INICORE

Datasheet

UARTMODULE

Revision 2.8.1

INICORE INC.
5600 Mowry School Road
Suite 180
Newark, CA 94560
t: 510 445 1529 *f:* 510 656 0995 *e:* info@inicore.com
www.inicore.com

Table of Contents

1 INTRODUCTION....4

1.1 Features....4

1.2 Implementation Features....4

1.2.1 Deliverables....5

1.3 Block Diagram....5

1.4 Block Description....6

1.4.1 System Bus Interface (UPI)....6

1.4.2 Interrupt Controller....6

1.4.3 UART....6

1.4.4 Rx- and Tx-FIFO....6

2 SIGNAL DESCRIPTIONS....7

2.1 I/O Ports....7

2.2 I/O Description....8

2.2.1 Global Signals....8

2.2.2 System Bus Interface....8

2.2.3 UART Interface....8

3 PROGRAMMING MODEL....9

3.1 Memory Mapping....9

3.2 Register Description....10

3.2.1 Receive Data....10

3.2.2 Transmit Data....10

3.2.3 Configuration....11

3.3 Command And Status Interface....13

3.4 Interrupt Controller....14

4 TIMING DIAGRAM.....18

4.1 Local Bus Interface.....18

Table of Figures

Figure 1: Block Diagram UARTmodule.....5
Figure 2: Symbol with I/Os.....7
Figure 3: Block diagram receive path.....10
Figure 4: Block diagram transmit path.....10
Figure 5: Local Bus Interface Timing.....18

1 Introduction

The UARTmodule contains one UART channel, an interrupt controller as well as a local bus interface. The built-in receive and transmit buffers are configurable in depth, therefore, enabling a gate optimized implementation.

This is a fully synchronous design implemented in technology independent VHDL and Verilog RTL.

1.1 Features

Following special features are available:

- Single UART channel
- On-chip interrupt controller
- Receive data FIFO
- Transmit data FIFO
- Edge triggered interrupt source
- Receive timeout counter
- Local loop back mode
- Synchronous bus interface
  - Zero wait-states
  - Supports system bus' such as AMBA APB version 2.0
- Technology independent

1.2 Implementation Features

- Supports register and SRAM based FIFO implementations.
- The Rx- and Tx-FIFO size is selectable enabling the optimal usage of the available gates and optimize system performance.
- Fully synchronous design: one clock, one global asynchronous reset.

- Technology independent implementation[1]. This design can be used for any technology as long as timing constraints are observed.

1.2.1 Deliverables

- VHDL or Verilog RTL source code
- Simulation testbench
- Timing constraints file
- Synthesis script
- User Guide

1.3 Block Diagram

The main building block and interface signals of the UARTmodule are shown in the block diagram below:

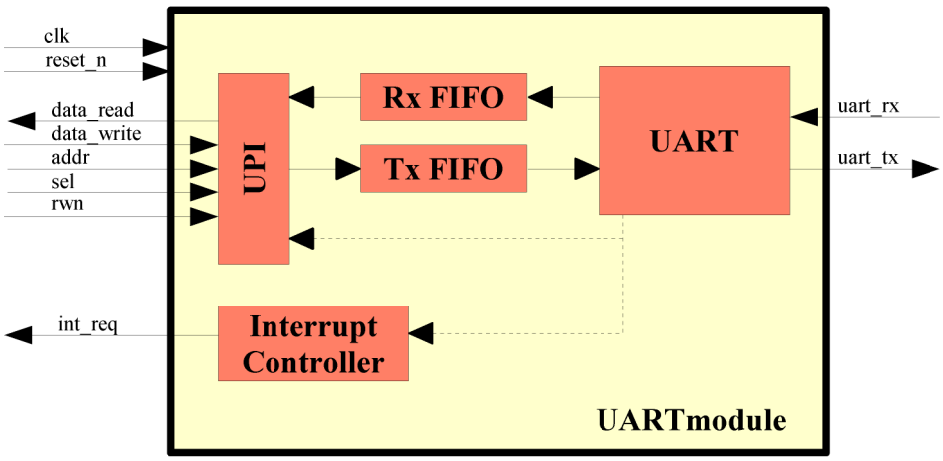


Figure 1: Block Diagram UARTmodule

1 If using register based FIFOs

1.4 Block Description

1.4.1 System Bus Interface (UPI)

A fully synchronous single clock cycle bus interface provides direct access to all local register resources. See the memory map for detailed description of all registers.

1.4.2 Interrupt Controller

The interrupt controller bundles all local interrupt sources together and provides one interrupt request line to the main system.

1.4.3 UART

This is the low level protocol controller. It oversees the generation of the low-level UART signal generation and observes the proper interface timing. It implements the serial-parallel and parallel-serial conversion.

1.4.4 Rx- and Tx-FIFO

To reduce the interrupt overhead for the CPU, a receive and a transmit FIFO with selectable FIFO depth is available. The size of the FIFO's can be selected prior to synthesis of the design and enables an optimal usage of the available resources. On-chip SRAM may be used for the FIFOs as well.

2 Signal Descriptions

2.1 I/O Ports

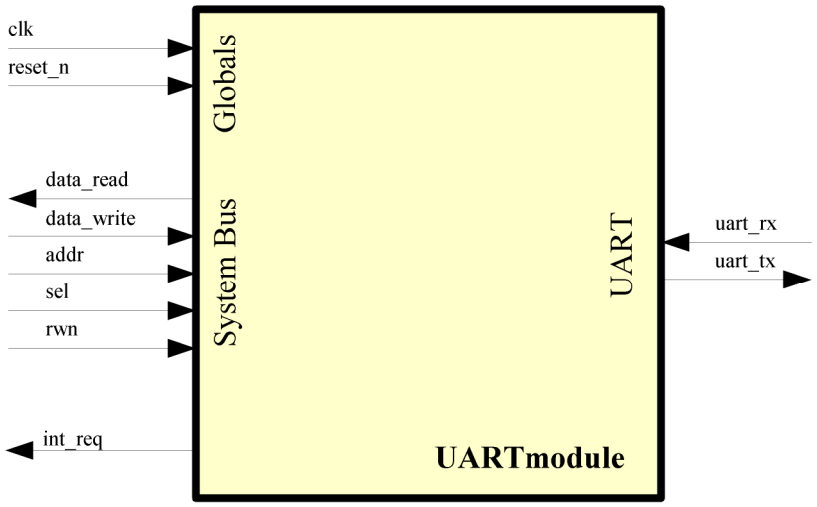


Figure 2: Symbol with I/Os

2.2 I/O Description

The following paragraphs list the inputs and outputs of the UARTmodule and provides an overview of their functionality.

2.2.1 Global Signals

All registers in this core are reset with the asynchronous active low reset_n. All registers are clocked with clk. There are no other local clock or asynchronous reset signals.

Pin Name	*Type*	*Description*
reset_n	in	Asynchronous reset, active low
clk	in	System clock

2.2.2 System Bus Interface

The local bus interface supports 0 wait-state access. See paragraph 4.3.1 for timing information.

Pin Name	*Type*	*Description*
data_read[7:0]	out	Data read bus
data_write[7:0]	in	Data write bus
addr[4:0]	in	Address bus
sel	in	Core select signal, active high
rwn	in	Read/write control signal '0': Write '1': Read
int_req	out	Interrupt request

2.2.3 UART Interface

This is the UART bus interface to the external slave devices.

Pin Name	*Type*	*Description*
uart_tx	out	UART transmit
uart_rx	in	UART receive

3 Programming Model

3.1 Memory Mapping

The following table shows the memory mapping of the UARTmodule:

Address	*Name*	*R/W*	*Description*
UART			
0x00	rx_data	R	Receive data register
0x01	tx_data	W	Transmit data register
0x02	baudrate_lb	rW	Baudrate, low byte
0x03	baudrate_hb	r/W	Baudrate, high byte
0x04	config	r/W	Configuration register
0x05	command	W	Command register
	status	R	Status register
Interrupt Controller			
0x10	int_status	R/W	Interrupt status register
0x11	int_ebl	r/W	Interrupt enable register
0x12	int_ack	W	Interrupt acknowledgment

Following acronyms are used in the table above:

r: Data readback

R: Data read

W: Data write

3.2 Register Description

3.2.1 Receive Data

Incoming data is stored in a register based FIFO. The FIFO depth can be selected prior to synthesis.

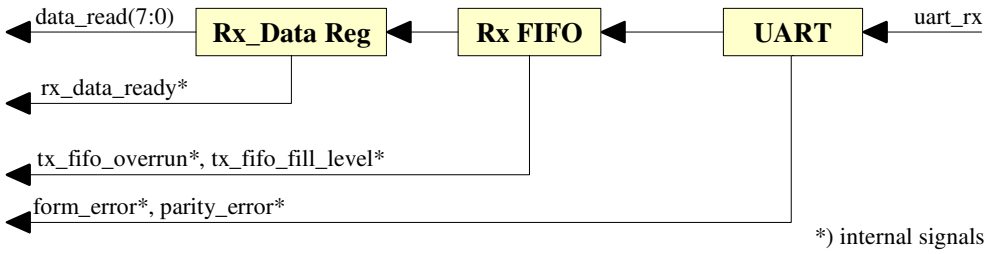


Figure 3: Block diagram receive path

Register Description:

Address	*Name*	*R/W*	*Description*
+0x00	rx_data	R	Receive data register Reading the Rx_Data register automatically acknowledges rx_data_ready flag.

3.2.2 Transmit Data

The transmit buffer uses a register based FIFO. Data written to the FIFIO can be continues, one byte per clock cycle until the FIFO is full. There is no need for an idle clock between access. The FIFO depth must be selected prior to synthesis.

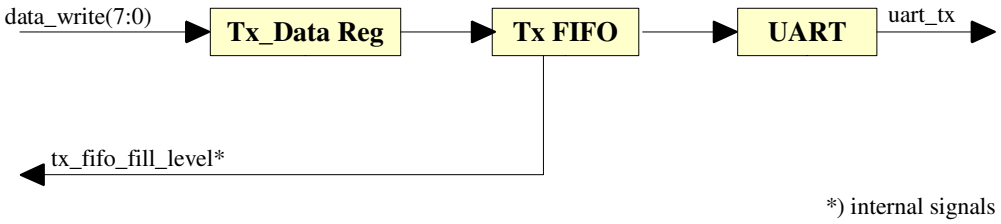


Figure 4: Block diagram transmit path

Register Description:

Address	Name	R/W	Description
+0x01	tx_data	W	Transmit data register Writing a new byte to this register automatically sets the transmit request for the UART.

3.2.3 Configuration

Several registers control the operation of the UART.

Register Description:

Address	Name	R/W	Description
+0x02	baudrate_lb	r/W	Baudrate, low byte
+0x03	baudrate_hb	r/W	Baudrate, high byte
+0x04	Config	r/W	UART configuration [0]: tx enable: '0': transmit disabled '1': enables the transmit path [1]: rx enable: '0': receive disabled '1': enables the receive path [2]: data_78: '0': 7-bit frames '1': 8-bit frames [3]: parity_ebl: '0': parity bit disabled '1': parity bit enabled [4]: parity: '0': even parity, number of ones including parity bit is even '1': odd parity, number of ones including parity bit is odd. [5]: stop-bits: '0': 1 stop bit '1': 2 stop bits

Address	Name	R/W	Description
+0x04	Config	r/W	UART configuration *(continued)* [7:6]: fifo_level: Rx FIFO fill-level: 0: more than 25% of FIFO filled 1: more than 50% of FIFO filled 2: more than 75% of FIFO filled 3: 100% of FIFO filled Tx FIFO fill-level: 0: more than 75% of FIFO filled 1: more than 50% of FIFO filled 2: more than 25% FIFO filled 3: empty

Baudrate Generator:

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 configuring the baudrate, the 16-bit value is calculated according the following formula:

$$Baudrate = \frac{f_{clk}}{16}\frac{n}{2^{17}} \quad \textit{respective} \quad n = \frac{2^{17}}{f_{clk}}16\,Baudrate$$

where

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

Example:

- For 8 Mhz clock and 64 kbps, *n* is 16777(dec)
- For 10 Mhz clock and 1200 bps, *n* is 252(dec)

Note:

- The higher the baudrate, the better is the accuracy.
- The higher the clock, the smaller is the edge jitter. Maximum absolute jitter is always equal 1/fclk.

3.3 Command And Status Interface

These registers are used to indicate and acknowledge additional UART events. All error flags are sticky bits, meaning they remain set until they are cleared by writing a '1' to their respective position.

Register Description:

The table shown on the next page shows the command/status interface.

Address	*Name*	*R/W*	*Description*
+0x05	command	W	Command Register [0]: Reset Tx FIFO '0': No Action 1': Reset Tx FIFO controller [1]: Reset Rx FIFO 0': No Action 1': Reset Tx FIFO controller [2]: Acknowledge parity error 0': No action 1': Erases parity error flag [3]: Acknowledge form error 0': No action 1': Erases form error flag [4]: Acknowledge FIFO overrun error '0': No action '1': Erases FIFO overrun error flag [5]: n/a [6]: Set local loopback test mode '0': No action '1': Set loopback test mode [7]: Clear loopback test mode '0': No action '1': Clear loopback test mode

Address	Name	R/W	Description
	status	R	Status Registers [0]: Tx_Data buffer empty '0': Tx_Data buffer busy '1': Tx_Data buffer is empty and ready for new data [1]: rx_data_ready '0': Rx data buffer is empty '1': Rx data buffer contains a valid data byte [2]: Parity error '0': Normal operation '1': Byte with wrong parity has been received and discarded [3]: Form error '0': Normal operation '1': Byte with wrong format was received and discarded [4]: FIFO overrun error '0': Normal operation '1': A FIFO overrun happened and the new byte has been discarded [5]: Tx FIFO full '0': Tx FIFO not full '1': Tx FIFO full [6]: Loopback mode '0': Normal operation '1': transmit data is looped back to the receive pin

3.4 Interrupt Controller

The interrupt controller issues edge triggered interrupt requests to an external interrupt controller. Before a new interrupt request can happen the current interrupt request has to be acknowledged by a CPU write access to int_ack (Data is don't care).

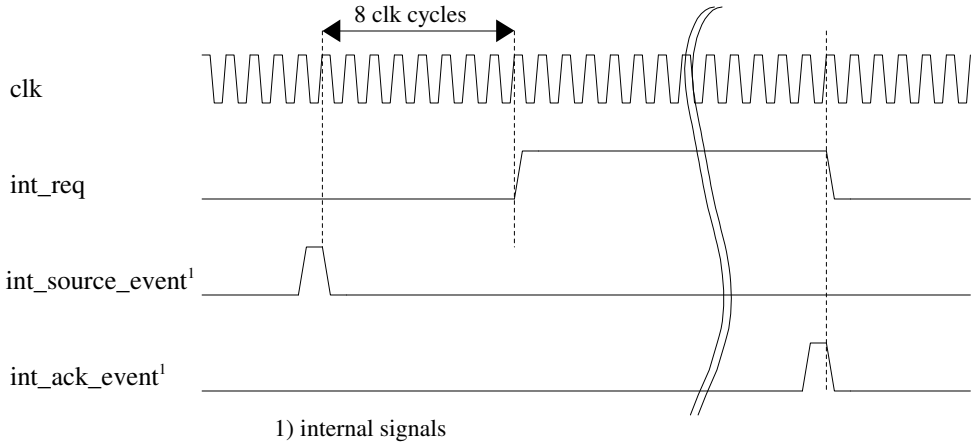
A new interrupt request is generated if:

- a new interrupt source event happens
  and
- the respective interrupt source is enabled
  and
- the previous interrupt request is acknowledged

Independent of the interrupt enable flag and a pending interrupt request, an interrupt source event sets its int_status flag. All interrupt status flags are sticky bits, meaning they remain set until they are cleared by writing a '1' to their respective position.

Interrupt Request Line

The idle state of the int_req line is low. Upon an interrupt source event (int_source_event), the interrupt request line remains low for 8 clock cycles before it is asserted high. It remains high until the interrupt is acknowledged through writing to the int_ack register (int_ack_event).



1) internal signals

Timeout Interrupt

To avoid situations where a simple character remains in the receive FIFO when the entire receive data handling is interrupt driven, a receive timeout counter is used.

The timeout counter is reset by following events:

- When new data is received from the UART
- When RxData register is read
- When Rx FIFO is empty

A timeout is set to 44 bit-times. This is independent on the number of stop-bits, character length and parity enable. The bit-time is configured through the baudrate configuration with:

$$Bit-time=\frac{1}{baudrate}$$

The timeout counter is disabled when rx_enable='0'. So there are no timeout interrupts when no data is read.

Register Description:

Address	Name	R/W	Description
+0x010	int_status	R/W	Interrupt Status Register [0]: rx_timeout '0': No interrupt pending '1': Receive timeout [1]: rx_data_available '0': No interrupt pending '1': Rx FIFO contains at least *n* messages as set in the rx_level configuration register [2]: rx_error '0': No interrupt pending '1': A rx_error happened. Check UART status register for more information. [3]: tx_data_sent '0': No interrupt pending '1': Indicates that one byte was sent [4]: tx_fifo_empty '0': No interrupt pending '1': TxFIFO is empty [5]: tx_data_available '0': No interrupt pending '1': FIFO has space for *n* messages as set in tx_level configuration register
+0x11	int_ebl	r/W	Interrupt Enable Register [0]: rx_timeout '0': Interrupt disabled '1': Interrupt enabled [1]: rx_data_available '0': Interrupt disabled '1': Interrupt enabled

Address	Name	R/W	Description
+0x11	int_ebl	r/W	Interrupt Enable Register *(continued)* [2]: rx_error '0': Interrupt disabled '1': Interrupt enabled [3]: tx_data_sent '0': Interrupt disabled '1': Interrupt enabled [4]: tx_fifo_empty '0': Interrupt disabled '1': Interrupt enabled [5]: tx_data_available '0': Interrupt disabled '1': Interrupt enabled
+0x12	int_ack	W	Interrupt Acknowledgment Writing to this register acknowledges a pending interrupt request. Data is don't care. This doesn't acknowledge the respective int_status flag.

Note:

- The rx_data_available is a sticky bit. Once set it only can be reset by the CPU writing a '1' to it. It is re-evaluated with one of the following events:
    - a new data word is received and the Rx FIFO fill level condition is met
    - RxData register is read by the CPU
    - rx_data_available flag is reset by the CPU
- The tx_fifo_empty interrupt is only set once the FIFO gets empty. If the CPU acknowledges the interrupt but doesn't load the FIFO, there won't be a second interrupt. After reset, the system generates a tx_fifo_empty interrupt. This enables an interrupt service routine to be called automatically without any special startup code.
- The rx_error flag is set when one of the following status bits is set:
    - Parity Error
    - Form Error
    - FIFO Overrun Error

4 Timing Diagram

Timing numbers depend on the chosen device, synthesis options, place&route constraints.

4.1 Local Bus Interface

The microprocessor interface is designed to operate on a full synchronous bus with zero wait-states. The internal registers are selected using sel, rwn and addr. The selected register is written on the rising edge of the system clock. data_read is asynchronously generated.

Following figures shows the interface timing diagram.

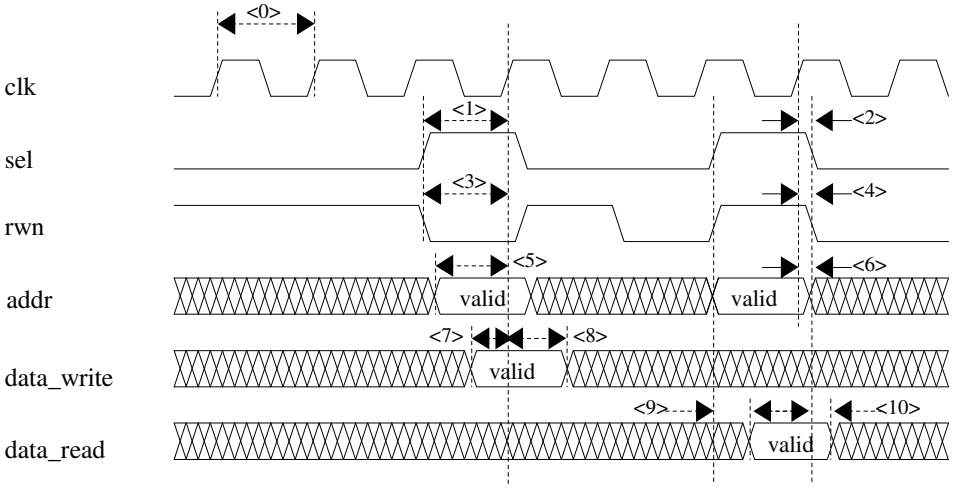


Figure 5: Local Bus Interface Timing

The timing numbers are technology dependent and can only be given once a target technology is selected.

Time nr	***t_{min} [ns]***	***t_{typ} [ns]***	***t_{max} [ns]***	***Comment***
<0>	tbd			Clock period
<1>		tbd		sel setup time
<2>		0		sel hold time
<3>		tbd		rwn setup time
<4>		0		rwn hold time
<5>		tbd		addr setup time
<6>		0		addr hold time
<7>		tbd		data_write setup time
<8>		0		data_write hold time
<9>		tbd		data_read valid after sel, rwn, addr valid
<10>		0		data_read hold time after sel, rwn, addr invalid

About Inicore

- FPGA and ASIC Design
- Easy-to-use IP Cores
- System-on-Chip Solutions
- Consulting Services
- ASIC to FPGA Migration
- Obsolete ASIC Replacements

Inicore is an experienced system design house providing FPGA / ASIC and SoC design services. The company's expertise in architecture, intellectual property, methodology and tool handling provides a complete design environment that helps customers shorten their design cycle and speed time to market. Our offering covers feasibility study, concept analysis, architecture definition, code generation and implementation. When ready, we deliver you a FPGA or take your design to an ASIC provider, whatever is more suitable for your unique solution.

Customer Advantages

We offer one-stop shopping for everything from the specifications to the chip or module solution. Our experience and fast turnaround time reduces your development costs and increases your returns from the market. Your system is not limited by the level of expertise and standard chip solutions you happen to have in-house. Achieve market success by differentiating and optimizing your product. Reusability builds the basis for further developments in the ever-decreasing product life cycle.

Visit us @ www.inicore.com