BUS_GPIO

April 1, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

BUS GPIO is a core for interfacing over generic input/output to a bus of choice. The data can then be accessed over a BUS, currently AXI lite or Wishbone Standard, and processed as needed. All input and output over the bus goes out directly to the IO. The following is information on how to use the device in an FPGA, software, and in simulation.

1.2 Dependencies

The following are the dependencies of the cores.

- fusesoc 2.X
- iverilog (simulation)
- cocotb (simulation)

1.2.1 axi_lite_gpio Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up_gpio:1.0.0
  - AD:common:up_axi:1.0.0
- dep_tb
  - AFRL:simulation:axis_stimulator
  - AFRL:utility:sim_helper

1.2.2 wishbone_standard_gpio Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up_gpio:1.0.0
  - AFRL:bus:up_wishbone_standard:1.0.0

1.2.3 up_gpio Depenecies

- dep
  - AFRL:utility:helper:1.0.0

1.3 In a Project

First, pick a core that matches the target bus in question. Then connect the BUS GPIO core to that bus. Once this is complete the GPIO pins will need to be routed.

2 Architecture

This core is made up of other cores that are documented in detail in there source. The cores this is made up of are the,

- **up_axi** An AXI Lite to uP converter core (see core for documentation).
- **up_wishbone_standard** A wishbone standard to uP converter core (see core for documentation).
- **up_gpio** Takes uP bus and coverts it to interface with general purpose input/output (see module documentation for information 5).

For register documentation please see up_gpio in 5

3 Building

The BUS GPIO is written in Verilog 2001. It should synthesize in any modern FPGA software. The core comes as a fusesoc packaged core and can be included in any other core. Be sure to make sure you have meet the dependencies listed in the previous section.

3.1 fusesoc

Fusesoc is a system for building FPGA software without relying on the internal project management of the tool. Avoiding vendor lock in to Vivado or Quartus. These cores, when included in a project, can be easily integrated and targets created based upon the end developer needs. The core by itself is not a part of a system and should be integrated into a fusesoc based system. Simulations are setup to use fusesoc and are a part of its targets.

3.2 Source Files

3.2.1 axi_lite_gpio File List

- src

- src/axi_lite_gpio.v

- tb_cocotb
  - 'tb/tb_cocotb_axi_lite.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb_axi_lite.v': 'file_type': 'verilogSource'

3.2.2 wishbone_standard_gpio File List

- src
  - src/wishbone_standard_gpio.v
- tb_cocotb
  - 'tb/tb_cocotb_wishbone_standard.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb_wishbone_standard.v': 'file_type': 'verilogSource'
- tb
  - tb/tb_wishbone_slave.v

3.2.3 up_gpio File List

- src
  - src/up_gpio.v
- tb_cocotb
  - 'tb/tb_cocotb_up.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb_up.v': 'file_type': 'verilogSource'
- tb
  - tb/tb_up_gpio.v

3.3 Targets

3.3.1 axi_lite_gpio Targets

- default

  Info: Default for IP intergration.

- sim_cocotb

  Info: Cocotb unit tests

3.3.2 wishbone_standard_gpio Targets

- default

  Info: Default for IP intergration.

- sim

  Info: Base simulation using icarus as default.

- sim_cocotb

  Info: Cocotb unit tests

3.3.3 up_gpio Targets

- default

  Info: Default for IP intergration.

- sim

  Info: Base simulation using icarus as default.

- sim_cocotb

  Info: Cocotb unit tests

3.4 Directory Guide

Below highlights important folders from the root of the directory.

1. **docs** Contains all documentation related to this project.
   - **manual** Contains user manual and github page that are generated from the latex sources.
2. **src** Contains source files for the core
3. **tb** Contains test bench files for iverilog and cocotb
   - **cocotb** testbench files

4 Simulation

There are a few different simulations that can be run for this core.

4.1 iverilog

iverilog is used for simple test benches for quick verification, visually, of the core.

4.2 cocotb

Future simulations will use cocotb. This feature is not yet implemented.

5 Module Documentation

up_gpio is the module that provides the general purpose input/output. The uP bus is the microprocessor bus based on Analog Devices design. It resembles a APB bus in design, and is the bridge to other buses BUS UART can use. This makes changing for AXI Lite, to Wishbone to whatever quick and painless.

axi_lite_gpio module adds a AXI Lite to uP (microprocessor) bus converter. The converter is from Analog Devices.

wishbone_standard_gpio module adds a Wishbone Standard to uP (microprocessor) bus converter. This converter was designed for Wishbone Standard only, NOT pipelined.

The next sections document these modules in great detail. up_gpio contains the register map explained, and what the various bits do.

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

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

ADDRESS_WIDTH parameter	Width of the axi address bus, max 32 bit.
GPIO_WIDTH parameter	Width of the GPIO for inputs and outputs
IRQ_ENABLE parameter	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

INSTANTIANTED MODULES

inst_up_axi

```
up_axi #(
                                                                                .
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_ra
```

Module instance of up_gpio.

wishbone_standard_gpio.v

AUTHORS

JAY CONVERTINO

DATES

2024/07/25

INFORMATION

Brief

Wishbone standard UART core.

License MIT

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, inp
```

Wishbone Classic Standard based uart device.

Parameters

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

BUS_WIDTH Width of the data bus in bytes.
parameter

GPIO_WIDTH Width of the GPIO for inputs and outputs
parameter

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

INSTANTIANTED 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_ra
```

Module instance of up_gpio.

up_gpio.v

AUTHORS

JAY CONVERTINO

DATES

2024/07/25

INFORMATION

Brief

uP Core for interfacing with general purpose input/output.

License MIT

up_gpio

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

```
 IRQ_ENABLE
  =
 0
) ( input clk, input rstn, input up_rreq, output up_rack, input [ADDRESS_WI
```

uP based GPIO device.

Parameters

ADDRESS_WIDTH	Width of the uP address port, max 32 bit.
parameter		
BUS_WIDTH	Width of the uP bus data port.
parameter		
GPIO_WIDTH	Width of the GPIO for inputs and outputs
parameter		
IRQ_ENABLE	Enable interrupt
parameter		

Ports

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

DIVISOR

```
localparam DIVISOR = BUS_WIDTH/2
```

Divide the address register default location for 1 byte access to multi byte access. (register offsets are byte offsets).

REGISTER INFORMATION

Core has 4 registers at the offsets that follow.

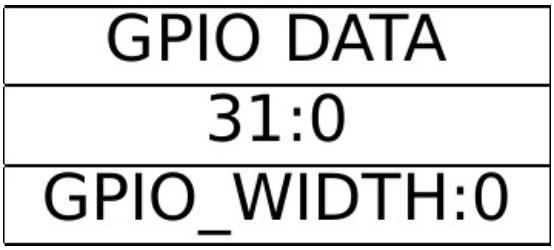
GPIO_DATA	h000
GPIO_TRI	h004
GPIO2_DATA	h008 N/A
GPIO2_TRI	h00C N/A
GIER	h11C

IP_ISR	h120
IP_IER	h128

GPIO_DATA

```
localparam GPIO_DATA = 12'h000 >> DIVISOR
```

Defines the address offset for GPIO DATA



Valid bits are from GPIO_WIDTH:0, input or output data.

GPIO_TRI

```
localparam GPIO_TRI = 12'h004 >> DIVISOR
```

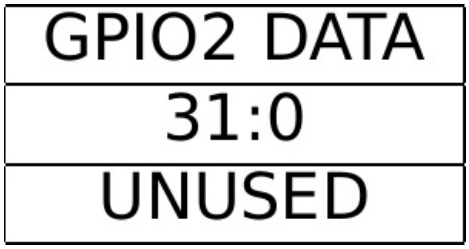
Defines the address offset for GPIO TRI.

Valid bits are from GPIO_WIDTH:0, 1 indicates input, 0 is output.

GPIO2_DATA

```
localparam GPIO2_DATA = 12'h008 >> DIVISOR
```

Defines the address offset for GPIO2 DATA

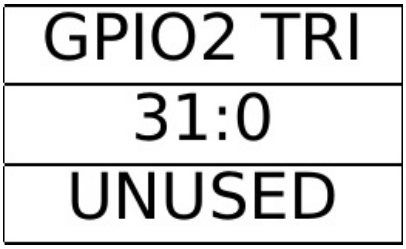


Valid bits are from GPIO2_WIDTH:0, input or output data. This Register is not implimented in this design.

GPIO2_TRI

```
localparam GPIO2_TRI = 12'h00C >> DIVISOR
```

Defines the address offset for GPIO2 TRI.

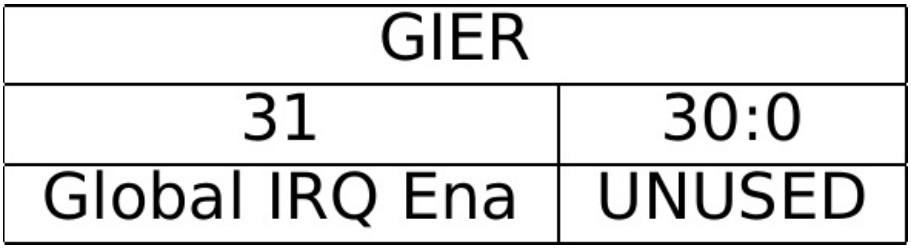


Valid bits are from GPIO2_WIDTH:0, 1 indicates input, 0 is output. This register is not implimented in this design.

GIER

```
localparam GIER = 12'h11C >> DIVISOR
```

Defines the address offset for GIER.



Bit 31 is the Global interrupt enable. Write a 1 to enable interrupts.

IP_ISR

```
localparam IP_ISR = 12'h120 >> DIVISOR
```

Defines the address offset for IP_ISR.

IP ISR	
31:1	0
UNUSED	IRQ Status

Bit 0 is GPIO IRQ status, On write this will toggle(acknowledge) the interrupt.

IP_IER

```
localparam IP_IER = 12'h128 >> DIVISOR
```

Defines the address offset to set the control bits.

IP IER	
31:1	0
UNUSED	IRQ Ena

Bit 0 is GPIO IRQ enable interrupt. Write a 1 to bit 0 to enable interrupt.