Versatile FIFO

A versatile FIFO supporting both sync and async implementations with multiple parallell channels

Brought to You By ORSoC / OpenCores

Legal Notices and Disclaimers

Copyright Notice

General Disclaimer

The Publisher has strived to be as accurate and complete as possible in the creation of this ebook, notwithstanding the fact that he does not warrant or represent at any time that the contents within are accurate due to the rapidly changing nature of information.

The Publisher will not be responsible for any losses or damages of any kind incurred by the reader whether directly or indirectly arising from the use of the information found in this ebook.

This ebook is not intended for use as a source of legal, business, accounting, financial, or medical advice. All readers are advised to seek services of competent professionals in the legal, business, accounting, finance, and medical fields.

No guarantees of any kind are made. Reader assumes responsibility for use of the information contained herein. The Publisher reserves the right to make changes without notice. The Publisher assumes no responsibility or liability whatsoever on the behalf of the reader of this report.

Distribution Rights

The Publisher grants you the following rights for re-distribution of this ebook.

[YES] Can be given away.
[YES] Can be packaged.
[YES] Can be offered as a bonus.
[NO] Can be edited completely and your name put on it.
[YES] Can be used as web content.
[NO] Can be broken down into smaller articles.
[NO] Can be added to an e-course or auto-responder as content.
[NO] Can be submitted to article directories (even YOURS) IF at least half is rewritten!
[NO] Can be added to paid membership sites.
[NO] Can be added to an ebook/PDF as content.
[NO] Can be offered through auction sites.
[NO] Can sell Resale Rights.
[NO] Can sell Master Resale Rights.
[NO] Can sell Private Label Rights.

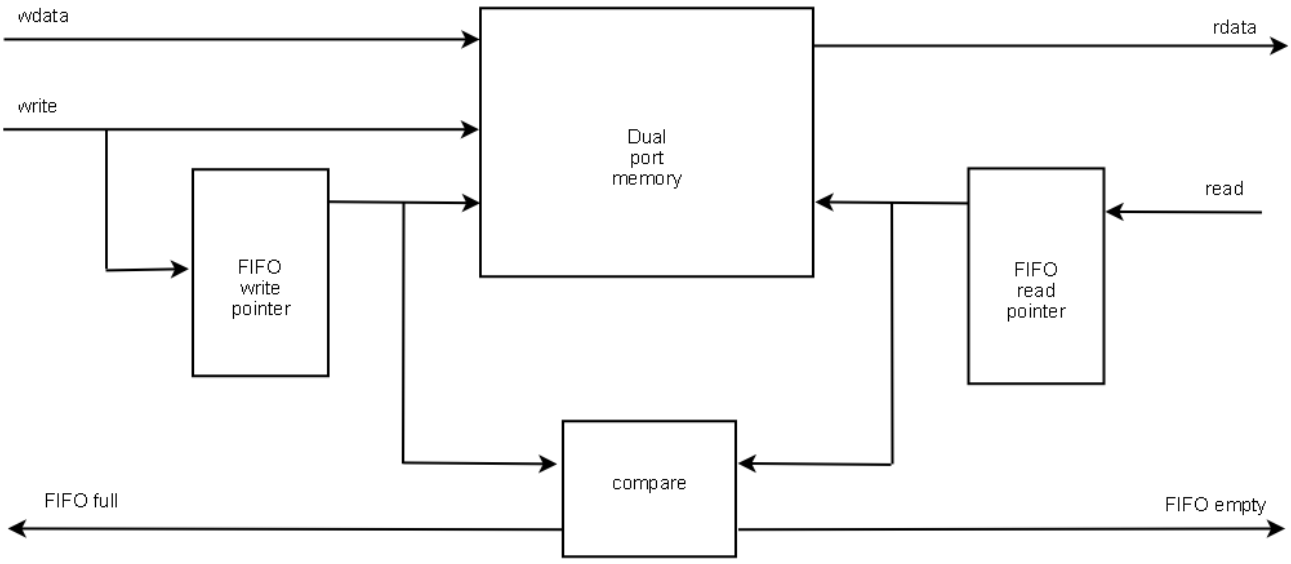
Table of Contents

Chapter 1 Introduction

The FIFO implementation outlined in this document can easily be configured to suit the following

- asynchronous FIFO with different clock domains for read and write sides
- synchronous FIFO with programmable flags
- multiple FIFO sharing the same memory resource

Asynchronous FIFO implementation

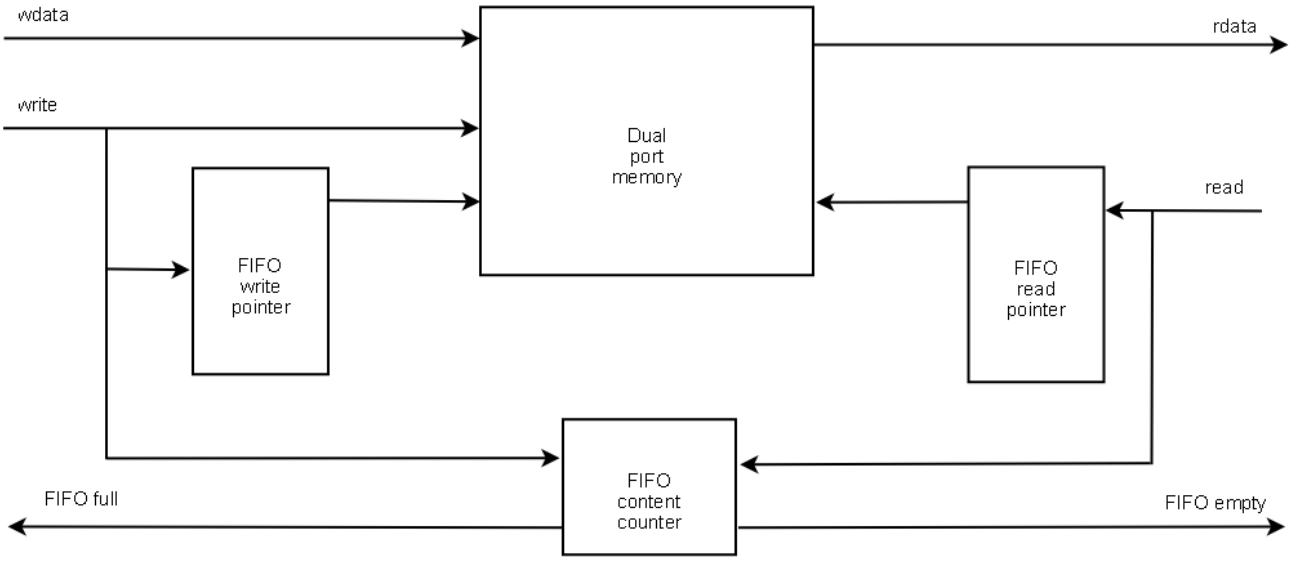


This configuration uses gray counter as FIFO pointers and an asynchronous compare function. The read and write pointers are within different clock domains. To be able to have glitch free compare function it is important that the pointers are clocked signals and that no more than one signal can change its value on any clock signal.

For read and write pointers use "Versatile counter" found at OpenCores. Configured as gray counter.

This FIFO has one clock domain for the write side and an other for the read side.

Synchronous FIFO implementation



For minimal area and highest performance this implementation uses linear feedback shift registers, LFSR, as memory pointers. The logic generating FIFO flags could either be a compare function based on q_next from address counters or based on an up/down counter keeping track of number of data words in memory.

For read and write pointers use "Versatile counter" found at OpenCores. Configured as LFSR counter. This applies also to FIFO content counter.

This FIFO has one clock domain for the write side and the read side.

Multiple FIFO implementation

In some cases more than one FIFO can share the same memory. This makes better use of the FPGA resources. Most FPGA has built-in memories with 4 kbits to 8 kbits. These memories can in many cases be configured as true dual port memories, that is with read and write possibilities on both sides.

This is the case for the following FPGA families

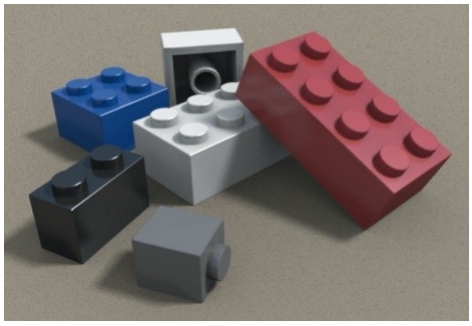
- ACTEL ProASIC3
- ALTERA Cyclone III

Many application have one wishbone interface and many FIFO channels with different type of real time data. In this case only one FIFO can be written to from the system bus side meaning that there is no problem sharing the dual port memory.

An example of a system that can use this type of implementation is a system with an interface towards an external AC´97 compatible audio codec. The external interface is a bit stream with audio data. System bus interface could have 6 FIFO channels (for 5.1 audio support). From the system side there will be 6 individual FIFO queues. Internally all queues can share one memory instance. A FSM will read out audio data from the FIFO sequentially.

Chapter 2 FIFO building blocks

All FIFO implementations are based on the following building blocks. Each block is available as a Verilog RTL module.



1. dual port memory
2. read and write pointers
3. logic generating FIFO empty and FIFO full and optionally other FIFO flags
4. optional multiplexer used for multiple FIFO queues sharing a common memory

From this a large number of different FIFO can be constructed. Pick your bricks

Dual port memory

There are a few variants of the dual port memory implementations. The memory could have read/write support on both sides (true dual port) or have one read and one write side. Also there can be one individual clock signals for read or write or one clock per side.

All variants come from the same source, *dual_port_ram.v*". The Makefile in rtl/verilog builds all targets, make dual_port_ram.

Single clock dual port RAM

A dual port memory with one write side and one read side with one common read and write clock.

Single clock dual port RAM	
Filename	versatile_fifo_dual_port_ram_sc_1w.v
Module name	versatile_fifo_dual_port_ram_sc_1w
Parameters	**default value**
DATA_WIDTH	8
ADDR_WIDTH	9
A side	
d_a	input
adr_a	input
we_a	input
B side	
q_b	output
adr_b	output
common	
clk	input

Single clock true dual port RAM

A dual port memory with two read/write sides with one common clock.

Single clock dual port RAM	
Filename	versatile_fifo_dual_port_ram_sc_1w.v
Module name	versatile_fifo_dual_port_ram_sc_1w
Parameters	**default value**
DATA_WIDTH	8
ADDR_WIDTH	9
A side	
d_a	input
q_a	output
adr_a	input
we_a	input
B side	
d_a	input
q_b	output
adr_b	output
we_b	input
common	
clk	input

Dual clock dual port RAM

A dual port memory with one write side and one read side with individual read and write clock.

Single clock dual port RAM	
Filename	versatile_fifo_dual_port_ram_sc_1w.v
Module name	versatile_fifo_dual_port_ram_sc_1w
Parameters	**default value**
DATA_WIDTH	8
ADDR_WIDTH	9
A side	
d_a	input
adr_a	input
we_a	input
clk_a	input
B side	
q_b	output
adr_b	output
clk_b	input

Dual clock true dual port RAM

A dual port memory with two read/write sides with individual clocks.

Single clock dual port RAM	
Filename	versatile_fifo_dual_port_ram_sc_1w.v
Module name	versatile_fifo_dual_port_ram_sc_1w
Parameters	**default value**
DATA_WIDTH	8
ADDR_WIDTH	9
A side	
d_a	input
q_a	output
adr_a	input
we_a	input
clk_a	input
B side	
d_a	input
q_b	output
adr_b	output
we_b	input
clk_b	input

Read and write pointers

The read and write pointer are two instances of the same type of counters.

For FIFO implementations with two clock domains gray counter must be used to avoid glitches on the FIFO flags.

For FIFO implementations with one clock domain either binary or LFSR counters could be used. LFSR counters have one state shorter cycle but require less area and have no carry chains. LFSR should be used when low area usage and/or high performance is important, binary when memory depth is most important.

Note:
For gray type counter an optional binary count output can be used as write address while the gray output is used for FIFO full and empty indications.

For all different counter types use *Versatile counter*. IP can be found at OpenCores.org together with information on how to build application specific implementations.

FIFO flag generation

There are two different cases, one or two clock domains. For two clock domains the compare logic will be asynchronous.

Programmable flags for FIFO with two clock domains are not supported in the current release of versatile FIFO but might be included in future releases.

FIFO with one clock domain can have an optional up/down counter that keep track of number of words currently in FIFO. From this counter additional FIFO flags can be generated.

Asynchronous compare

The write pointer holds the next address to write to.

The read pointer holds the current address to read from.

When the write pointer equals the read pointer the FIFO is either empty or full. We need something to distinguish empty or full condition. The memory can be divided into four quadrants. The MSB of the pointers will have the following bit pattern (gray code):

Quadrant	Pattern
Q1	00
Q2	01
Q3	11
Q4	10

We use signal *direction* to indicate whether FIFO is filling up or going empty.

- *direction* = 1 when write pointer is one quadrant behind read pointer
- *direction* = 0 when read pointer is one quadrant behind write pointer
- direction = 0 upon reset, in this case FIFO is empty

Direction set condition

write pointer		read pointer		direction_set
Q1	00	Q2	01	1
Q2	01	Q3	11	1
Q3	11	Q4	10	1
Q4	10	Q1	00	1
default				0

Direction clear condition

write pointer		read pointer		direction_clr
Q1	00	Q4	10	1
Q2	01	Q1	00	1
Q3	11	Q2	01	1
Q4	10	Q3	11	1
default				0

An SR-type flip-flop is used to generate signal *direction*.

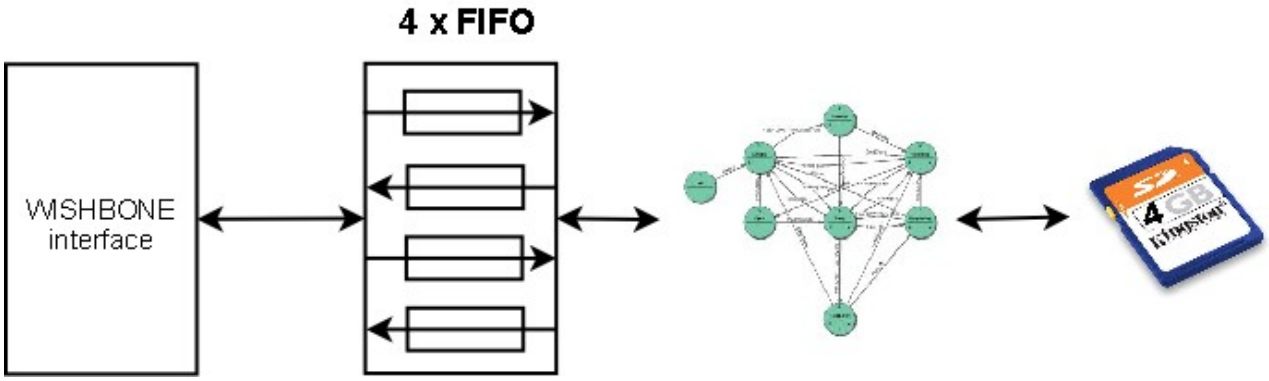
Setting of *direction* is synchronous to write clock, clearing is synchronous to read clock. Dual flip-flops clocked by write clock with asynchronous reset from

async empty synchronize FIFO flag. Analogous for empty flag.

Single clock dual port RAM	
Filename	versatile_fifo_async_cmp.v
Module name	versatile_fifo_async_cmp
Parameters	**default value**
ADDR_WIDTH	4
IO signals	
wptr, rptr	input
fifo_empty, fifo_full	output
wclk, rclk, rst	input

Chapter 3 Example implementations

SD FLASH controller



This example is a SD FLASH controller with a wishbone interface. We want to have FIFO queues for the following

1. commands going from wishbone to SD card
2. command response going from SD card to wishbone
3. write data going from wishbone to SD card
4. read data going from SD card to wishbone

The wishbone side will have on clock domain and the SD card side an other. This makes it possible to run the SD card at maximum specified speed.

All FIFO queues should share the same memory resource. The length of the queues should be 512 bytes each.

The FIFO module will have the following IO signals:

Single clock dual port RAM	
Filename	sd_flash_fifo.v
Module name	sd_flash_fifo
wishbone side signals	
wb_adr_i[1:0]	to select FIFO queue
wb_dat_i, wb_dat_o	data buses
wb_re, wb_we	read and write enable
fifo1_full, fifo2_empty, fifo3_full, fifo4_empty	FIFO flags
wb_clk	wishbone clock
SD side signals	
sd_adr_i[1:0]	to select FIFO queue
sd_dat_i, sd_dat_o	data buses
sd_re, sd_we	read and write enable
fifo1_empty, fifo2_full, fifo3_empty, fifo4_full	FIFO flags
sd_clk	sd clock

Submodules

The design will use the following modules

1. read and write pointers, 8 instances
   9 bit gray counter with binary outputs
   define file: sd_counter_defines.v
   outfile : sd_counter.v
2. async compare, 4 instances
3. dual port memory; dual clock, dual way
   2048 x 8

To generate design

```
make sd
```

Recommended Resources

ORSoC – http://www.orsoc.se

ORSoC is a fabless ASIC design & manufacturing services company, providing RTL to ASIC design services and silicon fabrication service. **ORSoC** are specialists building complex system based on the OpenRISC processor platform.

Open Source IP – http://www.opencores.org

Your number one source for open source IP and other FPGA/ASIC related information.