

axi_lite_gpio.v

AUTHORS

JAY CONVERTINO

DATES

2024/07/25

INFORMATION

Brief

AXI Lite GPIO is a core for creating a generic programmable input/output

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.

axi_lite_gpio

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

```

    IRQ_ENABLE
    =
    0
) ( input aclk, input arstn, input s_axi_aclk, input s_axi_aresetn, input s_

```

AXI Lite based gpio device.

Parameters

ADDRESS_WIDTH <small>parameter</small>	Width of the axi address bus, max 32 bit.
GPIO_WIDTH <small>parameter</small>	Width of the GPIO for inputs and outputs
IRQ_ENABLE <small>parameter</small>	Enable interrupt

Ports

aclk	Clock for all devices in the core
arstn	Negative reset
s_axi_awvalid	Axi Lite aw valid
s_axi_awaddr	Axi Lite aw addr
s_axi_awprot	Axi Lite aw prot
s_axi_awready	Axi Lite aw ready
s_axi_wvalid	Axi Lite w valid
s_axi_wdata	Axi Lite w data
s_axi_wstrb	Axi Lite w strb
s_axi_wready	Axi Lite w ready
s_axi_bvalid	Axi Lite b valid
s_axi_bresp	Axi Lite b resp
s_axi_bready	Axi Lite b ready
s_axi_arvalid	Axi Lite ar valid
s_axi_araddr	Axi Lite ar addr
s_axi_arprot	Axi Lite ar prot
s_axi_arready	Axi Lite ar ready
s_axi_rvalid	Axi Lite r valid
s_axi_rdata	Axi Lite r data
s_axi_rresp	Axi Lite r resp
s_axi_rready	Axi Lite r ready
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_axi

```
up_axi #(
    ADDRESS_WIDTH(ADDRESS_WIDTH),
    ) inst_up_axi ( .up_rstn (arstn), .up_clk (aclk), .up_axi_awvalid(s_axi_awv
```

Module instance of up_axi for the AXI Lite 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.