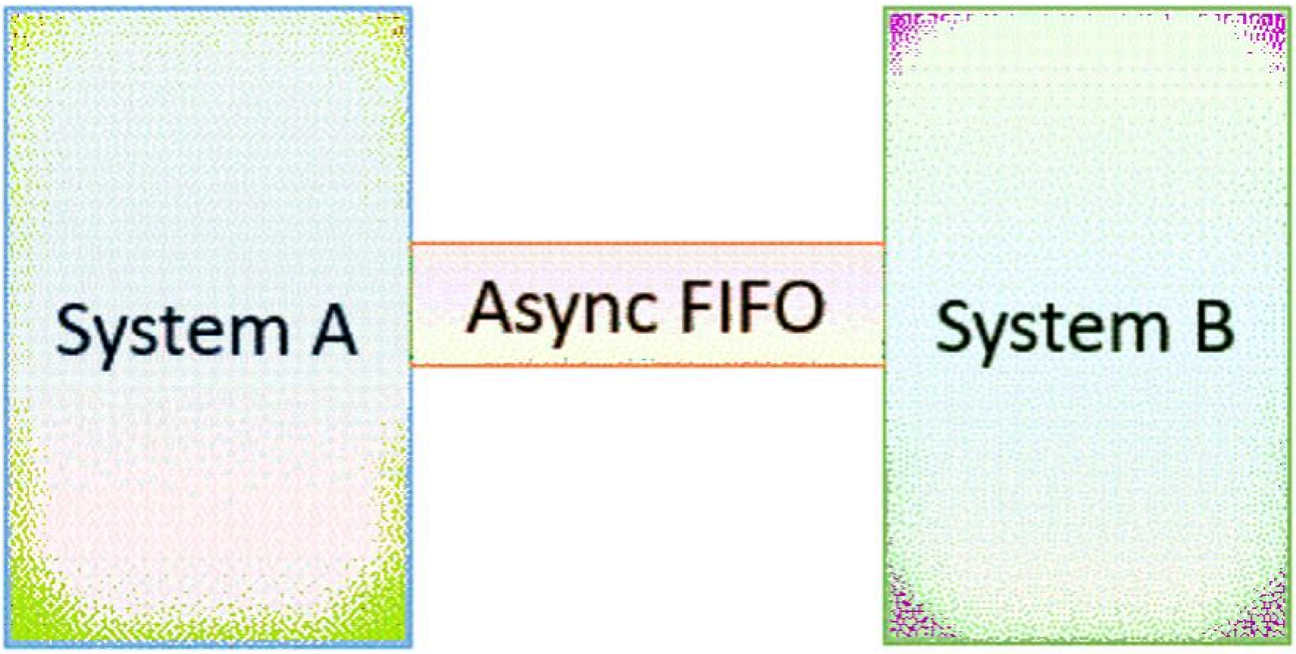

System A
Async FIFO
System B

PROJECT = ①

Asti_tyagi_verilog

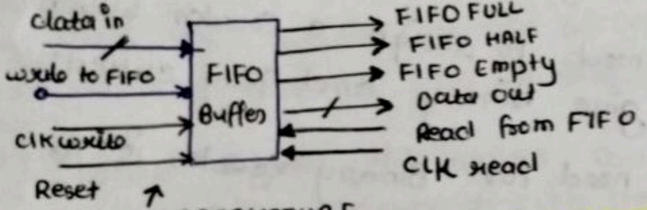
Asynchronous FIFO DESIGN

• Asyncronous FIFO :-

- write to the FIFO and READ from the FIFO happen on different clocks.
- most of the times, we use Asyncronous FIFO.

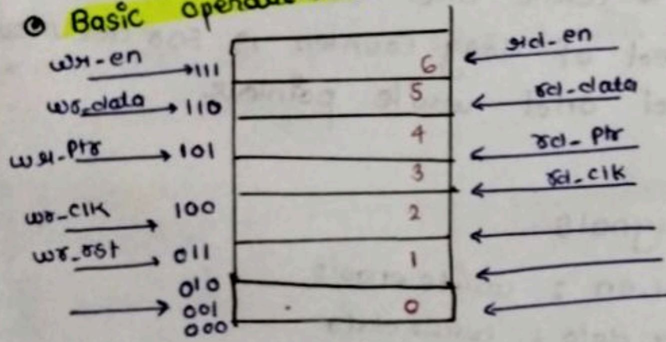
• FIFO are often used to safely pass data from one clk domain to another asyncronous clk domain.

• wherein two clock domain are asyncronous to each other.



① FIFO STRUCTURE

◉ Basic operation of asyncronous FIFO :-



• we have two pointer in particular which will control whole operations the pointers are
  1.) read pointer
  11.) write pointer

∴ So read pointer is used to read the data from the FIFO

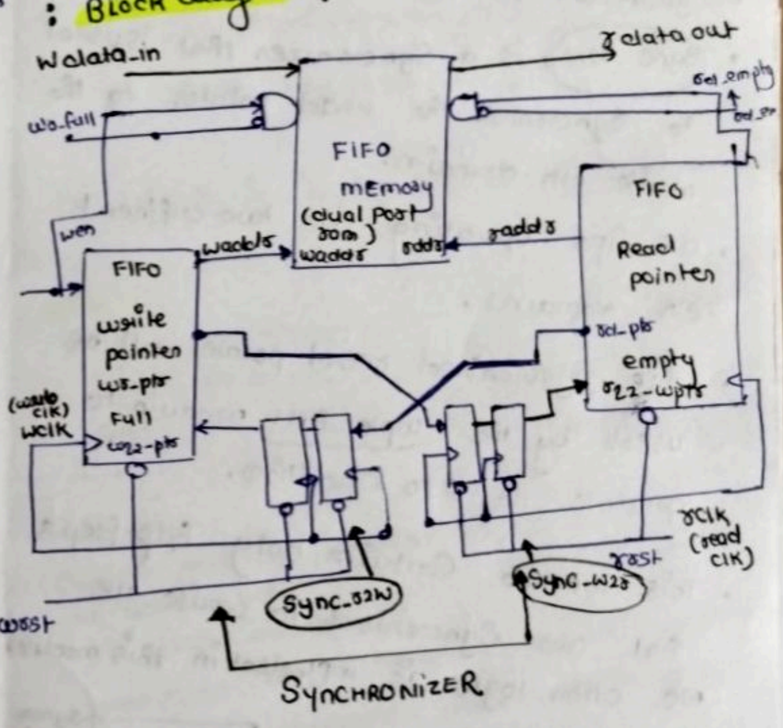
∴ write pointer is used to write the data to the point.

• read pointer and write pointer are controlled by using write clk and write enable clk.

- So write enable and write clk and controlled the read pointer.

- whereas read pointer (rd_ptr) is controlled by read enable (rd_en) and read_clk (rd_clk)

: Block diagram :-



:- in this digram consists of three module
  1.) memory module
  2.) write pointer module
  3.) read pointer module

- for FIFO memory, we need to have a write pointer to write data so this is called write pointer

- we need to read the data from FIFO memory we have read pointer module.

Aditi tyagi_verilog

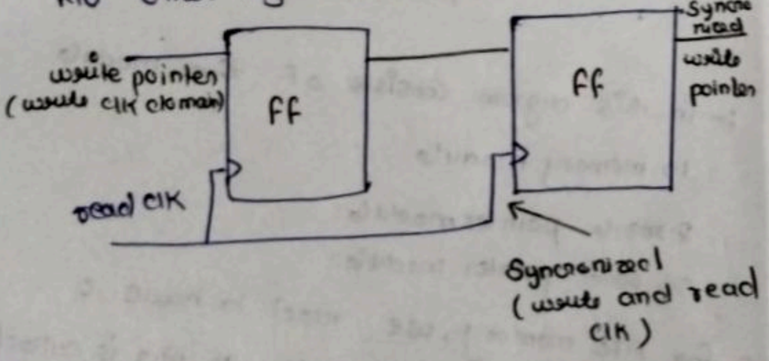
- This read pointer module and write pointer module operates two difference clk. that is wclk (write clk) and rclk (read clk)

→ so two modules communicate on two different clocks so we send data the data will be not syncronized so it will get wrong data. or mismatch.

→ we use to syncronizers.

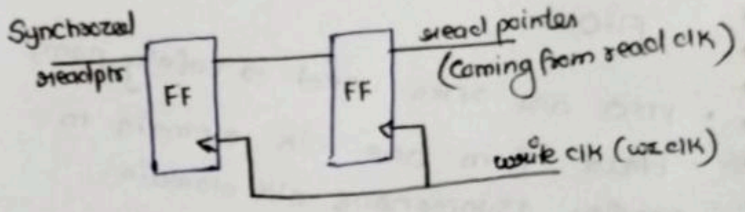
⊙ SYNCHRONIZER :-

- Syncronizers are made of 2 flip-flops.
- Sync_r2w, is a syncronizer that is used to syncronize the read pointer to the write clk domain.
- as fifo operating at two different clk domains.
- the syncronized read pointer will be used by the wptr_full module to generate the FIFO condition.
- This module contains only flip-flops that are syncronized to write clk. No other logic is included in this module.



- Sync_w2r, this syncronizor module that is used to syncronize the write pointer to read clk domin.

- this is used write pointer will be rptr_empty module generate FIFO empty condition.
- This is used only contains flip-flop that are syncronized to the read clk. No other logic is include in this module.



⊙ Synchronizers help to reduces metastability to a great extent.

- we need to design a counter which can give Binary and Gray outputs.
- The need for Binary counter is to address the FIFO MEMORY. i.e → write and read address. & need of Gray counter is for addressing read and write pointers.

⊙ Signals

wr_en	:	write enable
wr_data	:	write data
Full	:	FIFO is full
empty	:	FIFO is empty
rd_en	:	read enable
rd_data	:	read data
	:	binary write pointer
	:	gray write pointer

⊙ FLAG IN FIFO:

- Asyncronous FIFO provides us with following two Flags. to determine the status and to intrupt the operation of FIFO.

1.) EMPTY flag:-

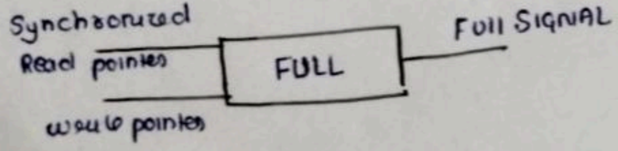
∴ This flag is usefull to avoid the case of invalid request of read opration when the FIFO is already empty.

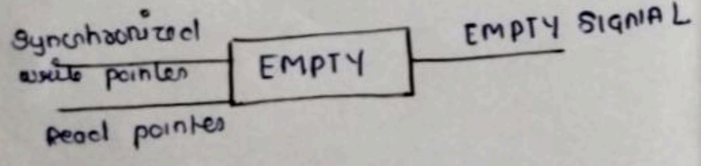
2.) FULL flag:-

∴ This flag is usefull to avoid the case of invalid request of write operation when the FIFO is already full

∴ when ~~signal~~ the status counter reach the maximum FIFO depth it will assert FIFO Full Signal &

- when its value zero it will assert FIFO empty





⊙ PointER To Control OPERATION:

- write pointer

- The write pointer always points to the next word to be written, therefor on reset, both pointer are set to zero which also happens to be next FIFO word location to be written

```
if (write pointer == {~Syncronized Read
        pointer [4:3], Syncronized Read pointer
                                        [2:0]})
    then   Full=1;
```

Read pointer:

→ The output of Syncronizer gray-wptr-syn is fed as an input to the Read pointer to generate Empty condtion:

```
if( Synchronized write pointer == Read pointer) &&
    (Synchronized write ptr [3:0] == Read ptr[3:0]))
    then   EMPTY=1
```

∴ [This is Empty and Full logic generation]

NOTE:-

- Status Signals :-
  Full : High when FIFO is Full
  EMPTY : High when FIFO is empty.

- CounTER:-
  Counter will be incremented if :
  ↔ write takes place and read pointer take place. means
  whenever write happens, increment wr-ptr by 1
  whenever read happen, increment rd-ptr by 1.

⊙ SYNCHRONIZER :-

- Synchronizers are made of two DFlip flop as fifo operating 2 different clock frequency so there is a need to Synchronize write and read pointers for generate empty and full flag.

@ why Synchronization is need ?

- wr-ptr work with write_clk → So we need read pointer.
- rd-ptr work with read_clk So we need write pointer.

@ Synchronizes help to avoid metastablity.

NOTE :-

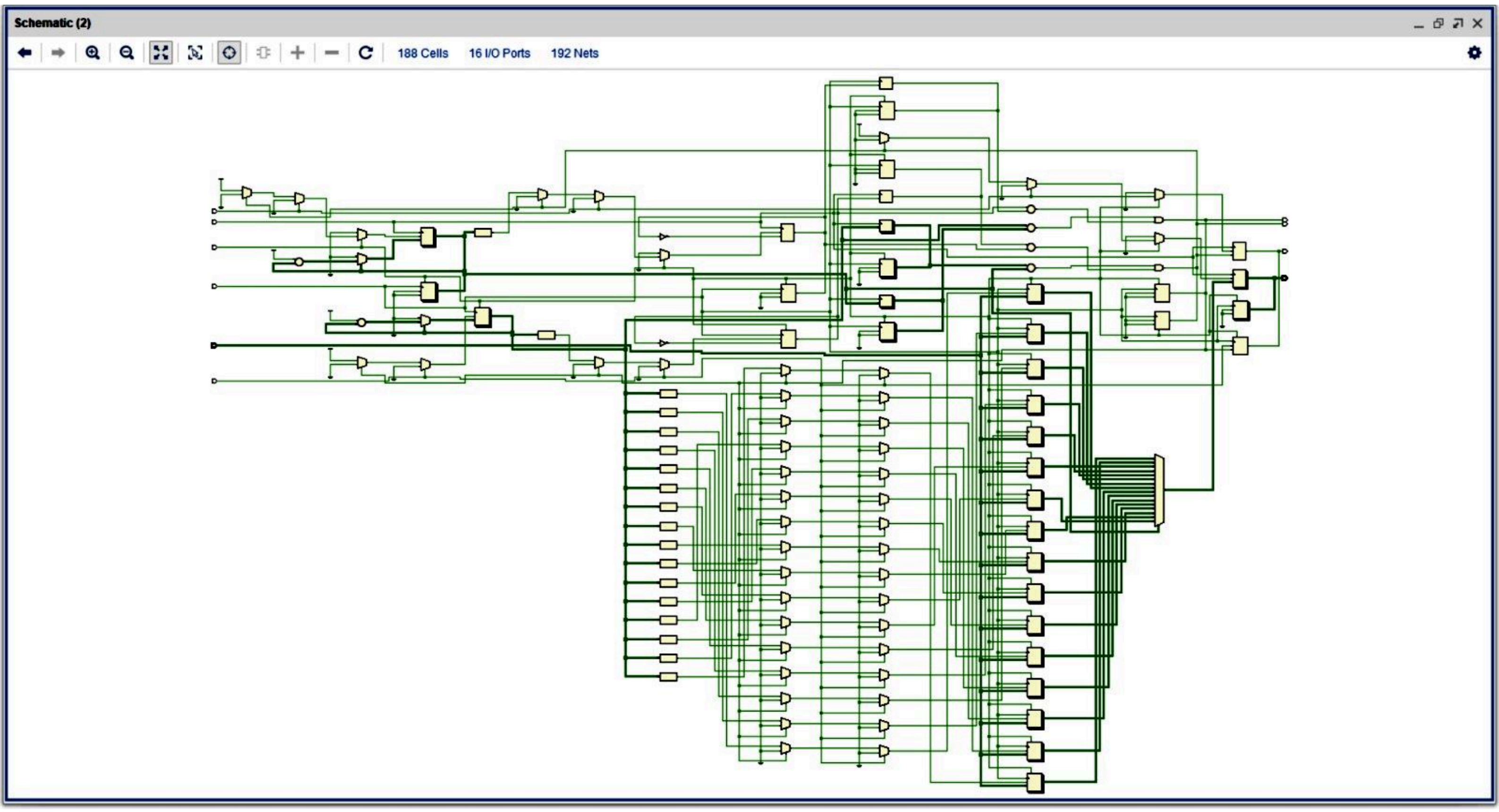
1) FIFO does not have address lines.
2) FIFO is used for Synchronization purpose. i.e when two peripherals are working in different clock frequency then we will go for FIFO.

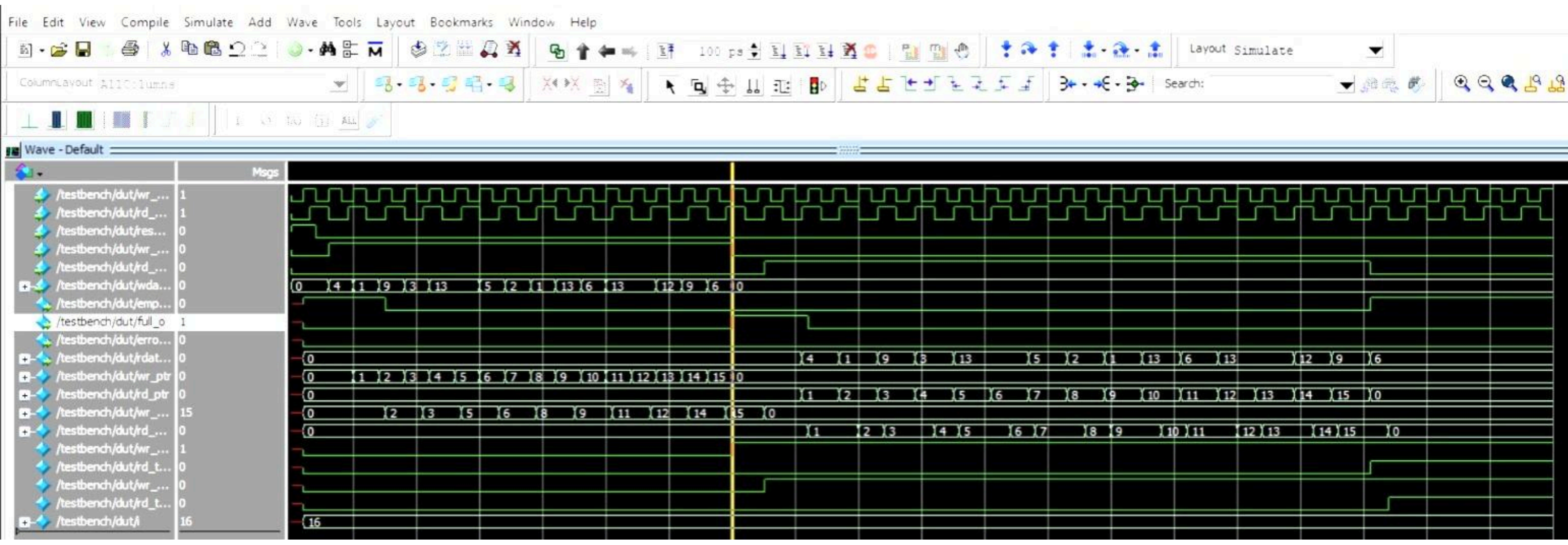
@ difference Between Synchronous FIFO and Asynchronous FIFO :-

- FIFO (First in First out) are Commonly used for Synchronizing across two process and when you have need for temporary storage. One Source read out and other Sources write to the FIFO.

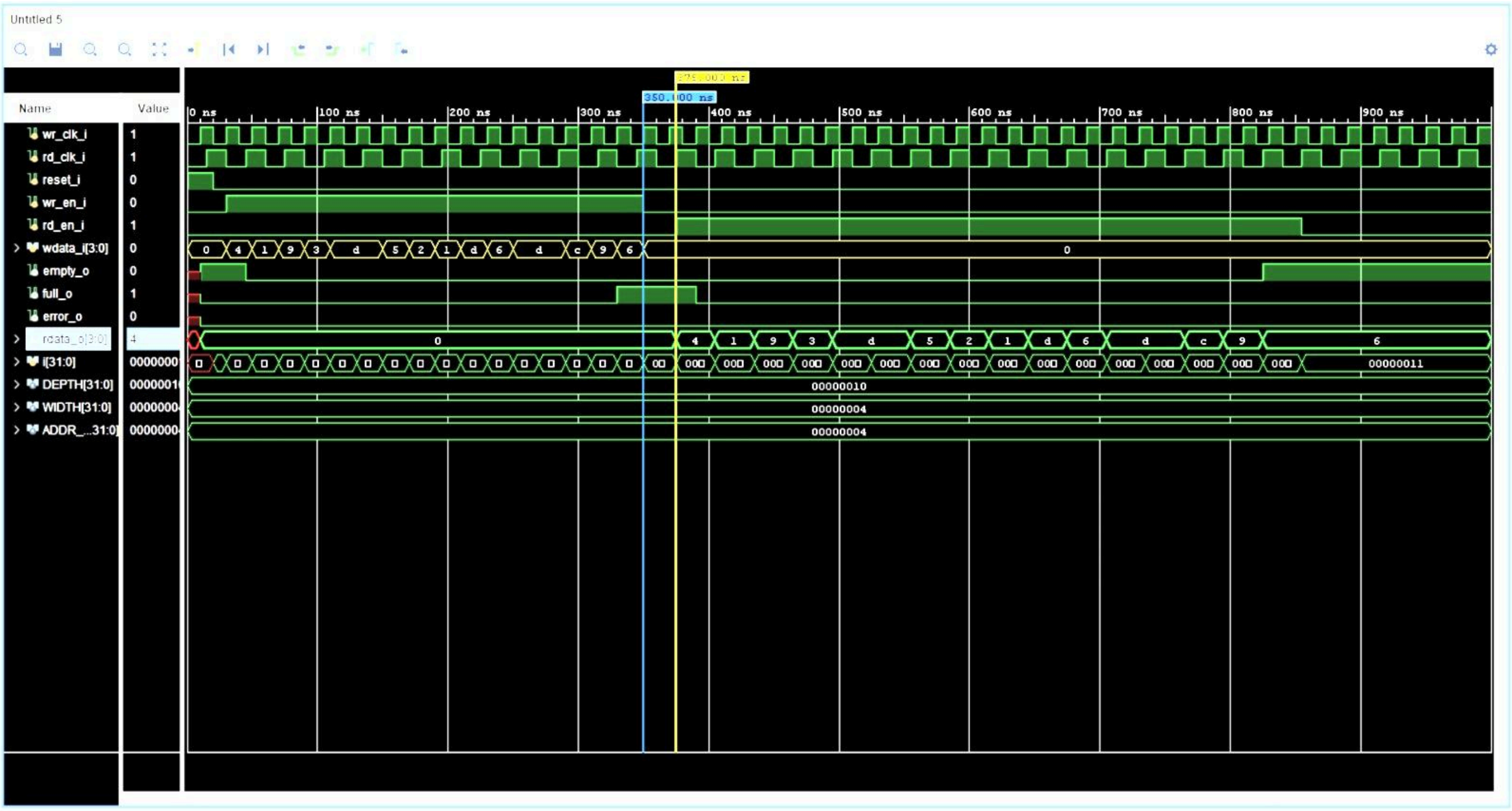
- in a Synchronous FIFO, both the read and write Signal have have Same clock and hence this is used in application that needs temporary Storage while processing, all in a Single clock frequency.

- in a Asynchronous FIFO, the read and write clock are different which means the write to the FIFO happens in clock domain and the read happens in different clock domain.


Schematic (2)
188 Cells
16 I/O Ports
192 Nets


File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
100 ps
Layout Simulate
ColumnLayout AllColumns
Search:
Wave - Default
Msgs

```
/testbench/dut/wr_...     1
/testbench/dut/rd_...     1
/testbench/dut/res...     0
/testbench/dut/wr_...     0
/testbench/dut/rd_...     0
/testbench/dut/wda...     0    0 4 1 9 3 13 5 2 1 13 6 13 12 9 6 0
/testbench/dut/emp...     0
/testbench/dut/full_o     1
/testbench/dut/erro...    0
/testbench/dut/rdat...    0    0 4 1 9 3 13 5 2 1 13 6 13 12 9 6
/testbench/dut/wr_ptr     0    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
/testbench/dut/rd_ptr     0    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
/testbench/dut/wr_...     15   0 2 3 5 6 8 9 11 12 14 15 0
/testbench/dut/rd_...     0    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
/testbench/dut/wr_...     1
/testbench/dut/rd_t...    0
/testbench/dut/wr_...     0
/testbench/dut/rd_t...    0
/testbench/dut/i          16   16
```


Untitled 5
Name
Value
wr_clk_i 1
rd_clk_i 1
reset_i 0
wr_en_i 0
rd_en_i 1
wdata_i[3:0] 0
empty_o 0
full_o 1
error_o 0
rdata_o[3:0] 4
i[31:0] 0000000
DEPTH[31:0] 0000001
WIDTH[31:0] 0000000
ADDR_...31:0] 0000000
375.000 ns
350.000 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
0 4 1 9 3 d 5 2 1 d 6 d c 9 6 0
0 4 1 9 3 d 5 2 1 d 6 d c 9 6
00000011
00000010
00000004
00000004