BUS_BLOCK_RAM

November 14, 2024

Jay Convertino

Contents

1 Usage

1.1 Introduction

Selectable BUS block RAM for any FPGA target. Currently supports wishbone classic or AXI lite. This will create FPGA block RAM that is accessable via the selected bus.

1.2 Dependencies

The following are the dependencies of the cores.

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

1.2.1 axi_lite_block_ram Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:ram:dc_block_ram:1.0.0
- dep_tb
  - AFRL:simulation:clock_stimulator
  - AFRL:utility:sim_helper

1.2.2 wishbone_classic_block_ram Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:ram:dc_block_ram:1.0.0
  - AFRL:bus:up_wishbone_classic:1.0.0
- dep_tb
  - AFRL:simulation:clock_stimulator
  - AFRL:utility:sim_helper

1.3 In a Project

Connect the device using the bus selected, see 5 for details

2 Architecture

There are two bus block RAM cores. The AXI lite block RAM and the Wishbone Classic block RAM.

AXI lite block RAM is made up of the following modules.

- **up_axi** Convert AXI lite to the Analog Devices uP BUS. (see core for documentation).
- **dc_block_ram** Provides a dual clock block RAM. (see core for documentation).

This core has 1 always blocks that are sensitive to the positive clock edge.

- **register reqest to the acknoledge** Takes the request and registers to the acknoledge. All reads and writes will produce something.

Please see 5 for more information.

Wishbone Classic block RAM is made up of the following modules.

- **up_wishbone_classic** Convert Wishbone Classic to the Analog Devices uP BUS. (see core for documentation).
- **dc_block_ram** Provides a dual clock block RAM. (see core for documentation).

This core has 1 always blocks that are sensitive to the positive clock edge.

- **register reqest to the acknoledge** Takes the request and registers to the acknoledge. All reads and writes will produce something.

Please see 5 for more information.

3 Building

The BUS block RAM cores are written in Verilog 2001. They 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_block_ram File List

- src

  Type: verilogSource

  - src/axi_lite_block_ram.v

- tb

  - 'tb/tb_fifo.v': 'file_type': 'verilogSource'

- constr

  - 'tool_vivado ? (constr/fifo_constr.tcl)': 'file_type': 'SDC'

3.2.2 wishbone_classic_block_ram File List

- src

  Type: verilogSource

  - src/wishbone_classic_block_ram.v

- tb

  - 'tb/tb_wishbone_slave.v': 'file_type': 'verilogSource'

3.3 Targets

3.3.1 axi_lite_block_ram Targets

- default

  Info: Default for IP intergration.

  - src
  - dep

  - constr

- sim

  Info: Constant data value with file check.

  - src
  - dep
  - constr
  - tb
  - dep_tb
  - IN_FILE_NAME
  - OUT_FILE_NAME
  - RAND_FULL
  - FIFO_DEPTH

- sim_rand_data

  Info: Feed random data input with file check

  - src
  - dep
  - constr
  - tb
  - dep_tb
  - IN_FILE_NAME=random.bin
  - OUT_FILE_NAME=out_random.bin
  - RAND_FULL
  - FIFO_DEPTH

- sim_rand_ready_rand_data

  Info: Feed random data input, and randomize the read ready on the output. Perform output file check.

  - src
  - dep
  - constr
  - tb
  - dep_tb
  - IN_FILE_NAME=random.bin
  - OUT_FILE_NAME=out_random.bin
  - RAND_FULL=1

  - FIFO_DEPTH

- sim_8bit_count_data

  Info: Feed a counter data as input, perform file check.

  - src
  - dep
  - constr
  - tb
  - dep_tb
  - IN_FILE_NAME=8bit_count.bin
  - OUT_FILE_NAME=out_8bit_count.bin
  - RAND_FULL
  - FIFO_DEPTH

3.3.2 wishbone_classic_block_ram Targets

- default

  Info: Default for IP intergration.

  - src
  - dep

- sim

  Info: Default for IP intergration.

  - src
  - dep
  - tb
  - dep_tb

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

There are two different BUS block RAM modules that can be used in a project.

- **axi_lite_block_ram** AXI lite block RAM
- **wishbone_classic_block_ram** Wishbone Classic block RAM

The next sections document the module in great detail.

axi_lite_block_ram.v

AUTHORS

JAY CONVERTINO

DATES

2024/03/07

INFORMATION

Brief

axi lite block ram

License MIT

axi_lite_block_ram

```
module axi_lite_block_ram #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
DEPTH
```

```
  =
512,
parameter
RAM_TYPE
  =
"block",
parameter
HEX_FILE
  =
""
) ( input aclk, input arstn, input s_axi_awvalid, input [ADDRESS_WIDTH-1:0]
```

axi lite block ram

Parameters

ADDRESS_WIDTH parameter	Width of the axi address bus in bits.
BUS_WIDTH parameter	Bus width for data paths in bytes.
DEPTH parameter	Depth of the RAM in terms of data width words.
RAM_TYPE parameter	Used to set the ram_style atribute.
HEX_FILE parameter	Hex file to write to RAM.

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

up_rreq

```
wire up_rreq
```

uP read bus request

up_rack

```
reg up_rack
```

uP read bus acknowledge

up_raddr

```
wire [ADDRESS_WIDTH-3:0] up_raddr
```

uP read bus address

up_rdata

```
wire [(
BUS_WIDTH*4
)-1:0] up_rdata
```

uP read bus request

up_wreq

```
wire up_wreq
```

uP write bus request

up_wack

```
reg up_wack
```

uP write bus acknowledge

up_waddr

```
wire [ADDRESS_WIDTH-3:0] up_waddr
```

uP write bus address

up_wdata

```
wire [(
BUS_WIDTH*4
)-1:0] up_wdata
```

uP write bus data

INSTANTIANTED MODULES

inst_up_axi

```
up_axi inst_up_axi (
.up_rstn                        (arstn),
.up_clk                         (aclk),
.up_axi_awvalid(s_axi_awvalid),
.up_axi_awaddr(s_axi_awaddr),
.up_axi_awready(s_axi_awready),
.up_axi_wvalid(s_axi_wvalid),
.up_axi_wdata(s_axi_wdata),
.up_axi_wstrb(s_axi_wstrb),
.up_axi_wready(s_axi_wready),
.up_axi_bvalid(s_axi_bvalid),
.up_axi_bresp(s_axi_bresp),
.up_axi_bready(s_axi_bready),
.up_axi_arvalid(s_axi_arvalid),
.up_axi_araddr(s_axi_araddr),
.up_axi_arready(s_axi_arready),
.up_axi_rvalid(s_axi_rvalid),
.up_axi_rresp(s_axi_rresp),
.up_axi_rdata(s_axi_rdata),
.up_axi_rready(s_axi_rready),
.up_wreq(up_wreq),
.up_waddr(up_waddr),
.up_wdata(up_wdata),
.up_wack(up_wack),
```

```
                                                                        .
 up_rreq(up_rreq),                                                      .
 up_raddr(up_raddr),                                                    .
 up_rdata(up_rdata),                                                    .
 up_rack(up_rack)
)
```

Module instance of up_axi for the AXI Lite bus to the uP bus.

inst_dc_block_ram

```
 dc_block_ram #(
                                                                        .
 RAM_DEPTH(DEPTH),                                                      .
 BYTE_WIDTH(BUS_WIDTH),                                                 .
 ADDR_WIDTH(ADDRESS_WIDTH),                                             .
 HEX_FILE(HEX_FILE),                                                    .
 RAM_TYPE(RAM_TYPE)
) inst_dc_block_ram ( .rd_clk(aclk), .rd_rstn(arstn), .rd_en(up_rreq), .rd_
```

Module instance of dc_block_ram that connects to the uP BUS directly.

wishbone_classic_block_ram.v

AUTHORS

JAY CONVERTINO

DATES

2024/03/07

INFORMATION

Brief

Wishbone classic block RAM core.

License MIT

wishbone_classic_block_ram

```
module wishbone_classic_block_ram #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
DEPTH
```

```
  =
512,
parameter
RAM_TYPE
  =
"block",
parameter
HEX_FILE
  =
""
) ( input clk, input rst, input s_wb_cyc, input s_wb_stb, input s_wb_we, inp
```

Wishbone classic block RAM core.

Parameters

ADDRESS_WIDTH parameter — Width of the axi address bus in bits.

BUS_WIDTH parameter — Bus width for data paths in bytes.

DEPTH parameter — Depth of the RAM in terms of data width words.

RAM_TYPE parameter — Used to set the ram_style atribute.

HEX_FILE parameter — Hex file to write to RAM.

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_bte	Burst Type Extension
s_wb_cti	Cycle Type
s_wb_ack	Bus transaction terminated
s_wb_data_o	Output data
s_wb_err	Active high when a bus error is present

c_PWR_RAM

```
localparam c_PWR_RAM = clogb2(
DEPTH
)
```

power of 2 conversion of DEPTH

c_RAM_DEPTH

```
localparam c_RAM_DEPTH = 2 ** c_PWR_RAM
```

create RAM depth based on power of two depth size.

up_rreq

```
wire up_rreq
```

uP read bus request

up_rack

```
reg up_rack
```

uP read bus acknowledge

up_raddr

```
wire [ADDRESS_WIDTH-3:0] up_raddr
```

uP read bus address

up_rdata

```
wire [(
BUS_WIDTH*4
)-1:0] up_rdata
```

uP read bus request

up_wreq

```
wire up_wreq
```

uP write bus request

up_wack

```
reg up_wack
```

uP write bus acknowledge

up_waddr

```
wire [ADDRESS_WIDTH-3:0] up_waddr
```

uP write bus address

up_wdata

```
wire [(
BUS_WIDTH*4
)-1:0] up_wdata
```

uP write bus data

INSTANTIANTED MODULES

inst_up_wishbone_classic

```
up_wishbone_classic #(
                                                                                .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                                .
BUS_WIDTH(BUS_WIDTH)
) inst_up_wishbone_classic ( .clk(clk), .rst(rst), .s_wb_cyc(s_wb_cyc), .s_w
```

Module instance of up_wishbone_classic for the Wishbone Classic bus to the uP bus.

inst_dc_block_ram

```
dc_block_ram #(
                                                                                .
RAM_DEPTH(c_RAM_DEPTH),
                                                                                .
BYTE_WIDTH(BUS_WIDTH),
                                                                                .
ADDR_WIDTH(c_PWR_RAM),
                                                                                .
HEX_FILE(HEX_FILE),
                                                                                .
RAM_TYPE(RAM_TYPE)
) inst_dc_block_ram ( .rd_clk(clk), .rd_rstn(~rst), .rd_en(up_rreq), .rd_dat
```

Module instance of dc_block_ram that connects to the uP BUS directly.