up_wishbone_classic.v

AUTHORS

JAY CONVERTINO

DATES

2024/03/01

INFORMATION

Brief

Wishbone Classic slave to uP interface

License MIT

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 parameter	Width of the Wishbone address port in bits.
BUS_WIDTH parameter	Width of the Wishbone bus data port in bytes.

Ports

clk	Clock
rst	Positive reset
s_wb_cyc	Bus Cycle in process
s_wb_stb	Valid data transfer cycle
s_wb_we	Active High write, low read
s_wb_addr	Bus address
s_wb_data_i	Input data
s_wb_sel	Device Select
s_wb_ack	Bus transaction terminated
s_wb_data_o	Output data
s_wb_err	Active high when a bus error is present
up_rreq	uP bus read request
up_rack	uP bus read ack
up_raddr	uP bus read address
up_rdata	uP bus read data
up_wreq	uP bus write request
up_wack	uP bus write ack
up_waddr	uP bus write address
up_wdata	uP bus write data

VARIABLES

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 = r_err
```

check for errors

up_raddr

```
assign up_raddr = (
                                                    ~s_wb_we & ~r_rst[0] ?

s_wb_addr[ADDRESS_WIDTH-1:shift]
:
0
)
```

assign address to read address port if selected

up_waddr

```
assign up_waddr = (
                                                     s_wb_we & ~r_rst[0] ?

s_wb_addr[ADDRESS_WIDTH-1:shift]
:
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.