DC_BLOCK_RAM

May 21, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

Dual clock block RAM for any FPGA target. Includes a byte enable for selecting bytes to write from the bus.

1.2 Dependencies

The following are the dependencies of the cores.

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

1.2.1 fusesoc_info Depenecies

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

1.3 In a Project

Connect the device using the read write signals see **??** for details

2 Architecture

This core is made up of a single module.

- **ft245_sync_to_axis** Interface AXIS to F245 device (see core for documentation).

This core has 2 always blocks that are sensitive to the positive clock edge.

- **Produce Data** Takes write input data and stores it in RAM at a specified address. BE will filter out bytes if the corresponding bits not set to active high.
- **Consume Data** Read data from RAM at a specified address and output over read interface.

Please see **??** for information on read/write interface ports.

3 Building

The DC block RAM 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. Linting is performed by verible using the lint target.

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 fusesoc_info File List

- src
  - src/dc_block_ram.v
- tb
  - 'tb/tb_dc_block_ram.v': 'file_type': 'verilogSource'
- tb_cocotb
  - 'tb/tb_cocotb.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb.v': 'file_type': 'verilogSource'

3.3 Targets

3.3.1 fusesoc_info Targets

- default

  Info: Default for IP intergration.

- lint

  Info: Lint with Verible

- sim

Info: Verilog Sim

- 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

4 Simulation

There are a few different simulations that can be run for this core. The backend used for testing is iverilog for verilog or cocotb simulations. Usually GTKWave is used to view the fst waveform output. Cocotb are the unit tests that attempt to give a pass/fail verification to the core operation.

4.1 iverilog

iverilog is used for simple test benches for quick visual verification of the core. This will autofinish after it has run up to a certain number of words have been output.

4.2 cocotb

This method allows for quick writing of test benches that actually assert and check the state of the core. These tests are much more conclusive since it will run all test vectors and generate a report if they pass or fail. All tests output waves to a single fst file. The method of launching the tests is to use fusesoc. These have not been written to use a python runner method or makefiles. To use the cocotb tests you must install the following python libraries.

```
$ pip install cocotb
```

The targets available are listed below.

- **sim_cocotb** Standard simulation for dc$_{b}lock_{r}am$.
  The targets above can be run with various parameters. This test will check the input/output against each other to validate core operation.

```
$ fusesoc run ––target sim_cocotb AFRL:ram:dc_block_ram:1.0.0
```

5 Module Documentation

- **dc_block_ram** Generic dual clock block RAM
- **tb_dc_block_ram-v** Verilog test bench
- **tb_cocotb-py** Cocotb python test routines
- **tb_cocotb-v** Cocotb verilog test bench

The next sections document the module.

dc_block_ram.v

AUTHORS

JAY CONVERTINO

DATES

2024/03/07

INFORMATION

Brief

Generic Dual Port RAM

License MIT

dc_block_ram

```
module dc_block_ram #(
parameter
RAM_DEPTH
 =
1,
parameter
BYTE_WIDTH
 =
1,
parameter
ADDR_WIDTH
 =
1,
parameter
```

```
HEX_FILE
 =
"",
parameter
RAM_TYPE
 =
"block"
) ( input rd_clk, input rd_rstn, input rd_en, output [(BYTE_WIDTH*8)-1:0] r
```

Generic Dual Port RAM

Parameters

RAM_DEPTH parameter	Number of words using the size of BYTE_WIDTH.
BYTE_WIDTH parameter	Width of the data bus in bytes.
ADDR_WIDTH parameter	Width of the address bus in bits.
HEX_FILE parameter	Read a hex value text file as the initial state of the RAM.
RAM_TYPE parameter	Used to set the ram_style atribute.

Ports

rd_clk	Read clock positive edge
rd_rstn	Read reset active low
rd_en	Read enable active high
rd_data	Read data output
rd_addr	Read data address select
wr_clk	Write clock positive edge
wr_rstn	Write reset active low
wr_en	Write enable active high
wr_ben	Write byte enable, each bit represents one byte of write data.
wr_data	Write data input
wr_addr	Write data address select

tb_dc_block_ram.v

AUTHORS

JAY CONVERTINO

DATES

2025/01/17

INFORMATION

Brief

Test bench for Generic Dual Port RAM

License MIT

dc_block_ram

```
module tb_dc_block_ram #(
parameter
RAM_DEPTH
 =
256,
parameter
BYTE_WIDTH
 =
4,
parameter
ADDR_WIDTH
 =
32,
parameter
```

```
HEX_FILE
 =
"",
parameter
RAM_TYPE
 =
"block"
)()
```

Test bench for Generic Dual Port RAM

Parameters

RAM_DEPTH Number of words using the size of BYTE_WIDTH.
parameter

BYTE_WIDTH Width of the data bus in bytes.
parameter

ADDR_WIDTH Width of the address bus in bits.
parameter

HEX_FILE Read a hex value text file as the initial state of the RAM.
parameter

RAM_TYPE Used to set the ram_style atribute.
parameter

INSTANTIATED MODULES

clk_stim

```
clk_stimulus #(
                                                                                .
CLOCKS(1),
                                                                                .
CLOCK_BASE(1000000),
                                                                                .
RESETS(1),
                                                                                .
RESET_BASE(2000)
) clk_stim ( .clkv(tb_dut_clk), .rstnv(tb_dut_rstn), .rstv() )
```

Generate a 50/50 duty cycle set of clocks and reset.

inst_dc_block_ram

```
dc_block_ram #(
                                                                                .
RAM_DEPTH(RAM_DEPTH),
                                                                                .
BYTE_WIDTH(BYTE_WIDTH),
                                                                                .
ADDR_WIDTH(ADDR_WIDTH),
                                                                                .
HEX_FILE(HEX_FILE),
                                                                                .
RAM_TYPE(RAM_TYPE)
) inst_dc_block_ram ( .rd_clk(tb_dut_clk), .rd_rstn(tb_dut_rstn), .rd_en(tb_
```

Module instance of dc_block_ram

tb_cocotb.py

AUTHORS

JAY CONVERTINO

DATES

2025/05/21

INFORMATION

Brief

Cocotb test bench

License MIT

FUNCTIONS

random_bool

```
def random_bool()
```

Return a infinte cycle of random bools

Returns: List

start_clock

```
def start_clock(
dut
)
```

Start the simulation clock generator.

Parameters

dut Device under test passed from cocotb test function

reset_dut

```
async def reset_dut(
dut
)
```

Cocotb coroutine for resets, used with await to make sure system is reset.

random test

Coroutine that is identified as a test routine. Write random data, on one clock edge, read on the next.

Parameters

dut Device under test passed from cocotb.

in_reset

```
@cocotb.test()
async def in_reset(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if device stays in unready state when in reset.

Parameters

dut Device under test passed from cocotb.

no_clock

```
@cocotb.test()
async def no_clock(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if no ready when clock is lost and device is left in reset.

Parameters

dut Device under test passed from cocotb.

tb_cocotb.v

AUTHORS

JAY CONVERTINO

DATES

2025/05/21

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
parameter
RAM_DEPTH
 =
1,
parameter
BYTE_WIDTH
 =
1,
parameter
ADDR_WIDTH
 =
1,
parameter
```

```
HEX_FILE
 =
"",
parameter
RAM_TYPE
 =
"block",
parameter
RD_CLOCK_SPEED
 =
10000000,
parameter
WR_CLOCK_SPEED
 =
10000000
) ( input rd_clk, input rd_rstn, input rd_en, output [(BYTE_WIDTH*8)-1:0] rd
```

Generic Dual Port RAM wrapper

Parameters

RAM_DEPTH parameter	Number of words using the size of BYTE_WIDTH.
BYTE_WIDTH parameter	Width of the data bus in bytes.
ADDR_WIDTH parameter	Width of the address bus in bits.
HEX_FILE parameter	Read a hex value text file as the initial state of the RAM.
RAM_TYPE parameter	Used to set the ram_style atribute.
RD_CLOCK_SPEED parameter	COCOTB CLOCK SPEED FOR READ IN HZ.
WR_CLOCK_SPEED parameter	COCOTB CLOCK SPEED FOR WRITE IN HZ.

Ports

rd_clk	Read clock positive edge
rd_rstn	Read reset active low
rd_en	Read enable active high
rd_data	Read data output
rd_addr	Read data address select
wr_clk	Write clock positive edge
wr_rstn	Write reset active low
wr_en	Write enable active high
wr_ben	Write byte enable, each bit represents one byte of write data.
wr_data	Write data input
wr_addr	Write data address select

INSTANTIATED MODULES

dut

```
dc_block_ram #(
                                                                                .
RAM_DEPTH(RAM_DEPTH),
                                                                                .
BYTE_WIDTH(BYTE_WIDTH),
                                                                                .
ADDR_WIDTH(ADDR_WIDTH),
                                                                                .
HEX_FILE(HEX_FILE),
                                                                                .
RAM_TYPE(RAM_TYPE)
) dut ( .rd_clk(rd_clk), .rd_rstn(rd_rstn), .rd_en(rd_en), .rd_data(rd_data)
```

Device under test, dc_block_ram