IOB-UART, a RISC-V UART

User Guide, V0.1 , Build b5ea573

April 27, 2022

Contents

List of Tables

List of Figures

1 Introduction

The IObundle UART is a RISC-V-based Peripheral written in Verilog, which users can download for free, modify, simulate and implement in FPGA or ASIC. It is written in Verilog and includes a C software driver. The IObundle UART is a very compact IP that works at high clock rates if needed. It supports full-duplex operation and a configurable baud rate. The IObundle UART has a fixed configuration for the Start and Stop bits. More flexible licensable commercial versions are available upon request.

2 Symbol

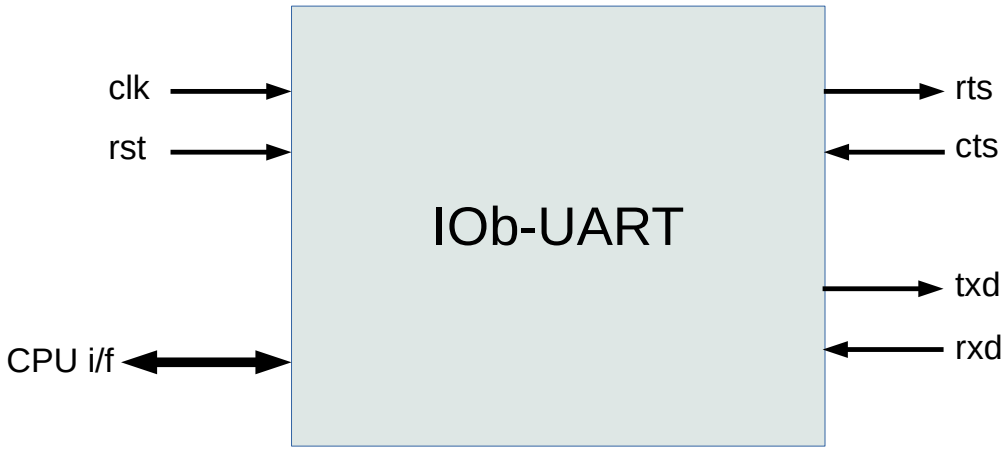

Figure 1: IP core symbol.

3 Features

- Supported in IObundle's RISC-V IOb-SoC open-source and free of charge template.
- IObundle's IOb-SoC native CPU interface.
- Verilog basic UART implementation.
- Soft reset and enable functions.
- Runtime configurable baud rate
- C software driver at the bare-metal level.
- Simple Verilog testbench for the IP's *nucleus*.
- System-level Verilog testbench available when simulating the IP embedded in IOb-SoC.
- Simulation Makefile for the open-source and free of charge Icarus Verilog simulator.
- FPGA synthesis and implementation scripts for two FPGA families from two FPGA vendors.
- Automated creation of FPGA netlists

- Automated production of documentation using the open-source and free Latex framework.
- IP data automatically extracted from FPGA tool logs to include in documents.
- Makefile tree for full automation of simulation, FPGA implementation and document production.
- AXI4 Lite CPU interface (premium option).
- Parity bits (premium option).

4 Benefits

- Compact and easy to integrate hardware and software implementation
- Can fit many instances in low cost FPGAs and ASICs
- Low power consumption

5 Deliverables

- ASIC or FPGA synthesized netlist or Verilog source code, and respective synthesis and implementation scripts
- ASIC or FPGA verification environment by simulation and emulation
- Bare-metal software driver and example user software
- User documentation for easy system integration
- Example integration in IOb-SoC (optional)

6 Block Diagram and Description

Figure 2 presents a high-level block diagram of the core, followed by a brief description of each block.

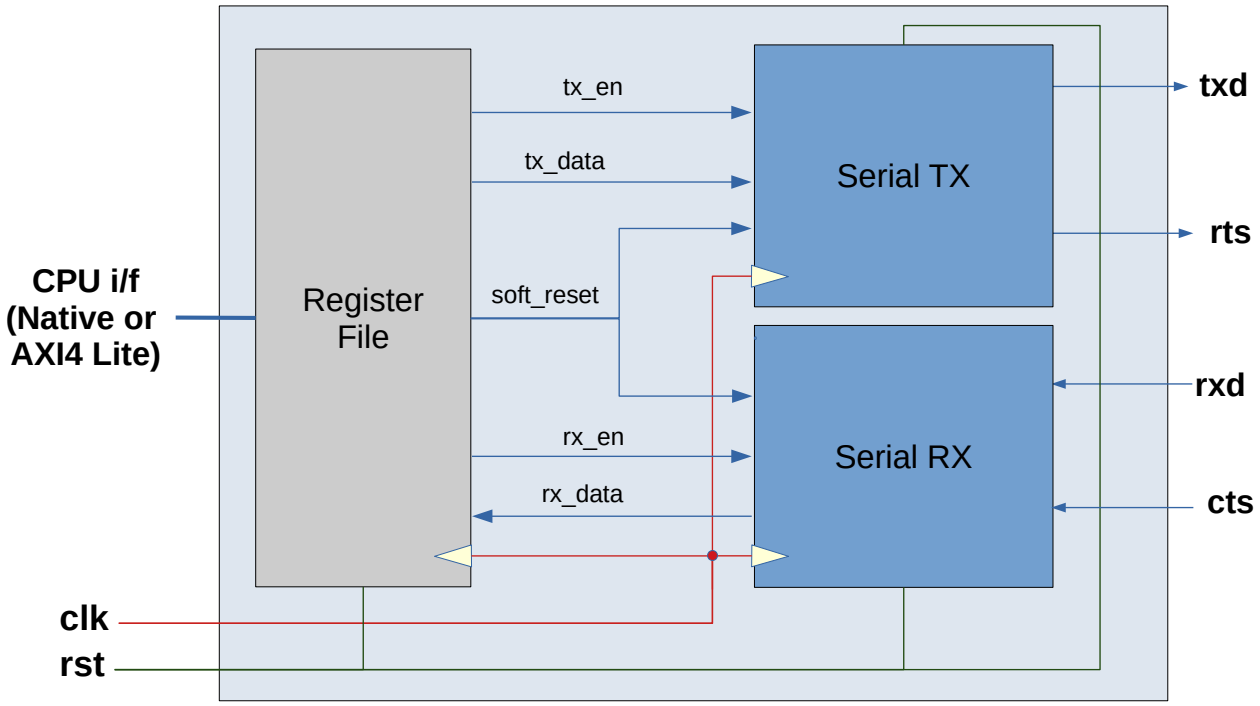

Figure 2: High-level block diagram.

REGISTER FILE Configuration control and status register file.

7 Interface Signals

Name	Direction	Width	Description
clk	INPUT	1	System clock input
rst	INPUT	1	System reset, asynchronous and active high

Table 1: General interface signals.

Name	Direction	Width	Description
valid	INPUT	1	Native CPU interface valid signal
address	INPUT	ADDR_W	Native CPU interface address signal
wdata	INPUT	DATA_W	Native CPU interface data write signal
wstrb	INPUT	DATA_W/8	Native CPU interface write strobe signal
rdata	OUTPUT	DATA_W	Native CPU interface read data signal
ready	OUTPUT	1	Native CPU interface ready signal

Table 2: IObundle Interface Signals

Name	Direction	Width	Description
interrupt	OUTPUT	1	to be done
txd	OUTPUT	1	Serial transmit line
rxd	INPUT	1	Serial receive line
cts	INPUT	1	Clear to send; the destination is ready to receive a transmission sent by the UART
rts	OUTPUT	1	Ready to send; the UART is ready to receive a transmission from the sender.

Table 3: RS232 Interface Signals

8 Software Accessible Registers

The software accessible registers of the core are described in the following tables. The tables give information on the name, read/write capability, word aligned addresses, used word bits, and a textual description.

Name	R/W	Addr	Bits	Initial Value	Description
UART_SOFTRESET	W	0	1	0	Bit duration in system clock cycles.
UART_DIV	W	4	16	0	Bit duration in system clock cycles.
UART_TXDATA	W	8	8	0	TX data
UART_TXEN	W	12	1	0	TX enable.
UART_TXREADY	R	16	1	0	TX ready to receive data
UART_RXDATA	R	20	8	0	RX data
UART_RXEN	W	24	1	0	RX enable.
UART_RXREADY	R	28	1	0	RX data is ready to be read.

Table 4: UART software accessible registers.

9 Instantiation and External Circuitry

Figure 4 illustrates how to instantiate the IP core and, if applicable, the required external blocks. A Verilog instantiation template is provided for convenience.

bla bla bla...

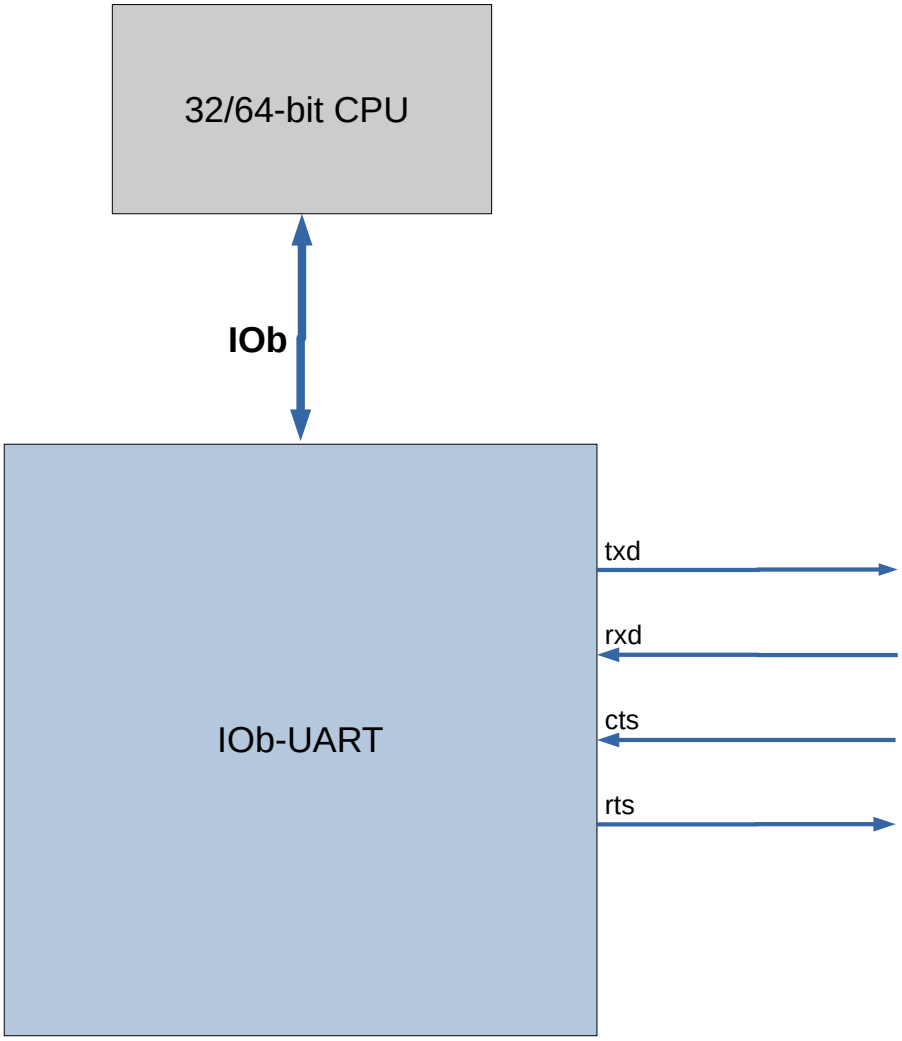

Figure 3: Core instance and required surrounding blocks

10 Simulation

The provided testbench uses the core instance described in Section 9. A high-level block diagram of the testbench is shown in Figure 4. The testbench is organized in a modular fashion, with each test described in a separate file. The test suite consists of all the test case files to make adding, modifying, or removing tests easy.

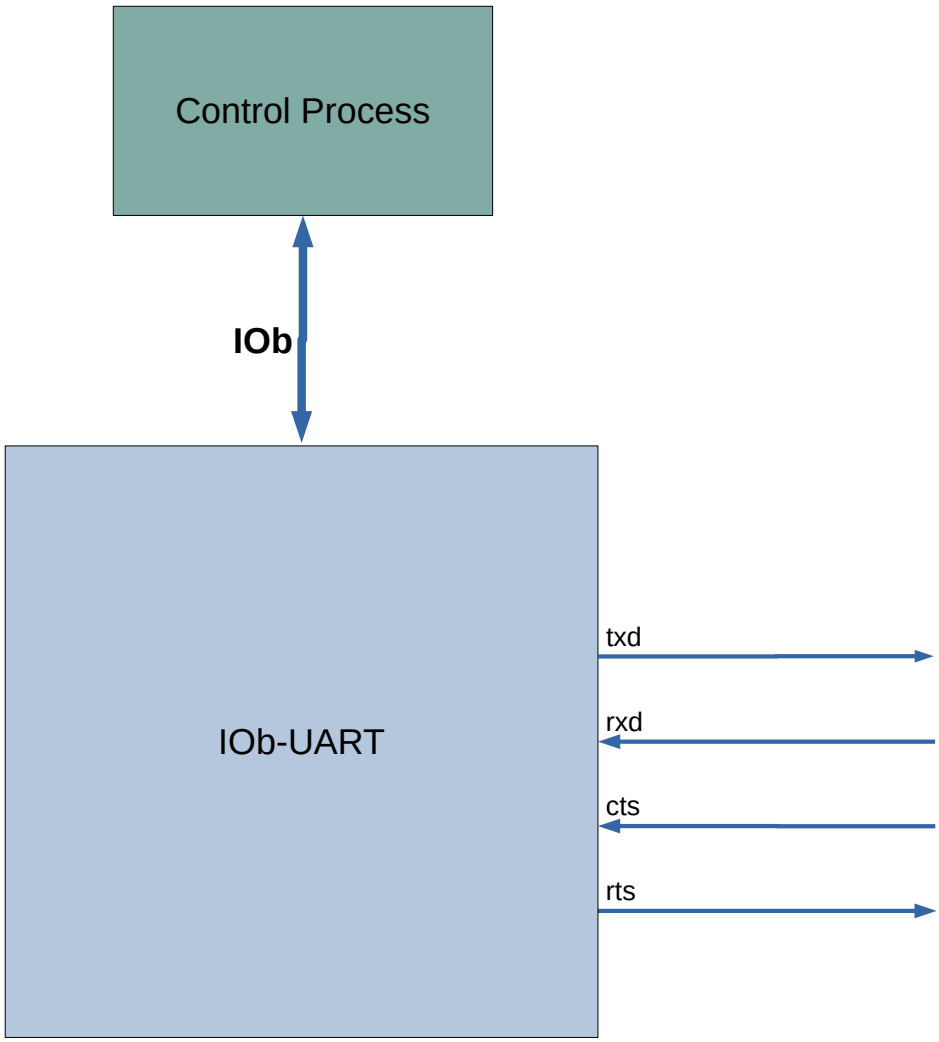

Figure 4: Testbench block diagram

In this preliminary version, simulation is fully functional. The provided testbench drives the clock and reset signals as well as the rxd signal.

11 Synthesis

11.1 Synthesis Macros

The IP core has no user-definable macros other than the default synthesis parameters values.

11.2 Synthesis Parameters

This IP core has no synthesis parameters.

11.3 Synthesis Script and Timing Constraints

A simple `.tcl` script is provided for the Cadence Genus synthesis tool. The script reads the technology files, compiles and elaborates the design, and proceeds to synthesise it. The timing constraints are contained within the constraints file provided, or provided in a separate file.

After synthesis, reports on silicon area usage, power consumption, and timing closure are generated. A post-synthesis Verilog file is created, to be used in post-synthesis simulation.

In this preliminary version, synthesis of the IP core is functional.