Product Description

The iniUART is an innovative, flexible implementation of an UART (Universal Asynchronous Receiver Transmitter). The iniUART implements the RS-232 serial protocol, provides the interface between a microprocessor and a serial port or between the system and a standard serial port.

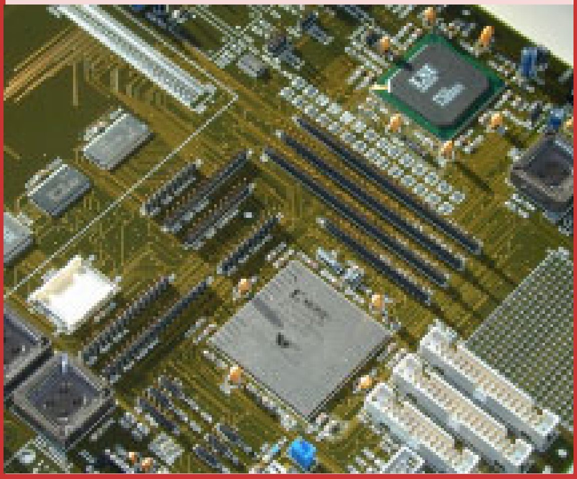
Key features: The core contains a highly accurate programmable baud rate generator, which allows to generate any baudrate independently (up to 1/64) of the system clock speed. 3 point input sampling and glitch rejection are implemented in the serial receiver.
The iniUART core is be used as a data link 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 iniUART without modification of the iniUART. MDS offers also complete UART modules, please refer to our webpage.
MDS's in-depth know how in serial communication and frequency synthesizers gives you the best benefit for the UART of your needs.
MDS offers the structural VHDL UART simulation/synthesis model for the target technology of your choice.

Features:

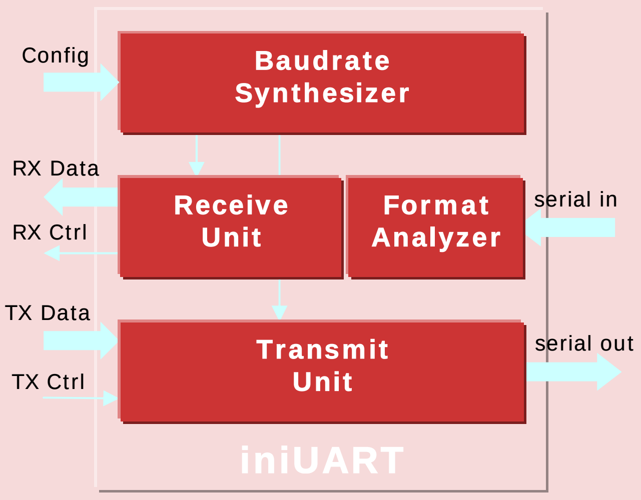
- **Baudrate Synthesizer for any baudrate up to 1/64 of clock speed**
- **No dedicated clock freq.**
- **7 or 8 Bits Data**
- **No/Odd/Even Parity**
- **Error Detection**
- **1 or 2 Stop Bits**
- **Format Check**
- **3-Point Input Sampling**
- **Parallel Interface with Event Control**

iniUART

Smart UART Technology



iniUART Structure



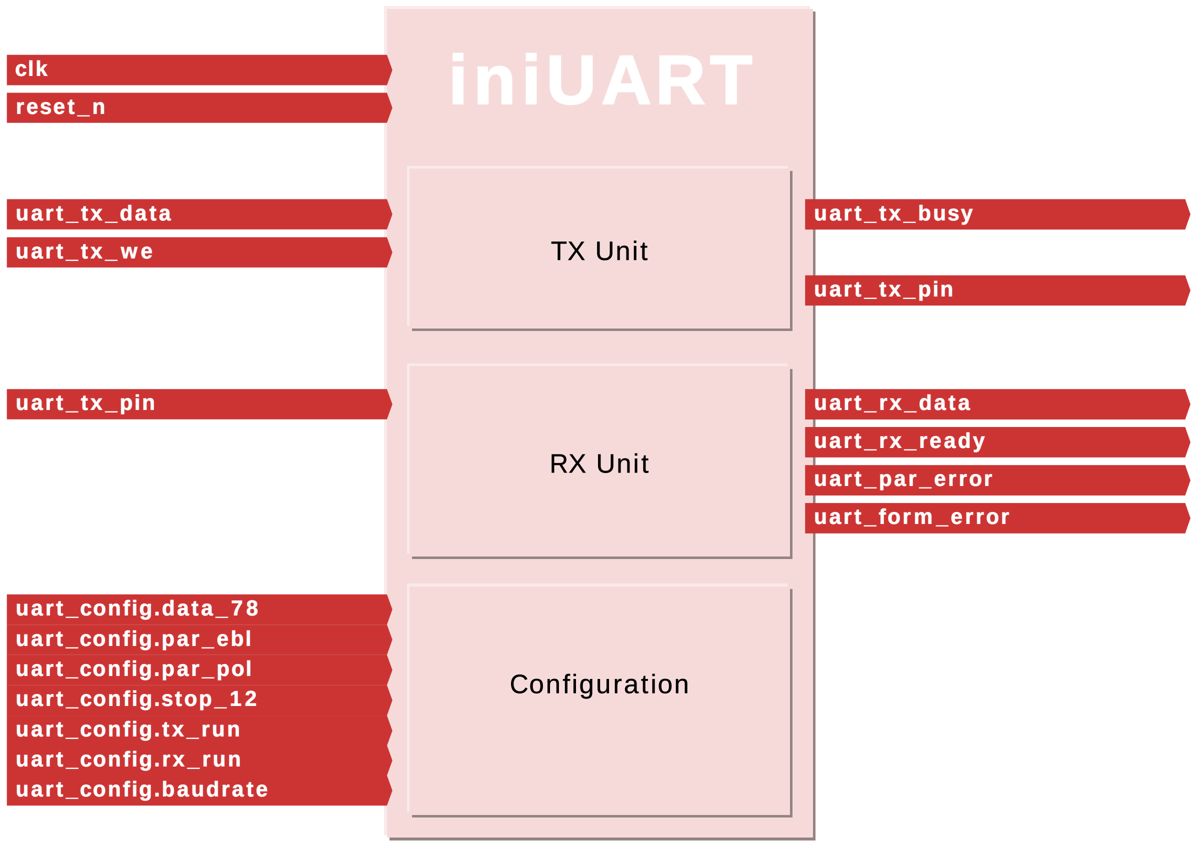
Memec Design Services

Memec Design Services
Memec Inicore AG
Mattenstrasse 6a
CH-2555 Brügg / Switzerland
e-mail: ask_us@inicore.ch
http://www.inicore.ch

Table of Content

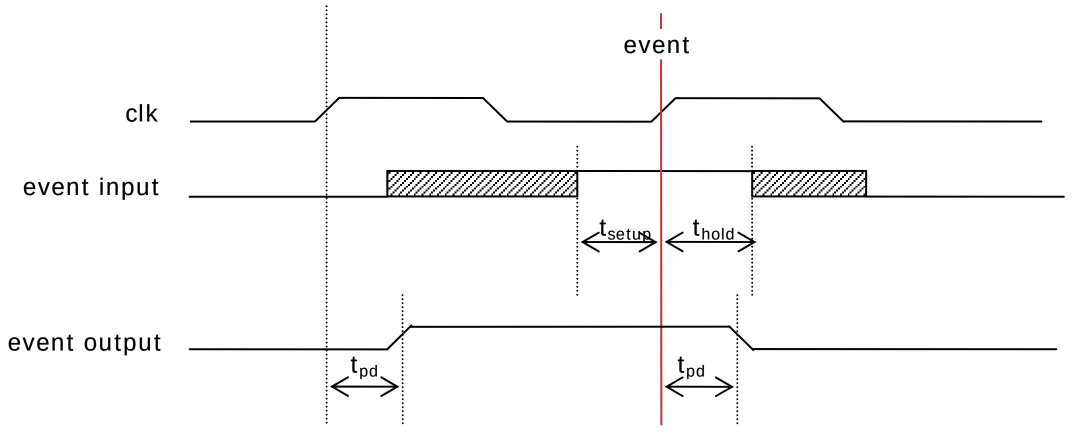
1 Overview

The iniUART core is generally used as a data link layer with parallel interfaces and event communication. Microprocessor specific interfaces are built around the iniUART, as well queues, interrupt controllers and status reporting circuits[1]. The following picture shows all inputs and outputs:



1.1 Event communication

For communicating events, the iniUART core uses or produces 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.

[1] For complete UART solutions, please check our iniUART 16f datasgeet.

2 IO description

The following part lists the input and output ports of the iniUART core and gives a short overview of their functionality.

2.1 General inputs

These pins are used to clock and initialize the whole iniUART core. There are no other clocks in this core.

pin name	type	description
clk	in	system clock, rising edge used only, must be at least 64 times higher than maximum baudrate
reset_n	in	asynchronous system reset, active low, goes to all flip flops

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
uart_config.baudrate[15:0]	in	Defines the baudrate. see 2.2.1, Baudrate for more details
uart_config.data_78	in	Transmit and receive data size: '0': use 7 bit data '1': use 8 bit data
uart_config.par_ebl	in	Parity enable: '0': no parity check, no parity bit transmitted and received '1': use parity check, parity bit inserted and checked
uart_config.par_pol	in	Parity polarity: '0': use even parity[2] '1': use odd parity This parameter is ignored when par_ebl is inactive!
uart_config.stop_12	in	Transmit and receive stop bit number: '0': use and check 1 stop bit '1': use and check 2 stop bits
uart_config.tx_run	in	Transmit control: '0': transmitter off, ignores all inputs, outputs inactive '1': transmitter is working
uart_config.rx_run	in	Receive control: '0': receiver off, ignores all inputs, outputs are inactive '1': receiver is working

2.2.1 Baudrate

The baudrate generator is not a simple prescaler, but an innovative DCO (digitally controlled oscillator) which allows generating all baudrates from the system clock within a certain range. There is no special clock frequency needed for that purpose so that you're free to choose the system clock for the iniUART, which simplifies considerably the clock structure.

To configurate the baudrate, the 16bit value is calculated according the following formula:

$$Baudrate = \frac{f_{clk}}{16}\frac{n}{2^{18}} \quad \text{respective} \quad n = \frac{2^{18}}{f_{clk}} 16 Baudrate$$

where *Baudrate* is the transmitting speed in bits per second
f_{clk} is the system clock speed in Hz
n is the 16bit value to be programmed

[2] number of ones in a byte, including parity bit is even

Examples:
For 8Mhz clock and 64kbps, n is 33554(dec), accuracy better than 13ppm,
For 10Mhz clock and 1200bps, n is 503(dec), accuracy better than 600ppm.

Limitations:
Values for *n* lower than 100(dec) should not be used, otherwise the accuracy may be below 1%.
e.g. for 8MHz clock, the possible baudrates with accuracy better than 1% range from 190bps to 125kbps.

Accuracy:
The worst case accuracy can be calculated by simply inversing the value *n*. Therefore, a value larger than 100 will guarantee accuracy better than 1%, values larger than 1000 produce results better than 0.1%.

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

2.3 Serial interface

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

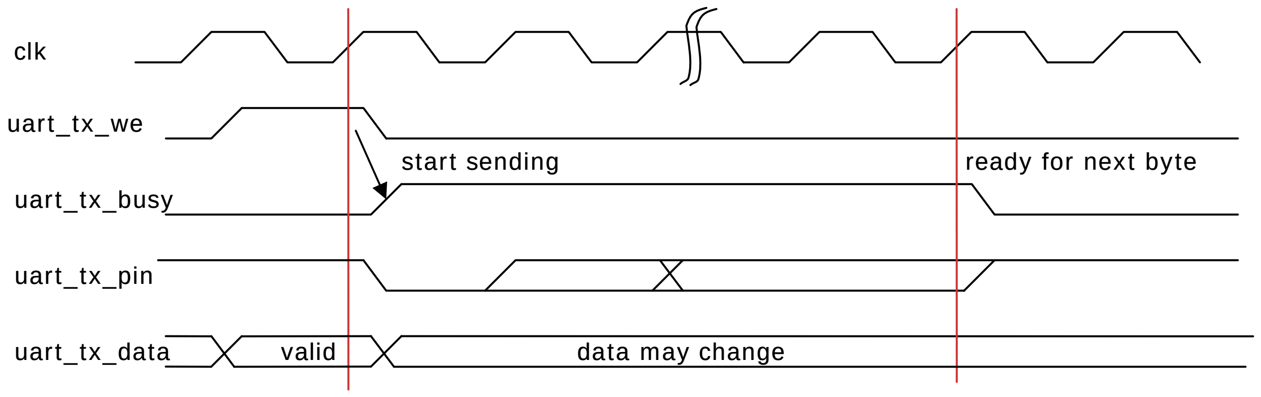
pin name	**type**	**description**
rx_pin	in	Pin for the incoming bit stream. The inactive state is logic ‘1’
tx_pin	out	Pin for the outgoing bit stream. The inactive state is logic ‘1’

3 Transmitter interface

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

pin name	type	description
uart_tx_data[7:0]	in	8bit data to be transmitted. For 7bit configuration, bit[7] is ignored. Data must be valid and stable when uart_tx_we is active.
uart_tx_we	in	Event for storing the tx_data in the transmit shift register and start of transmission. It's up to the system to not activate this input when the iniUART is busy.
uart_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 uart_tx_busy is '1', uart_tx_we mustn't be activated.

The following diagram shows a typical case:



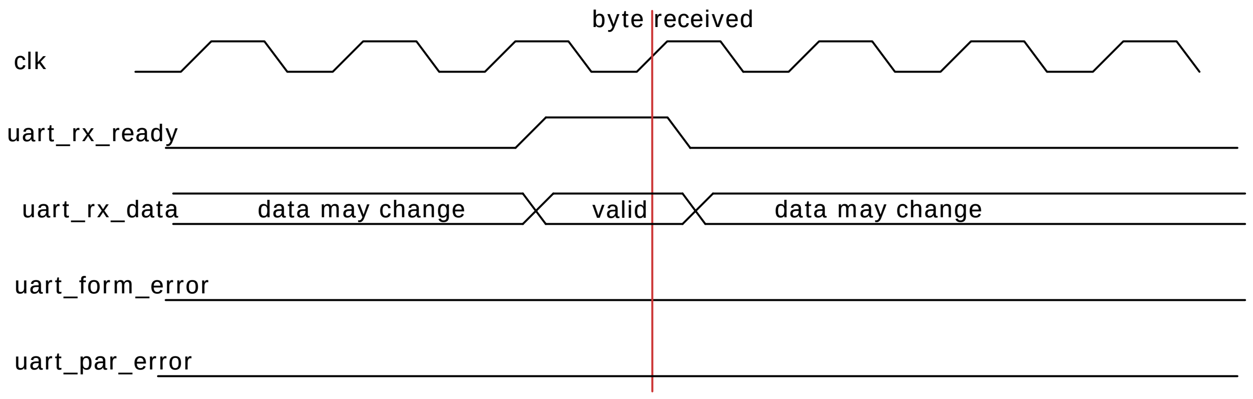
The transmitter path stores the incoming byte in the shift register by means of the uart_tx_we signal and starts the transmitting activity. uart_tx_busy goes high also and remains high until the data is sent.

4 Receiver interface

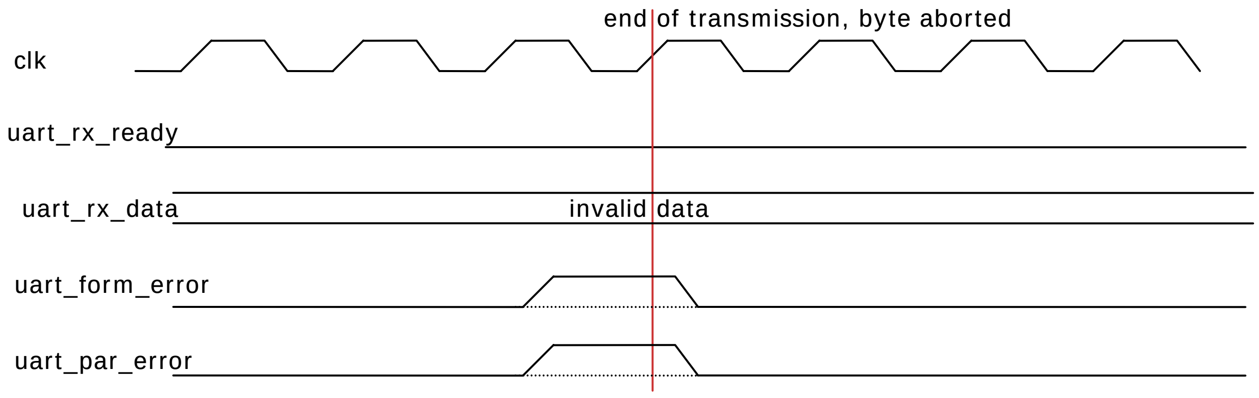
For receiving data, a similar type of interface is used as in the transmitter path.

pin name	type	description
uart_rx_data[7:0]	out	8bit data that has been received. For 7bit configuration, bit[7] is ignored. The data will be stable only during the active phase of uart_rx_ready. Add a buffer register if data should remain stable until reception of next character.
uart_rx_ready	out	Event (active '1') for signalling, that a new byte has arrived and the uart_rx_data is valid now.
uart_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.
uart_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...



... and when a parity or form error occurs



The receiver path contains several checks and special features. First, the level at the uart_rx_pin is watched. When a falling edge is detected, the receiver is started. A reception is started only when the start bit after a falling edge is detected low. If parity is enabled, it is checked and eventual failures are reported on uart_par_error. Missing stop bits (level not zero) are reported as format checks. In all error cases, the data byte is aborted and the error reason is reported. Please note that uart_rx_ready is not asserted when error reporting is done.