

# 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
    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

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

### Ports

<b>aclk</b>	Clock for all devices in the core
<b>arstn</b>	Negative reset
<b>s_axi_awvalid</b>	Axi Lite aw valid
<b>s_axi_awaddr</b>	Axi Lite aw addr
<b>s_axi_awprot</b>	Axi Lite aw prot
<b>s_axi_awready</b>	Axi Lite aw ready
<b>s_axi_wvalid</b>	Axi Lite w valid
<b>s_axi_wdata</b>	Axi Lite w data
<b>s_axi_wstrb</b>	Axi Lite w strb
<b>s_axi_wready</b>	Axi Lite w ready
<b>s_axi_bvalid</b>	Axi Lite b valid
<b>s_axi_bresp</b>	Axi Lite b resp
<b>s_axi_bready</b>	Axi Lite b ready
<b>s_axi_arvalid</b>	Axi Lite ar valid
<b>s_axi_araddr</b>	Axi Lite ar addr
<b>s_axi_arprot</b>	Axi Lite ar prot
<b>s_axi_arready</b>	Axi Lite ar ready
<b>s_axi_rvalid</b>	Axi Lite r valid
<b>s_axi_rdata</b>	Axi Lite r data
<b>s_axi_rresp</b>	Axi Lite r resp
<b>s_axi_rready</b>	Axi Lite r ready
<b>irq</b>	Interrupt when data is received
<b>gpio_io_i</b>	Input for GPIO
<b>gpio_io_o</b>	Output for GPIO
<b>gpio_io_t</b>	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(4),
    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.