IOb-UART16550

A NS16550A-compatible UART IP Core

January 12, 2026

User Guide, V0.1, Build 0ec68b4

Document Version History

Version	Date	Person	Changes from previous version
0.1	January 12, 2026	AN	Initial document version

Contents

List of Tables

List of Figures

1 Introduction

The UART (Universal Asynchronous Receiver/Transmitter) core provides serial communication capabilities, which allow communication with modem or other external devices, like another computer using a serial cable and RS232 protocol. This core is designed to be maximally compatible with the industry standard National Semiconductors' 16550A device.

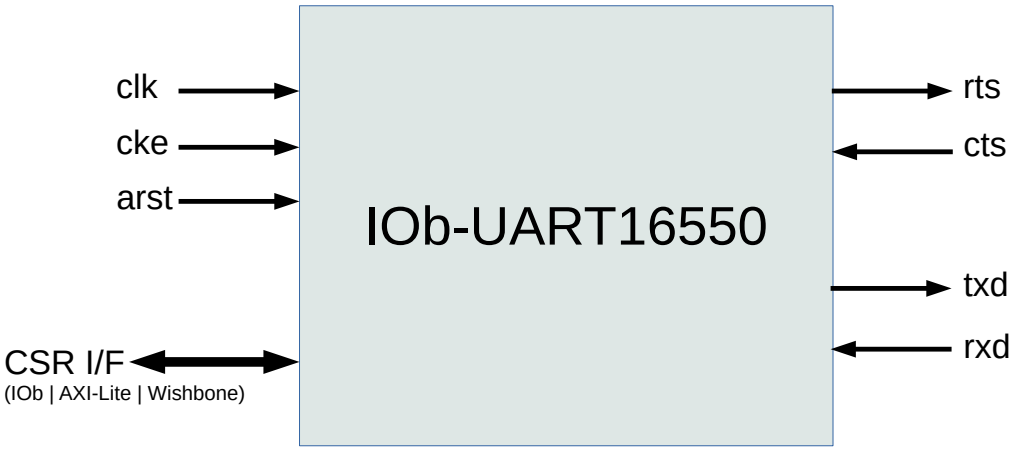

Figure 1: IP Core Symbol

1.1 Features

- FIFO only operation
- Register level and functionality compatibility with NS16550A.
- Debug Interface in 32-bit data bus mode.
- Support for multiple CSR interface types (selectable): IOb, AXI-Lite, WISHBONE

1.2 Deliverables

- Verilog RTL source code synthesizable for ASIC and FPGA
- Verilog testbench and simulation scripts for code coverage
- ASIC synthesis script and timing constraints
- FPGA synthesis scripts and timing constraints
- Bare-metal software driver and example user firmware
- Comprehensive user guide

2 Description

This section gives a detailed description of the IP core. The high-level block diagram is presented, along with a description of its subblocks. The parameters and macros that define the core configuration are listed and

explained. The interface signals are enumerated and described; if timing diagrams are needed, they are shown after the interface signals. Finally the Control and Status Registers (CSR) are outlined and explained.

2.1 Block Diagram

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

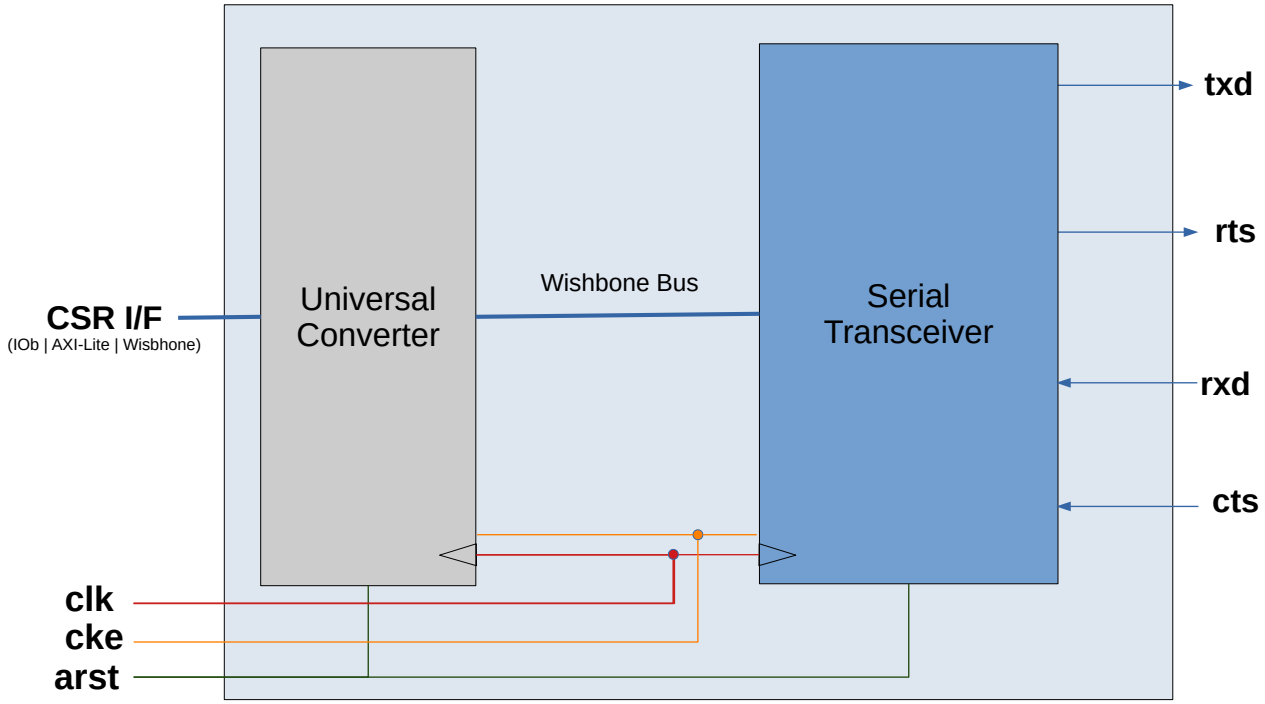

Figure 2: High-Level Block Diagram

The Verilog modules in the top-level entity of the core are described in the following tables. The table elements represent the subblocks in the Block Diagram.

Name	Description
universal_converter	Convert CSRs interface into internal wishbone bus
serial_transceiver	UART16550 Serial Transceiver

Table 1: Table of subblocks in the core.

2.2 Configuration

The following tables describe the IP core configuration. The core may be configured using macros or parameters:

'M' Macro: a Verilog macro or `define` directive is used to include or exclude code segments, to create core configurations that are valid for all instances of the core.

'P' Parameter: a Verilog parameter is passed to each instance of the core and defines the configuration of that particular instance.

Configuration	Type	Min	Typical	Max	Description
ADDR_W	P	NA	5	NA	Address bus width
DATA_W	P	NA	32	NA	Data bus width

Table 2: General operation group

The macros not listed above are constants. They improve the code readability and should not be changed by the user. These constants are listed below:

VERSION Product version. This 16-bit macro uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part. Value: 16'h0001 = V0.01.

2.3 Interface Signals

The interface signals of the core are described in the following tables. Note that the ouput signals are registered in the core, while the input signals are not.

Name	Direction	Width	Description
clk_i	input	1	Clock
cke_i	input	1	Clock enable
arst_i	input	1	Asynchronous active-high reset

Table 3: Clock, clock enable and reset

Name	Direction	Width	Description
iob_valid_i	input	1	Request address is valid.
iob_addr_i	input	5	Byte address.
iob_wdata_i	input	32	Write data.
iob_wstrb_i	input	32/8	Write strobe.
iob_rvalid_o	output	1	Read data valid.
iob_rdata_o	output	32	Read data.
iob_ready_o	output	1	Interface ready.

Table 4: Control and status interface, when selecting the IOb CSR interface.

Name	Direction	Width	Description
wb_dat_o	output	32	Data input.
wb_datout_i	input	32	Data output.
wb_ack_o	output	1	Acknowledge input. Indicates normal termination of a bus cycle.
wb_adr_i	input	5	Address output. Passes binary address.
wb_cyc_i	input	1	Cycle output. Indicates a valid bus cycle.
wb_sel_i	input	32/8	Select output. Indicates where valid data is expected on the data bus.
wb_stb_i	input	1	Strobe output. Indicates valid access.
wb_we_i	input	1	Write enable. Indicates write access.

Table 5: Control and status interface, when selecting the Wishbone CSR interface.

Name	Direction	Width	Description
axil_araddr_i	input	5	AXI-Lite address read channel byte address.
axil_arvalid_i	input	1	AXI-Lite address read channel valid.
axil_arready_o	output	1	AXI-Lite address read channel ready.
axil_rdata_o	output	32	AXI-Lite read channel data.
axil_rresp_o	output	2	AXI-Lite read channel response.
axil_rvalid_o	output	1	AXI-Lite read channel valid.
axil_rready_i	input	1	AXI-Lite read channel ready.
axil_awaddr_i	input	5	AXI-Lite address write channel byte address.
axil_awvalid_i	input	1	AXI-Lite address write channel valid.
axil_awready_o	output	1	AXI-Lite address write channel ready.
axil_wdata_i	input	32	AXI-Lite write channel data.
axil_wstrb_i	input	32/8	AXI-Lite write channel write strobe.
axil_wvalid_i	input	1	AXI-Lite write channel valid.
axil_wready_o	output	1	AXI-Lite write channel ready.
axil_bresp_o	output	2	AXI-Lite write response channel response.
axil_bvalid_o	output	1	AXI-Lite write response channel valid.
axil_bready_i	input	1	AXI-Lite write response channel ready.

Table 6: Control and status interface, when selecting the AXI-Lite CSR interface.

Name	Direction	Width	Description
rs232_rxd_i	input	1	Receive data.
rs232_txd_o	output	1	Transmit data.
rs232_rts_o	output	1	Request to send.
rs232_cts_i	input	1	Clear to send.

Table 7: RS232 interface

Name	Direction	Width	Description
interrupt_o	output	1	UART interrupt source

Table 8: UART16550 interrupt related signals

2.4 Control and Status Registers

The software accessible registers of the core are described in the following tables. The tables give information on the name, read/write capability, address, hardware and software width, and a textual description. The addresses are byte aligned and given in hexadecimal format. The hardware width is the number of bits that the register occupies in the hardware, while the software width is the number of bits that the register occupies in the software. In each address, the right-justified field having "Hw width" bits conveys the relevant information. Each register has only one type of access, either read or write, meaning that reading from a write-only register will produce invalid data or writing to a read-only register will not have any effect.

Name	R/W	Addr	Width		Default	Description
			Hw	Sw		
RBR_THR_DLL	RW	0x0	8	8	0	RBR (Receiver Buffer Register) when read, THR (Transmitter Holding Register) when written. When LCR.DLAB bit is set, this address accesses the Divisor Latch LSB (DLL).
IER_DLM	RW	0x1	8	8	0	Interrupt Enable Register. When LCR.DLAB bit is set, this address accesses the Divisor Latch MSB (DLM).
IIR_FCR	RW	0x2	8	8	193	IIR (Interrupt Identification Register) when read, FCR (FIFO Control Register) when written.
LCR	RW	0x3	8	8	3	Line Control Register. The DLAB bit (MSB) controls access to the Divisor Latch registers.
MCR	W	0x4	8	8	0	Modem Control Register.
LSR	R	0x5	8	8	96	Line Status Register.
MSR	R	0x6	8	8	0	Modem Status Register.
VERSION	R	0x8	16	16	0001	Product version. This 16-bit register uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part. For example V12.34 is represented by 0x1234.

Table 9: General Registers.

3 Usage

3.1 Instantiation

Figure 3 illustrates how to instantiate the IP core and, if applicable, the required external subblocks.

The RS232 interface that should be connected to an external UART (e.g. a USB-to-serial converter).

The CSRs bus (IOb native by default) should be connected to the desired manager component (e.g. a CPU).

The clock, clock enable, and reset ports can be connected to the desired clock and reset generator.

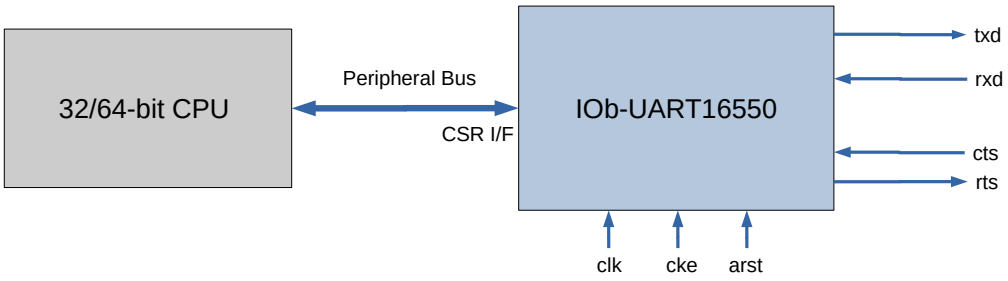

Figure 3: Core Instance and Required Surrounding Blocks

3.2 Simulation

The provided testbench uses the core instance described in Section 3.1. 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.

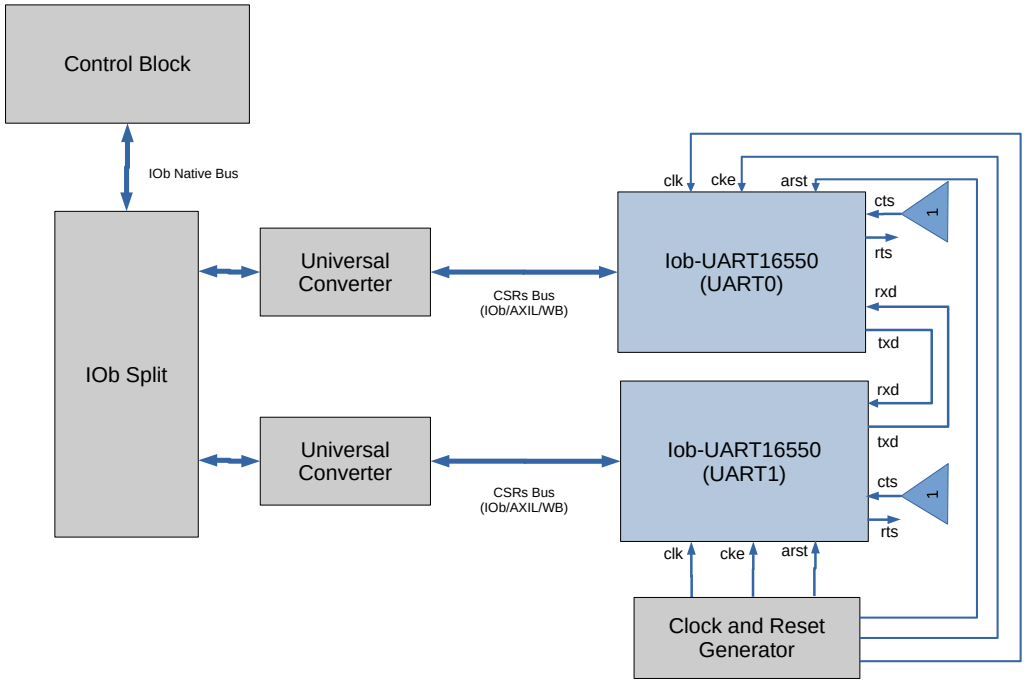

Figure 4: Testbench Block Diagram

The UART16550 testbench is configured to connect the IOb-UART16550 core's RS232 interface in loop-back mode. The testbench architecture involves the following components and data flow:

- Contains two instances of the IOb-UART16550 core: UART0 and UART1.
- Contains two instances of the iob_universal_converter core, one for each UART. The universal converter is used to convert the Testbench's IOb bus to the corresponding UART's CSR bus type.

- Contains a split core to route the testbench commands to the correct UART.

The testbench controller orchestrates the test sequence as follows:

1. Initializes all components.
2. Write test data to UART0 and read back the data via UART1 for comparison.
3. Exercise write registers.
4. Exercise read registers.
5. Try to read highest CSR address.
6. Transfer data between both UARTs and test receive CSRs.

System-level Simulation

Upon request, simulation files to run the core embedded in a RISC-V system can be provided. The core is exercised in various modes by the RISC-V processor, using a bare-metal software program written in the C programming language.

4 Baremetal Drivers

4.1 iob_uart16550.h File Reference

High level iob_uart16550 core functions.

```
   #include <stdarg.h>
#include <stdint.h>
#include <stdlib.h>
```

Macros

- #define **UART_PROGNAME** "IOb-UART"
  Prefix to IOb-UART16550 specific prints.
- #define **STX** 2
  Start text. Signal start of data sequence to be printed.
- #define **ETX** 3
  End text. Signal end of data sequence to be printed.
- #define **EOT** 4
  End of transmission. Signal end of UART16550 connection.
- #define **ENQ** 5
  Enquiry. Signal start of UART16550 connection.

- #define **ACK** 6
  Acknowledge. Signal reception of incoming message.
- #define **FTX** 7
  File transfer. Signal file transfer request.
- #define **FRX** 8
  File reception. Signal file reception request.

Functions

- void uart16550_init (int base_address, uint16_t div)
  Initialize UART16550.
- int uart16550_base (int base_address)
  Change UART16550 base.
- void uart16550_finish ()
  Close transmission.
- char uart16550_txready ()
  Check if TX is ready.
- void uart16550_txwait ()
  Wait for TX.
- char uart16550_rxready ()
  Check if RX is ready.
- void uart16550_rxwait ()
  Wait for RX Data.
- void uart16550_putc (char c)
  Print char.
- void uart16550_puts (const char ∗s)
  Print string.
- void uart16550_sendfile (char ∗file_name, int file_size, char ∗mem)
  Send file.
- char uart16550_getc ()
  Get char.
- int uart16550_recvfile (char ∗file_name, char ∗mem)
  Receive file.

4.1.1 Detailed Description

High level iob_uart16550 core functions.

The present IOb-UART16550 software drivers implement a way to interface with the IOb-UART16550 peripheral for serial communication.

The present drivers provide base functionalities such as:

- initialization and setup
- basic control functions
- single character send and receive functions
- simple protocol for multi byte transfers

4.1.2 Function Documentation

uart16550_base()

```
int uart16550_base (
              int base_address)
```

Change UART16550 base.

Set a new IOb-UART16550 base address.

Returns

Previous base

uart16550_finish()

```
void uart16550_finish ()
```

Close transmission.

Send end of transmission (EOT) command via UART16550. Active wait until TX transfer is complete. Use this function to close console program.

Returns

void.

uart16550_getc()

```
char uart16550_getc ()
```

Get char.

Active wait and receive char/byte from UART16550.

Returns

received byte from UART16550.

uart16550_init()

```
void uart16550_init (
              int base_address,
              uint16_t div)
```

Initialize UART16550.

Reset UART16550, set IOb-UART16550 base address and set the division factor. The division factor is the number of clock cycles per symbol transfered.

For example, for a case with fclk = 100 Mhz for a baudrate of 115200 we should have `div=(100*10^6/115200) = (868).`

Parameters

base_address	IOb-UART16550 instance base address in the system.
div	Equal to round (fclk/baudrate).

Returns

void.

uart16550_putc()

```
void uart16550_putc (
              char c)
```

Print char.

Send character via UART16550 to be printed by in console program.

Parameters

c	Character to print.

Returns

void.

uart16550_puts()

```
void uart16550_puts (
              const char * s)
```

Print string.

Send string via UART16550 to be printed by in console program.

Parameters

s	Pointer to char array to be printed.

Returns

void.

uart16550_recvfile()

```
int uart16550_recvfile (
              char * file_name,
              char * mem)
```

Receive file.

Request variable size file via UART16550. Order of commands:

1. Send file receive (FRX) command.
2. Send file_name.
3. Receive file_size (in little endian format).
4. Send ACK command.
5. Receive file.

If memory pointer is not initialized, allocates memory for incoming file.

Parameters

file_name	Pointer to file name string.
mem	Pointer in memory to store incoming file.

Returns

Size of received file.

uart16550_rxready()

```
char uart16550_rxready ()
```

Check if RX is ready.

Check if UART16550 has received data

Returns

RX ready flag

uart16550_rxwait()

```
void uart16550_rxwait ()
```

Wait for RX Data.

Active wait for RX incoming data.

Returns

void.

uart16550_sendfile()

```
void uart16550_sendfile (
              char * file_name,
              int file_size,
              char * mem)
```

Send file.

Send variable size file via UART16550. Order of commands:

1. Send file transmit (FTX) command.
2. Send file_name.
3. Send file_size (in little endian format).
4. Send file.

Parameters

file_name	Pointer to file name string.
file_size	Size of file to be sent.
mem	Pointer to file.

Returns

void.

uart16550_txready()

```
char uart16550_txready ()
```

Check if TX is ready.

Check if UART16550 has data to send

Returns

TX ready flag

uart16550_txwait()

```
void uart16550_txwait ()
```

Wait for TX.

Active wait until TX is ready to process new byte to send.

Returns

void.