IOB-UART

A RISC-V UART

June 1, 2022

User Guide, V0.1 , Build 3d83996

The following table shows the revision history for this document.

Date	Summary
June 1, 2022	Initial document version V0.1 .

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.

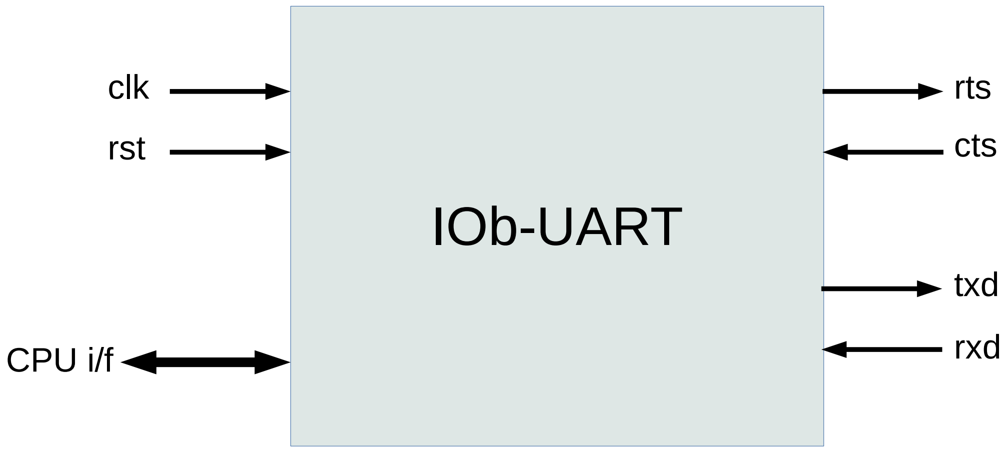

Figure 1: IP Core Symbol.

1.1 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).

1.2 Benefits

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

1.3 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)

2 Description

2.1 Block Diagram

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

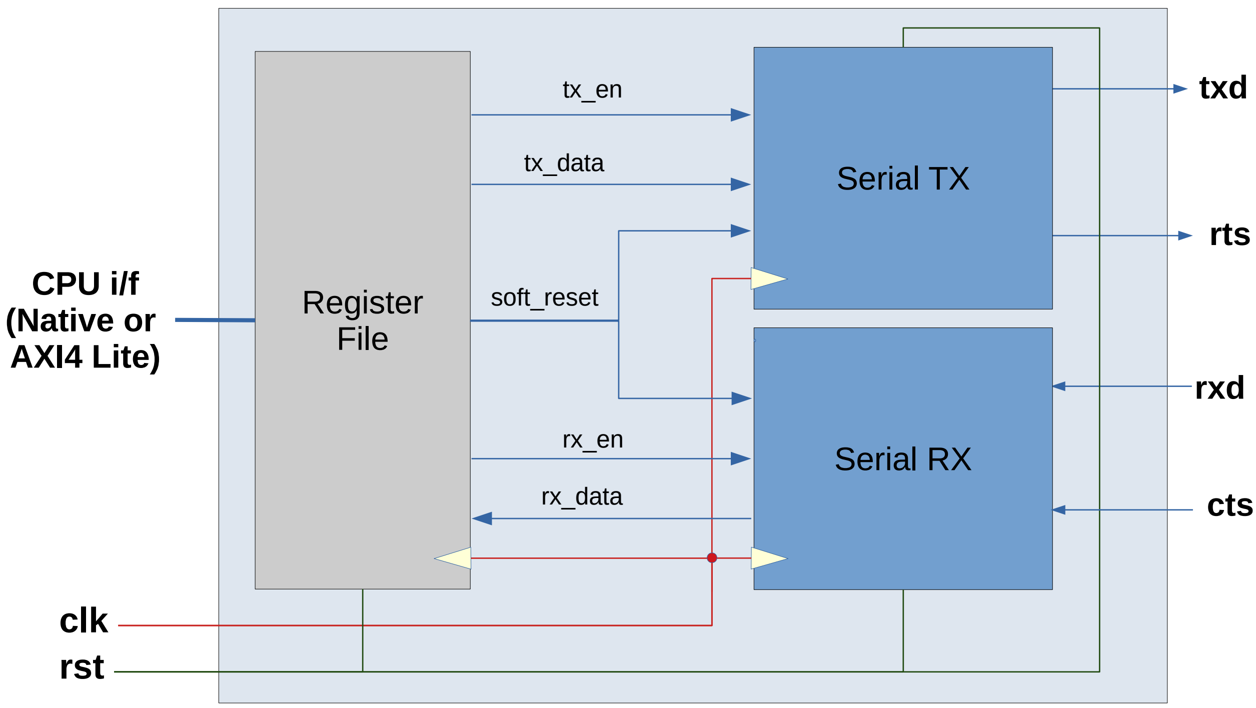

Figure 2: High-Level Block Diagram.

REGISTER FILE Configuration control and status register file.

2.2 Configuration

2.2.1 Macros

The IP core has no user-definable macros other than the default configuration parameter values, described in Section 2.2.2 if available.

2.2.2 Parameters

The configuration parameters of the core are presented in Table 1. Configuration parameters can vary from instance to instance.

Parameter	Min	Typ	Max	Description
DATA_W	32	32	64	CPU data width

Table 1: Synthesis Parameters.

2.3 Interface Signals

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

Table 2: 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 3: 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 4: RS232 Interface Signals

2.4 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	8	0	Bit duration in system clock cycles.
UART_DIV	W	2	16	0	Bit duration in system clock cycles.
UART_TXDATA	W	4	8	0	TX data
UART_TXEN	W	5	8	0	TX enable.
UART_TXREADY	R	0	8	0	TX ready to receive data
UART_RXDATA	R	4	8	0	RX data
UART_RXEN	W	6	8	0	RX enable.
UART_RXREADY	R	1	8	0	RX data is ready to be read.

Table 5: UART software accessible registers.

3 Usage

Figure 4 illustrates how to instantiate the IP core and, if applicable, the required external blocks.

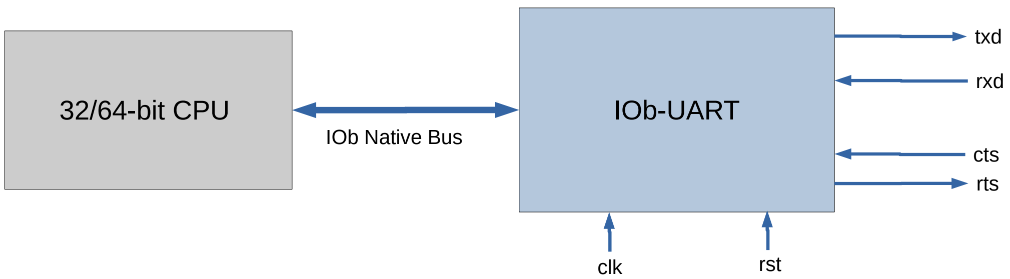

Figure 3: Core Instance and Required Surrounding Blocks

The IOb-UART is a fully synchronous, single clock (`clk`) domain design with an asynchronous active-high reset signal (`arst`) that drives all the flip-flops in the design. The reset signal should be de-asserted synchronously with the clock signal's rising edge.

The IOb-UART works attached to a CPU core, using the IOb Native Bus interface. It outputs (pin `txd`) and receives (pin `rxd`) an RS232-encoded serial data stream. The RS232 protocol also specifies and handshking signal pair consisting of the signals `cts` and `rts`.

3.1 Simulation

The provided testbench uses the core instance described in Section 3. 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.

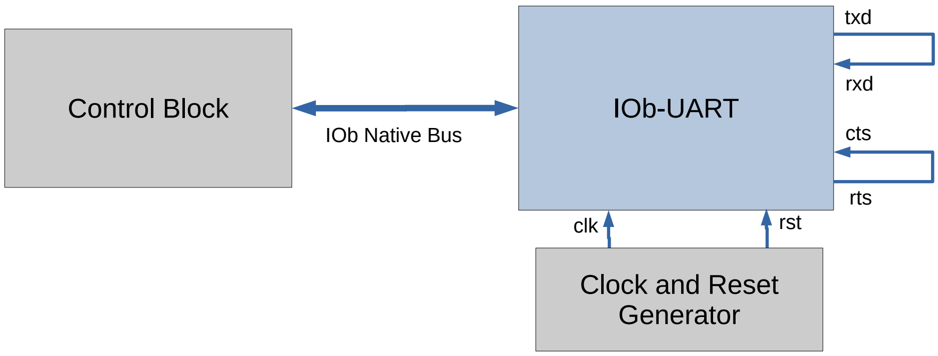

Figure 4: Testbench Block Diagram

The provided testbench implements a self-loop, where the IOb-UART handshakes (`cts` to `rts`, and sends data to itself (`txd` to `rxd`). The testbench drives the clock and reset signals, and emulates the CPU actions with a simple control block.

3.2 Synthesis

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 synthesize 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.

This IP core synthesizes for FPGA and ASIC with very low logic resources consumption. The implementation does not require memory or arithmetic resources.

4 Implementation Results

4.1 FPGA

This section presents FPGA implementation results.

Resource	Used
LUTs	115
Registers	113
DSPs	0
BRAM	0

Table 6: AMD Kintex Ultrascale FPGAs.

Resource	Used
ALM	96
FF	125
DSP	0
BRAM blocks	0

Table 7: Intel Cyclone V GT FPGAs.

4.2 ASIC

No ASIC implementation results have been obtained for this IP core.

