

wishbone_standard_gpio.v

AUTHORS

JAY CONVERTINO

DATES

2024/07/25

INFORMATION

Brief

Wishbone standard UART core.

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_standard_gpio

```
module wishbone_standard_gpio #(
  parameter
    ADDRESS_WIDTH
    =
    32,
  parameter
    BUS_WIDTH
    =
    4,
  parameter
    GPIO_WIDTH
    =
    32,
  parameter
```

```

    IRQ_ENABLE
    =
    0
) ( input clk, input rst, input s_wb_cyc, input s_wb_stb, input s_wb_we, input

```

Wishbone Classic Standard based uart device.

Parameters

ADDRESS_WIDTH Width of the address bus in bits, max 32 bit.
parameter

BUS_WIDTH parameter	Width of the data bus in bytes.
-------------------------------	---------------------------------

GPIO_WIDTH parameter	Width of the GPIO for inputs and outputs
--------------------------------	--

IRQ_ENABLE Enable interrupt
parameter

Ports

clk	Clock for all devices in the core
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
irq	Interrupt when data is received
gpio_io_i	Input for GPIO
gpio_io_o	Output for GPIO
gpio_io_t	Tristate for GPIO

up_rreq

```
wire up_rreq
```

uP read bus request

up_rack

```
wire up_rack
```

uP read bus acknowledge

up_raddr

```
wire [ADDRESS_WIDTH-(  
BUS_WIDTH  
  
2  
)-1:0] 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-(  
BUS_WIDTH  
  
2  
)-1:0] up_waddr
```

uP write bus address

up_wdata

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

uP write bus data

INSTANTIATED MODULES

inst_up_wishbone_standard

```
up_wishbone_standard #(  
  
  
.
```

```

ADDRESS_WIDTH(ADDRESS_WIDTH),
BUS_WIDTH(BUS_WIDTH)
) inst_up_wishbone_standard ( .clk(clk), .rst(rst), .s_wb_cyc(s_wb_cyc), .s_

```

Module instance of up_wishbone_standard for the Wishbone Classic Standard bus to the uP bus.

inst_up_gpio

```

up_gpio #(
ADDRESS_WIDTH(ADDRESS_WIDTH),
BUS_WIDTH(BUS_WIDTH),
GPIO_WIDTH(GPIO_WIDTH),
IRQ_ENABLE(IRQ_ENABLE)
) inst_up_gpio ( .clk(aclk), .rstn(arstn), .up_rreq(up_rreq), .up_rack(up_rack)

```

Module instance of up_gpio.