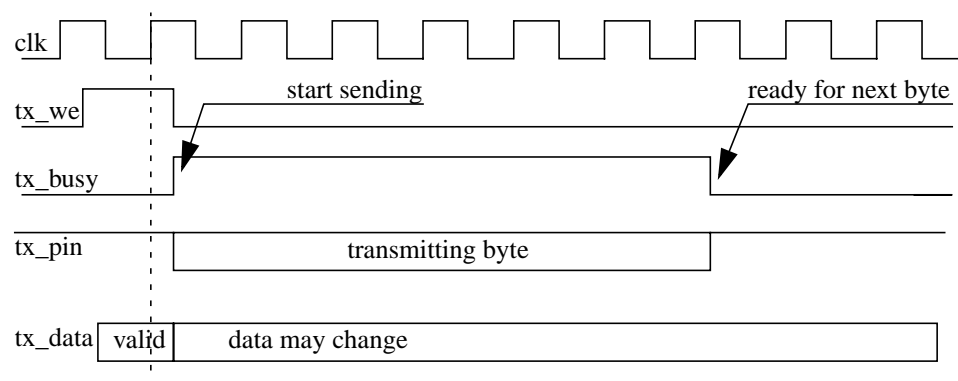
The transmitter path stores the incoming byte in the shift register by means of the tx_we signal and starts the transmitting activity. tx_busy goes high also and remains high until the data is sent.
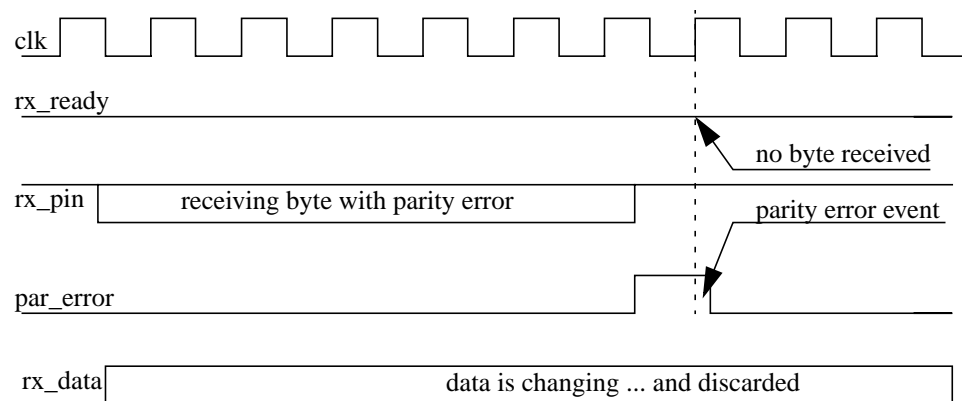
2.5 Receiver interface

For receiving data, the same type of interface is used as in the transmitter path.

pin name	type	description
rx_data[7:0]	out	8bit data that has been received. For 7bit configuration, bit[7] is ignored (use only rx_data[6:0]). The data will be stable when rx_ready is active, but it will change when the iniFUART is actually receiving a byte. For making the received byte available permanently, it has to be buffered.
rx_ready	out	Event (active 1) for signalling, that a new byte has arrived and the rx_data is valid now.
par_error	out	Event (active 1) for signalling, that a byte with wrong parity has been received and aborted (it's not visible at rx_ready) This signal is always inactive when par_ebl is deactivated.
form_error	out	Event (active 1) for signalling, that a byte with wrong format has been received and aborted (it's not visible at rx_ready)

This is a normal case, where a correct byte arrives...

clk

rx_ready

byte received

rx_pin   receiving byte

par_error

rx_data   data is changing   data valid

... and when a parity error occurs

clk

rx_ready

no byte received

rx_pin   receiving byte with parity error

par_error   parity error event

rx_data   data is changing ... and discarded

The receiver path contains several checks and special features. First, the level at the rx_pin is watched. When a falling edge is detected, the receiver is started. Glitches are successfully rejected since the level of the start bit is fully checked.

Now, data is shifted in the receive register and after the check of the stop bit(s) and the parity bit (when configured so) the byte is marked as valid. The following list shows the cases where an incoming byte is aborted:

Glitch on rx_pin	not reported
Stop bit(s) not equal logic 1	not reported
Parity bit doesn't match internally calculated parity	reported by par_error

3 Baudrate generator

The two independent baudrate generators produce the bit clocks, which are used for transmitting and receiving data. Both generators are configured by the same input (baudrate[15:0]) and therefore are working with the same speed. The advantage of using two independent baudrate generators is that there will be no unnecessary wait states in the transmitter block because of synchronizing to the bit clock. The Rx baudrate generator is controlled by the falling edge of an incoming start bit.

The baudrate generator is not a simple prescaler, but an innovative DCO (digitally controlled oscillator) which allows generating all baudrates from the system clock. There is no special quartz needed for that purpose and you're free in choosing the system clock frequency.

For configurating the baudrate, the 16bit value is calculated according to the following formula:

$$Baudrate = fclk\frac{n}{2^{18}} \quad \text{resp.} \quad n = \frac{2^{18}}{fclk}Baudrate$$

where
Baudrate is the transmitting speed in bits per second
fclk is the system clock speed in Hz
n is the 16bit value to be programmed

Example:
For 1Mhz clock and 64kbps, n is 16777(dec), accuracy better than 13ppm.
For 2Mhz clock and 19.2kbps, n is 2517(dec), accuracy better than 170ppm.

Remark: The faster the baudrate, the better the accuracy, but more jitter is added. Maximum absolute jitter is always equal 1/fclk.