

# up\_wishbone\_classic.v

---

## AUTHORS

---

JAY CONVERTINO

---

## DATES

---

2024/03/01

---

## INFORMATION

---

### Brief

---

Wishbone Classic slave to uP interface

### 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.

## up\_wishbone\_classic

---

```
module up_wishbone_classic #(
    parameter
    ADDRESS_WIDTH
    =
    16,
    parameter
    BUS_WIDTH
    =
    4
) ( input clk, input rst, input s_wb_cyc, input s_wb_stb, input s_wb_we, inp
```

Wishbone Classic slave to uP up\_wishbone\_classic

## Parameters

**ADDRESS\_WIDTH** Width of the APB3 address port in bits.

parameter

**BUS\_WIDTH** Width of the APB3 bus data port in bytes.

parameter

## Ports

<b>clk</b>	Clock
<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>up_rreq</b>	uP bus read request
<b>up_rack</b>	uP bus read ack
<b>up_raddr</b>	uP bus read address
<b>up_rdata</b>	uP bus read data
<b>up_wreq</b>	uP bus write request
<b>up_wack</b>	uP bus write ack
<b>up_waddr</b>	uP bus write address
<b>up_wdata</b>	uP bus write data

## VARIABLES

---

### s\_next\_address

---

```
assign s_next_address = wb_next_adr(
    r_address,
    r_wb_cti &
    s_wb_cti,
    s_wb_bte,
    BUS_WIDTH *
    8
)
```

Use the fusesoc wb\_next\_adr function to generate a address when wishbone classic is in a burst mode.

## valid

---

```
assign valid = s_wb_cyc & s_wb_stb & ~r_rst[0]
```

Indicate valid request from wishbone.

## up\_rreq

---

```
assign up_rreq = ~s_wb_we & r_req
```

Convert wishbone read requests to up requests

## up\_wreq

---

```
assign up_wreq = s_wb_we & r_req
```

Convert wishbone write requests to up requests

## s\_wb\_err

---

```
assign s_wb_err = 1'b0
```

TODO:check for burst address errors

## up\_raddr

---

```
assign up_raddr = (
    address_state == init_address ? s_wb_addr : s_next_address)
    :
    0
    )
```

assign address to read address port if selected

## up\_waddr

---

```
assign up_waddr = (
    address_state == init_address ? s_wb_addr : r_address)
    :
    0
    )
```

assign address to write address port if selected

## up\_ack

---

```
assign up_ack = (  
  up_rack |  
  up_wack  
)
```

ack is ack for both, or them so either may pass

## **s\_wb\_ack**

---

```
assign s_wb_ack = up_ack
```

combined uP ack is wishbone ack.