IOb-UART16550

A NS16550A-compatible UART IP Core

January 28, 2026

User Guide, V0.1, Build e8dd704

©2026 IObundle Lda. All rights reserved

Document Version History

Version	Date	Person	Changes from previous version
0.1	January 28, 2026	AN	Initial document version

©2026 IObundle Lda. All rights reserved

©2026 IObundle Lda. All rights reserved

Contents

©2026 IObundle Lda. All rights reserved

List of Tables

©2026 IObundle Lda. All rights reserved

List of Figures

©2026 IObundle Lda. All rights reserved

©2026 IObundle Lda. All rights reserved

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.

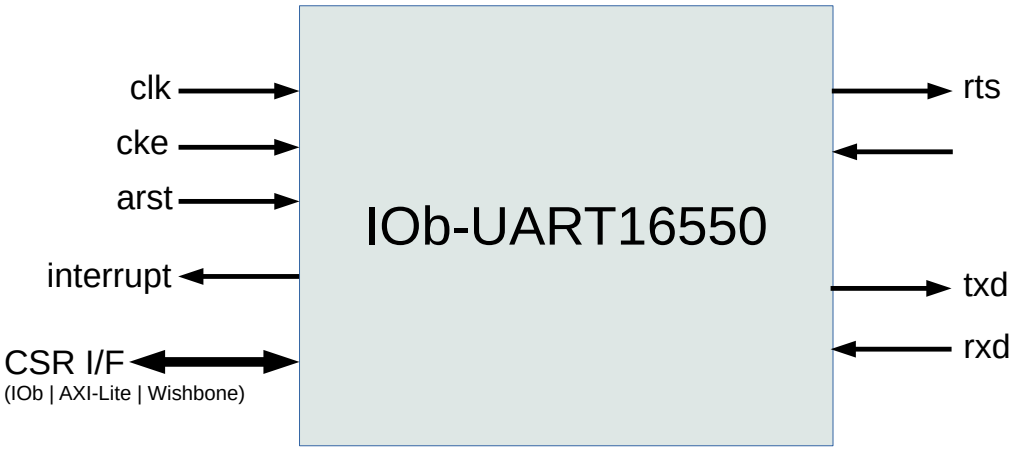

Figure 1: IP Core Symbol

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

©2026 IObundle Lda. All rights reserved

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.

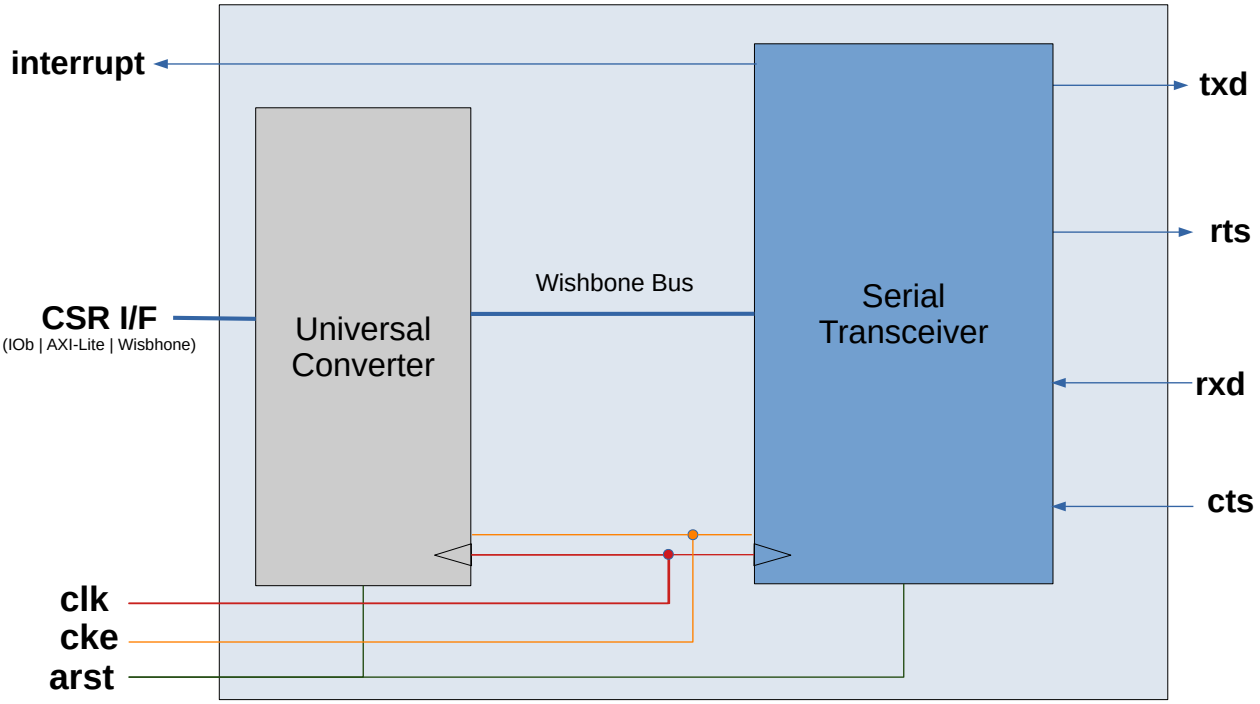

Figure 2: High-Level Block Diagram

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: Core subblocks.

©2026 IObundle Lda. All rights reserved

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
VERSION	M	NA	16'h0001	NA	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.

Table 2: General operation group

There may be other macros or generic parameters in the code that are not documented. They are typically derived from other primary macros or parameters or exist for documentation purposes.

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	4	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.

©2026 IObundle Lda. All rights reserved

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	4	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	4	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

©2026 IObundle Lda. All rights reserved

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.

©2026 IObundle Lda. All rights reserved

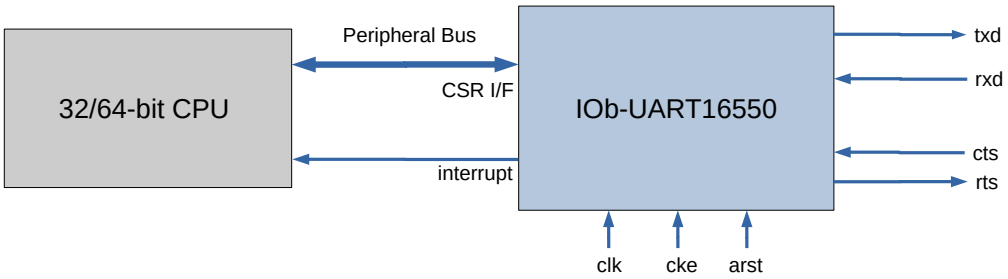

Figure 3: Core Instance and Required Surrounding Blocks

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.

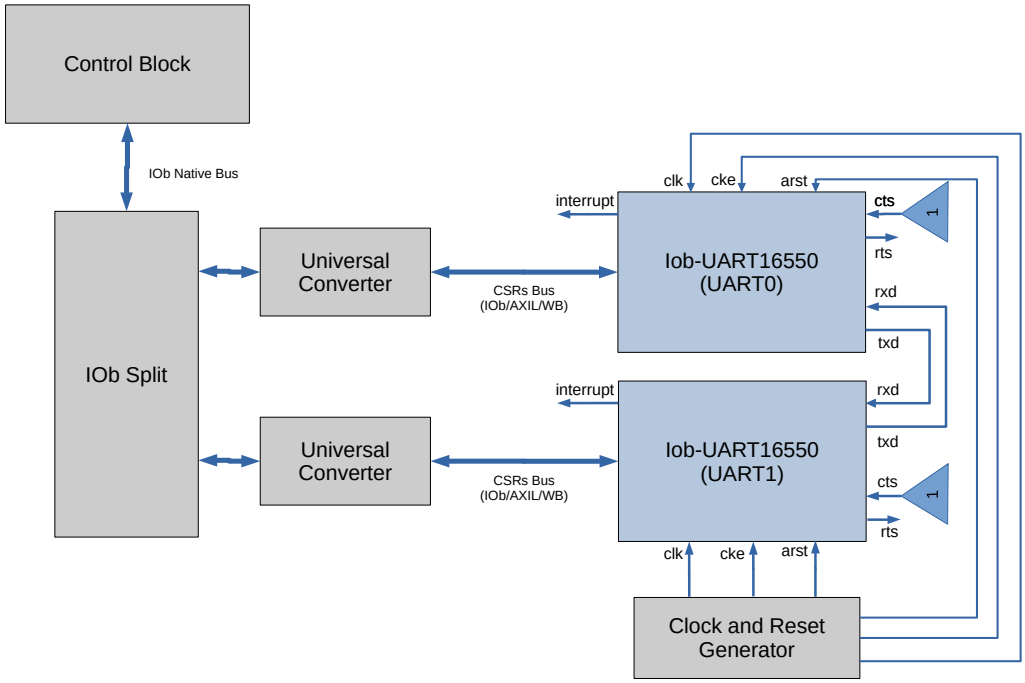

Figure 4: Testbench Block Diagram

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.

©2026 IObundle Lda. All rights reserved

- 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.

©2026 IObundle Lda. All rights reserved

- #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

 ©2026 IObundle Lda. All rights reserved

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.

©2026 IObundle Lda. All rights reserved

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.

©2026 IObundle Lda. All rights reserved

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

©2026 IObundle Lda. All rights reserved

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 ()
```


©2026 IObundle Lda. All rights reserved

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.

4.2 iob_uart16550_csrs.h File Reference

UART16550 Control and Status Register access functions.

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

Macros

- #define **IOB_UART16550_CSRS_CSRS_ADDR_W** 5
  Address width required to access all CSRs in UART16550.
- #define **IOB_UART16550_CSRS_RB_ADDR** (0)
  Receiver buffer.
- #define **IOB_UART16550_CSRS_TR_ADDR** (0)
  Transmitter.
- #define **IOB_UART16550_CSRS_IE_ADDR** (1)
  Interrupt enable.
- #define **IOB_UART16550_CSRS_II_ADDR** (2)
  Interrupt identification.
- #define **IOB_UART16550_CSRS_FC_ADDR** (2)
  FIFO control.
- #define **IOB_UART16550_CSRS_LC_ADDR** (3)
  Line control.
- #define **IOB_UART16550_CSRS_MC_ADDR** (4)
  Modem control.
- #define **IOB_UART16550_CSRS_LS_ADDR** (5)

©2026 IObundle Lda. All rights reserved

Scratch register.

- #define **IOB_UART16550_CSRS_MS_ADDR** (6)
  Modem status.
- #define **IOB_UART16550_CSRS_SR_ADDR** (7)
  Scratch register.
- #define **IOB_UART16550_CSRS_DL1_ADDR** (0)
  Divisor latch bytes (1).
- #define **IOB_UART16550_CSRS_DL2_ADDR** (1)
  Divisor latch bytes (2).
- #define **IOB_UART16550_CSRS_DB1_ADDR** (8)
  Debug register 1.
- #define **IOB_UART16550_CSRS_DB2_ADDR** (12)
  Debug register 2.
- #define **IOB_UART16550_IE_RDA** (0)
  Received Data Available.
- #define **IOB_UART16550_IE_THRE** (1)
  Transmitter Holding Register Empty.
- #define **IOB_UART16550_IE_RLS** (2)
  Receiver Line Status.
- #define **IOB_UART16550_IE_MS** (3)
  Modem Status.
- #define **IOB_UART16550_II_PND** (0)
  Pending Interrupt.
- #define **IOB_UART16550_II_RLS** (0b011)
  Receiver Line Status.
- #define **IOB_UART16550_II_RDA** (0b010)
  Receiver Data Available.
- #define **IOB_UART16550_II_TI** (0b110)
  Timeout Indication.
- #define **IOB_UART16550_II_THRE** (0b001)
  Transmitter Holding Register empty.
- #define **IOB_UART16550_II_MS** (0b000)
  Modem Status.
- #define **IOB_UART16550_FC_RF** (1)
  Clear Receive FIFO.
- #define **IOB_UART16550_FC_TF** (2)
  Clear Transmitter FIFO.
- #define **IOB_UART16550_FC_TL** (6)
  Receiver FIFO Trigger Level address.
- #define **IOB_UART16550_FC_TL_1** (0b00)
  Receiver FIFO Trigger Level: 1 byte.
- #define **IOB_UART16550_FC_TL_4** (0b01)
  Receiver FIFO Trigger Level: 4 byte.
- #define **IOB_UART16550_FC_TL_8** (0b10)
  Receiver FIFO Trigger Level: 8 byte.
- #define **IOB_UART16550_FC_TL_14** (0b11)
  Receiver FIFO Trigger Level: 14 byte.
- #define **IOB_UART16550_LC_BITS** (0)

©2026 IObundle Lda. All rights reserved

Bits per character (0:1).

- #define **IOB_UART16550_LC_SB** (2)
  Stop bits.
- #define **IOB_UART16550_LC_PE** (3)
  Parity enable.
- #define **IOB_UART16550_LC_EP** (4)
  Even parity.
- #define **IOB_UART16550_LC_SP** (5)
  Stick parity.
- #define **IOB_UART16550_LC_BC** (6)
  Break control.
- #define **IOB_UART16550_LC_DL** (7)
  Divisor latch access.
- #define **IOB_UART16550_MC_DTR** (0)
  Data Terminal Ready.
- #define **IOB_UART16550_MC_RTS** (1)
  Request To Send.
- #define **IOB_UART16550_MC_OUT1** (2)
  Loopback -> Ring Indicator.
- #define **IOB_UART16550_MC_OUT2** (3)
  Loopback -> Data Carrier Detect.
- #define **IOB_UART16550_MC_LB** (4)
  LoopBack mode.
- #define **IOB_UART16550_LS_DR** (0)
  Data Ready.
- #define **IOB_UART16550_LS_OE** (1)
  Overrun Error.
- #define **IOB_UART16550_LS_PE** (2)
  Parity Error.
- #define **IOB_UART16550_LS_FE** (3)
  Framing Error.
- #define **IOB_UART16550_LS_BI** (4)
  Break Interrupt.
- #define **IOB_UART16550_LS_TFE** (5)
  Transmit FIFO empty.
- #define **IOB_UART16550_LS_TE** (6)
  Transmitter Empty Indicator.
- #define **IOB_UART16550_LS_EI** (7)
  Error indicator.
- #define **IOB_UART16550_MS_DCTS** (0)
  Delta Clear To Send.
- #define **IOB_UART16550_MS_DDSR** (1)
  Delta Data Set Ready.
- #define **IOB_UART16550_MS_TERI** (2)
  Trailing Edge of Ring Indicator.
- #define **IOB_UART16550_MS_DDCD** (3)
  Delta Data Carrier Detect.
- #define **IOB_UART16550_MS_CCTS** (4)

©2026 IObundle Lda. All rights reserved

Complement Clear To Send.

- #define **IOB_UART16550_MS_CDSR** (5)
  Complement Data Set Ready.
- #define **IOB_UART16550_MS_CRI** (6)
  Complement Ring Indicator.
- #define **IOB_UART16550_MS_CDCD** (7)
  Complement Data Carrier Detect.
- #define **IOB_UART16550_CSRS_W** 8
  CSR Data Width.

Functions

- void iob_uart16550_csrs_init_baseaddr (uint32_t addr)
  Set core base address.
- void iob_write (uint32_t addr, uint32_t data_w, uint32_t value)
  Write access function prototype.
- uint32_t iob_read (uint32_t addr, uint32_t data_w)
  Read access function prototype.
- uint8_t iob_uart16550_csrs_get_rb ()
  Get receiver buffer. Receiver buffer.
- void iob_uart16550_csrs_set_tr (uint8_t value)
  Write to transmitter buffer. Transmitter buffer.
- uint8_t iob_uart16550_csrs_get_ie ()
  Get interrupt enable. Interrupt enable.
- void iob_uart16550_csrs_set_ie (uint8_t value)
  Set interrupt enable. Interrupt enable.
- uint8_t iob_uart16550_csrs_get_ii ()
  Get interrupt identification. Interrupt indentification.
- void iob_uart16550_csrs_set_fc (uint8_t value)
  Set FIFO control. FIFO control.
- uint8_t iob_uart16550_csrs_get_lc ()
  Get Line control. Line control.
- void iob_uart16550_csrs_set_lc (uint8_t value)
  Set Line control. Line control.
- void iob_uart16550_csrs_set_mc (uint8_t value)
  Set Modem control. Modem control.
- uint8_t iob_uart16550_csrs_get_ls ()
  Get Scratch register. Scratch register.
- uint8_t iob_uart16550_csrs_get_ms ()
  Get Modem status. Modem status.
- uint8_t iob_uart16550_csrs_get_dl1 ()
  Get Divisor latch bytes (1). Divisor latch bytes.
- void iob_uart16550_csrs_set_dl1 (uint8_t value)
  Set Divisor latch bytes (1). Divisor latch bytes (1)
- uint8_t iob_uart16550_csrs_get_dl2 ()
  Get Divisor latch bytes (2). Divisor latch bytes.
- void iob_uart16550_csrs_set_dl2 (uint8_t value)

 ©2026 IObundle Lda. All rights reserved

Set Divisor latch bytes (2). Divisor latch bytes (2)

- uint8_t iob_uart16550_csrs_get_db1 ()

  Get Debug register 1. Debug register.
- uint8_t iob_uart16550_csrs_get_db2 ()

  Get Debug register 2. Debug register.

4.2.1 Detailed Description

UART16550 Control and Status Register access functions.

The present IOb-UART16550 software drivers map the Control and Status Registers for direct core access.

4.2.2 Function Documentation

iob_read()

```
uint32_t iob_read (
               uint32_t addr,
               uint32_t data_w)
```

Read access function prototype.

Parameters

addr	Address to write to.
data_w	Data width in bits.

Returns

uint32_t Read data value.

iob_uart16550_csrs_get_db1()

```
uint8_t iob_uart16550_csrs_get_db1 ()
```

Get Debug register 1. Debug register.

Returns

uint8_t Debug register 1.

iob_uart16550_csrs_get_db2()

```
uint8_t iob_uart16550_csrs_get_db2 ()
```

Get Debug register 2. Debug register.

Returns

uint8_t Debug register 2.

iob_uart16550_csrs_get_dl1()

```
uint8_t iob_uart16550_csrs_get_dl1 ()
```

Get Divisor latch bytes (1). Divisor latch bytes.

Returns

uint8_t Divisor latch bytes (1).

iob_uart16550_csrs_get_dl2()

```
uint8_t iob_uart16550_csrs_get_dl2 ()
```

Get Divisor latch bytes (2). Divisor latch bytes.

Returns

uint8_t Divisor latch bytes (2).

iob_uart16550_csrs_get_ie()

```
uint8_t iob_uart16550_csrs_get_ie ()
```

Get interrupt enable. Interrupt enable.

Returns

uint8_t interrupt enable value.

©2026 IObundle Lda. All rights reserved

iob_uart16550_csrs_get_ii()

```
uint8_t iob_uart16550_csrs_get_ii ()
```

Get interrupt identification. Interrupt indentification.

Returns

uint8_t interrupt identification.

iob_uart16550_csrs_get_lc()

```
uint8_t iob_uart16550_csrs_get_lc ()
```

Get Line control. Line control.

Returns

uint8_t current Line control.

iob_uart16550_csrs_get_ls()

```
uint8_t iob_uart16550_csrs_get_ls ()
```

Get Scratch register. Scratch register.

Returns

uint8_t Scratch register.

iob_uart16550_csrs_get_ms()

```
uint8_t iob_uart16550_csrs_get_ms ()
```

Get Modem status. Modem status.

Returns

uint8_t Modem status.

iob_uart16550_csrs_get_rb()

```
uint8_t iob_uart16550_csrs_get_rb ()
```

Get receiver buffer. Receiver buffer.

©2026 IObundle Lda. All rights reserved

Returns

uint8_t receiver buffer value.

iob_uart16550_csrs_init_baseaddr()

```
void iob_uart16550_csrs_init_baseaddr (
              uint32_t addr)
```

Set core base address.

This function sets the base address for the core in the system. All other accesses are offset from this base address.

Parameters

addr	Base address for core.

iob_uart16550_csrs_set_dl1()

```
void iob_uart16550_csrs_set_dl1 (
              uint8_t value)
```

Set Divisor latch bytes (1). Divisor latch bytes (1)

Parameters

value	for Divisor latch bytes (1).

iob_uart16550_csrs_set_dl2()

```
void iob_uart16550_csrs_set_dl2 (
              uint8_t value)
```

Set Divisor latch bytes (2). Divisor latch bytes (2)

Parameters

value	for Divisor latch bytes (2).

©2026 IObundle Lda. All rights reserved

iob_uart16550_csrs_set_fc()

```
void iob_uart16550_csrs_set_fc (
              uint8_t value)
```

Set FIFO control. FIFO control.

Parameters

value	FIFO control.

iob_uart16550_csrs_set_ie()

```
void iob_uart16550_csrs_set_ie (
              uint8_t value)
```

Set interrupt enable. Interrupt enable.

Parameters

value	for interrupt enable.

iob_uart16550_csrs_set_lc()

```
void iob_uart16550_csrs_set_lc (
              uint8_t value)
```

Set Line control. Line control.

Parameters

value	for Line control.

iob_uart16550_csrs_set_mc()

```
void iob_uart16550_csrs_set_mc (
              uint8_t value)
```

Set Modem control. Modem control.

Parameters

value	for Modem control.

©2026 IObundle Lda. All rights reserved

iob_uart16550_csrs_set_tr()

```
void iob_uart16550_csrs_set_tr (
              uint8_t value)
```

Write to transmitter buffer. Transmitter buffer.

Parameters

value	to write to tx buffer.

iob_write()

```
void iob_write (
              uint32_t addr,
              uint32_t data_w,
              uint32_t value)
```

Write access function prototype.

Parameters

addr	Address to write to.
data_w	Data width in bits.
value	Value to write.

©2026 IObundle Lda. All rights reserved