

# wishbone\_classic\_uart.v

---

## AUTHORS

---

JAY CONVERTINO

---

## DATES

---

2024/02/29

---

## INFORMATION

---

### Brief

---

AXI Lite 1553 is a core for interfacing with 1553 devices over the AXI lite bus.

### License MIT

---

Copyright 2024 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## wishbone\_classic\_uart

---

```
module wishbone_classic_uart #(
    parameter
    ADDRESS_WIDTH
    =
    32,
    parameter
    BUS_WIDTH
    =
    4,
    parameter
    CLOCK_SPEED
```

```

    =
    100000000,
parameter
BAUD_RATE
    =
    115200,
parameter
PARITY_ENA
    =
    0,
parameter
PARITY_TYPE
    =
    0,
parameter
STOP_BITS
    =
    1,
parameter
DATA_BITS
    =
    8,
parameter
RX_DELAY
    =
    0,
parameter
RX_BAUD_DELAY
    =
    0,
parameter
TX_DELAY
    =
    0,
parameter
TX_BAUD_DELAY
    =
    0
) ( input clk, input rst, input s_wb_cyc, input s_wb_stb, input s_wb_we, ing

```

AXI Lite based uart device.

## Parameters

<b>ADDRESS_WIDTH</b> parameter	Width of the address bus in bits.
<b>BUS_WIDTH</b> parameter	Width of the data bus in bytes.
<b>CLOCK_SPEED</b> parameter	This is the aclk frequency in Hz
<b>BAUD_RATE</b> parameter	Serial Baud, this can be any value including non-standard.
<b>PARITY_ENA</b> parameter	Enable Parity for the data in and out.
<b>PARITY_TYPE</b> parameter	Set the parity type, 0 = even, 1 = odd, 2 = mark, 3 = space.
<b>STOP_BITS</b> parameter	Number of stop bits, 0 to crazy non-standard amounts.
<b>DATA_BITS</b> parameter	Number of data bits, 1 to crazy non-standard amounts.

<b>RX_DELAY</b> parameter	Delay in rx data input.
<b>RX_BAUD_DELAY</b> parameter	Delay in rx baud enable. This will delay when we sample a bit (default is midpoint when rx delay is 0).
<b>TX_DELAY</b> parameter	Delay in tx data output. Delays the time to output of the data.
<b>TX_BAUD_DELAY</b> parameter	Delay in tx baud enable. This will delay the time the bit output starts.

## Ports

<b>clk</b>	Clock for all devices in the core
<b>rst</b>	Positive reset
<b>s_wb_cyc</b>	Bus Cycle in process
<b>s_wb_stb</b>	Valid data transfer cycle
<b>s_wb_we</b>	Active High write, low read
<b>s_wb_addr</b>	Bus address
<b>s_wb_data_i</b>	Input data
<b>s_wb_sel</b>	Device Select
<b>s_wb_bte</b>	Burst Type Extension
<b>s_wb_cti</b>	Cycle Type
<b>s_wb_ack</b>	Bus transaction terminated
<b>s_wb_data_o</b>	Output data
<b>s_wb_err</b>	Active high when a bus error is present
<b>irq</b>	Interrupt when data is received
<b>tx</b>	transmit for UART (output to RX)
<b>rx</b>	receive for UART (input from TX)
<b>rts</b>	request to send is a loop with CTS
<b>cts</b>	clear to send is a loop with RTS

## up\_rreq

---

```
wire up_rreq
```

uP read bus request

## up\_rack

---

```
wire up_rack
```

uP read bus acknowledge

## up\_raddr

---

```
wire [ADDRESS_WIDTH-3:0] up_raddr
```

uP read bus address

## up\_rdata

---

```
wire [31:0] up_rdata
```

uP read bus request

## up\_wreq

---

```
wire up_wreq
```

uP write bus request

## up\_wack

---

```
wire up_wack
```

uP write bus acknowledge

## up\_waddr

---

```
wire [ADDRESS_WIDTH-3:0] up_waddr
```

uP write bus address

## up\_wdata

---

```
wire [31:0] up_wdata
```

uP write bus data

## INSTANTIATED MODULES

---

### inst\_up\_wishbone\_classic

---

```
up_wishbone_classic #(
    ADDRESS_WIDTH(ADDRESS_WIDTH),
    BUS_WIDTH(BUS_WIDTH)
) inst_up_wishbone_classic ( .clk(clk), .rst(rst), .s_wb_cyc(s_wb_cyc), .s_v
```

Module instance of up\_wishbone\_classic for the Wishbone Classic bus to the uP bus.

## inst\_up\_uart

---

```
up_uart #(
    ADDRESS_WIDTH(ADDRESS_WIDTH),
    BUS_WIDTH(BUS_WIDTH),
    CLOCK_SPEED(CLOCK_SPEED),
    BAUD_RATE(BAUD_RATE),
    PARITY_ENA(PARITY_ENA),
    PARITY_TYPE(PARITY_TYPE),
    STOP_BITS(STOP_BITS),
    DATA_BITS(DATA_BITS),
    RX_DELAY(RX_DELAY),
    RX_BAUD_DELAY(RX_BAUD_DELAY),
    TX_DELAY(TX_DELAY),
    TX_BAUD_DELAY(TX_BAUD_DELAY)
) inst_up_uart ( .clk(clk), .rstn(~rst), .up_rreq(up_rreq), .up_rack(up_rack)
```

Module instance of up\_uart creating a Logic wrapper for uart axis bus cores to interface with uP bus.