

# wishbone\_classic\_block\_ram.v

---

## AUTHORS

---

JAY CONVERTINO

---

## DATES

---

2024/03/07

---

## INFORMATION

---

### Brief

---

Wishbone classic block RAM core.

### 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.

## 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, input
```

Wishbone classic block RAM core.

## Parameters

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

## Ports

<b>clk</b>	Clock for all devices in the core
<b>rst</b>	Positive reset
<b>s_wb_cyc</b>	Bus Cycle in process
<b>s_wb_stb</b>	Valid data transfer cycle
<b>s_wb_we</b>	Active High write, low read
<b>s_wb_addr</b>	Bus address
<b>s_wb_data_i</b>	Input data
<b>s_wb_sel</b>	Device Select
<b>s_wb_bte</b>	Burst Type Extension
<b>s_wb_cti</b>	Cycle Type
<b>s_wb_ack</b>	Bus transaction terminated
<b>s_wb_data_o</b>	Output data
<b>s_wb_err</b>	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-(  
ADDRESS_WIDTH  
16  
)-1:0] 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-(  
ADDRESS_WIDTH
```

```
16
)-1:0] up_waddr
```

uP write bus address

## up\_wdata

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

uP write bus data

## INSTANTIATED 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_v
```

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.