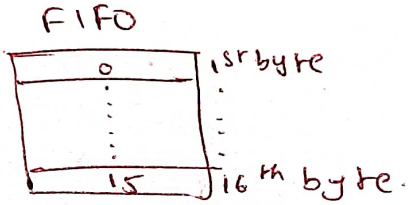
Sahish Reddy V

Asynchronous - FIFO

IIT Bombay

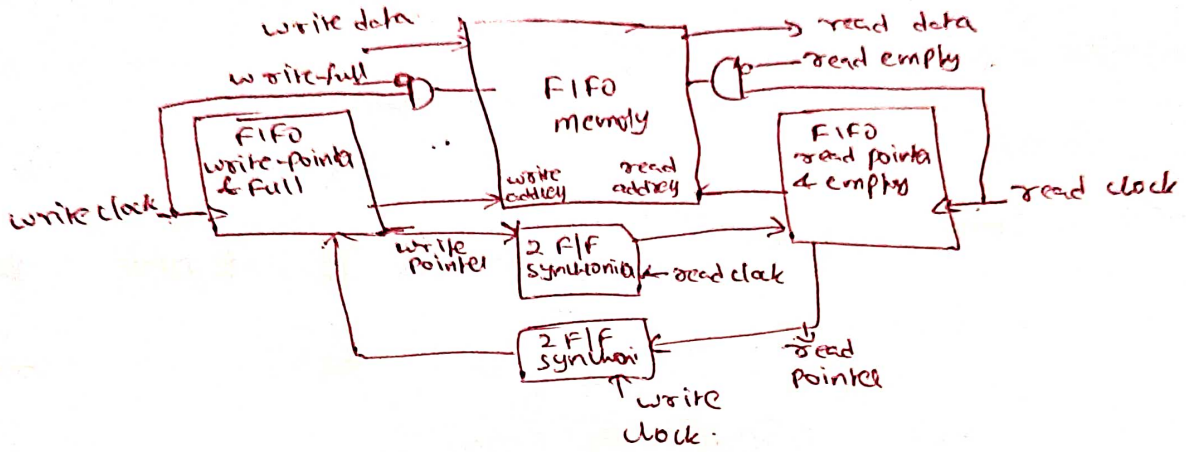
FIFOs are required when a transmitter is transmitting data more fastes than the receiver. If FIFOs are not present, then the data which is transmitted will be lost because the receiver reads at slow rate. so In this project, I took a FIFO of depth 16 and width 8 i.e, FIFO can store 16 bytes of data at 16 different locations.



The implementation of FIFO is not simple as it looks like. Because, the read clock and write clock works at different clock frequencies, the ~~recei~~ transmitter can't keep on writing data if the receiver is slow, because the data will be lost. so we need to have some status signals like Full, Empty etc to indicate whether the FIFO is full or empty. If FIFO is full transmitter can't write data and if the FIFO is empty, receiver can't read data. we need to check the status signals before performing actions

Here, we use 2 synchronizer for read pointer and write pointer both are controlled by write clock and read clock respectively. Because to have the communication between reading and writing data. Pointer is used to hold the next address location.

Block diagram of FIFO

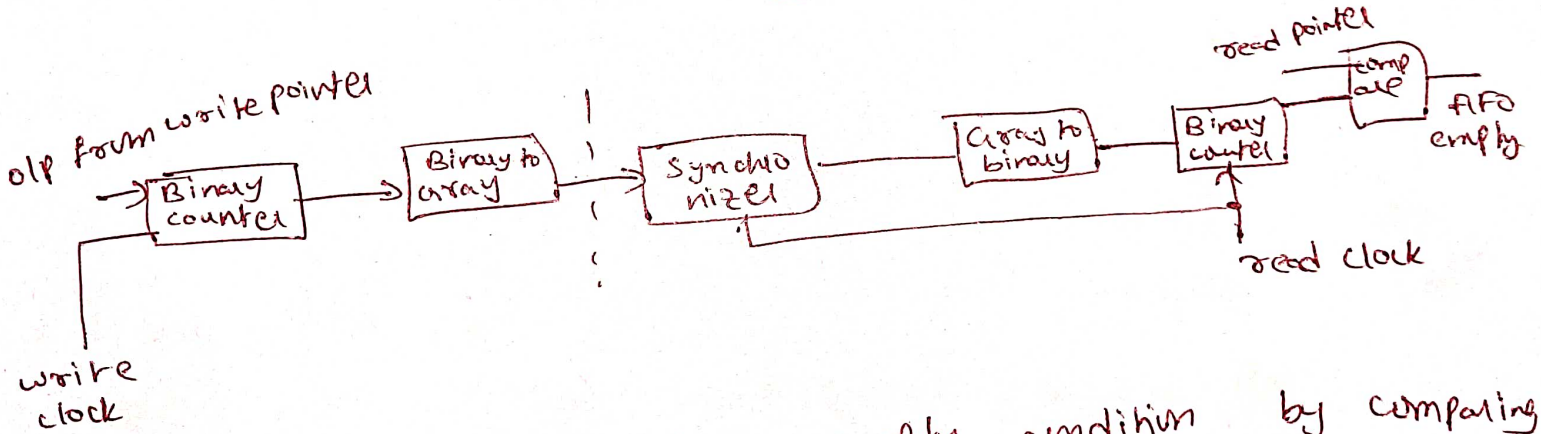


In the above module I have assumed that the write data is coming from a RAM which is 256 size. The 2 Flip Flop synchronizer are for communication between read pointer and write pointer. Because if read pointer = write pointer the FIFO may be full or empty depending on the extra bit we append to read or write pointer.

Actually depth of FIFO is 16. So 4 bits are sufficient for read and write pointers. But to differentiate between full and empty, we add one more bit to MSB. ~~So if it is 0 it is empty, if it is~~

~~FIFO is full~~. so write hay to stop writing when full = 1. likewise read hay to stop reading whem empty = 1. But, this caye empty=1 is nor going to happen in this project, becauye we ae writing faytea than reading. so upon writing in the rb blocky of FIFO full becumey '1' and continua-usly toggles for one clock cycle of read clock because when full = 1 writing shops but when reading is done one block of FIFO is empty and writing can be dune. so full becomes 0. we ue gray code instead of binay for communicchion between the write pointea and read pointea becauye binay is prune to errors mostly.

communicchion unit



we ae genecting empty condition by comparing the preyent read pointea and the write pointea obtained from 2 stage synchonizea. if both ae eyual then FIFO is empty.

Similay way, we generate full condition.

condition for full and empty:-

write pointer = 5 bit = $w_4 w_3 w_2 w_1 w_0$

read pointer = 5 bit = $r_4 r_3 r_2 r_1 r_0$

empty condition :- $w_4 w_3 w_2 w_1 w_0 = r_4 r_3 r_2 r_1 r_0 \Rightarrow$ empty = 1

else empty = 0.

write pointer[4 downto 0] = read pointer[4 downto 0]

than means they should be equal.

Full condition:-

Here write and read pointers should be at the same position, but the write pointer msB bit should be different from read pointer msB indicating that write may come again to that particular location to write before the reading happens.

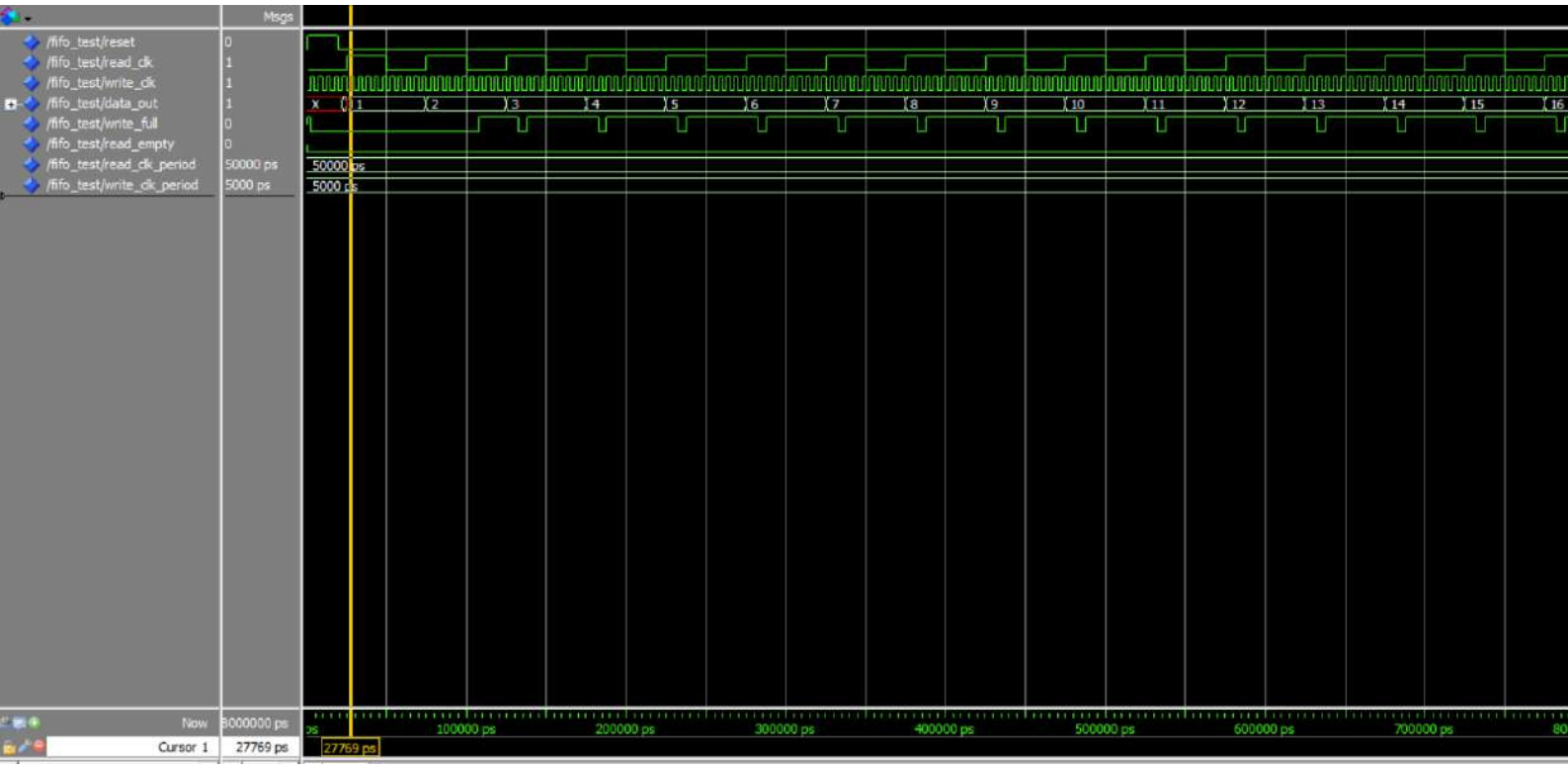
so

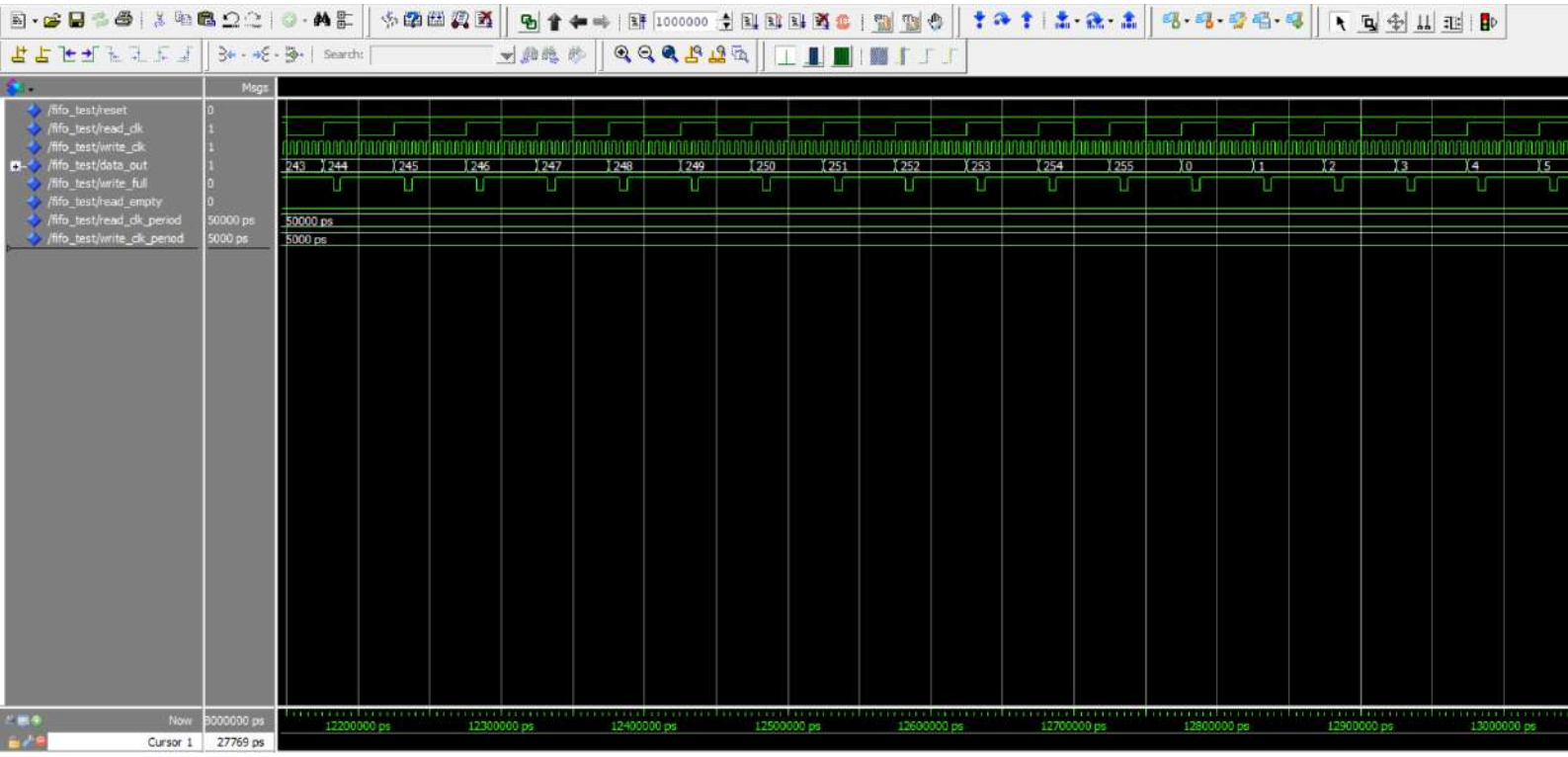
Full $\Rightarrow \overline{w_4} w_3 w_2 w_1 w_0 = r_4 r_3 r_2 r_1 r_0 \Rightarrow$ Full = '1'.

else Full = '0'

Full condition generation is done by comparing present write pointer and read pointer obtained from synchronizer.

Empty condition generation is done by comparing present read pointer and write pointer obtained from other synchronizer.


Msgs
/fifo_test/reset
/fifo_test/read_clk
/fifo_test/write_clk
/fifo_test/data_out
/fifo_test/write_full
/fifo_test/read_empty
/fifo_test/read_clk_period
/fifo_test/write_clk_period
0
1
1
1
0
0
50000 ps
5000 ps
X
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
50000 ps
5000 ps
Now
8000000 ps
Cursor 1
27769 ps
27769 ps
100000 ps
200000 ps
300000 ps
400000 ps
500000 ps
600000 ps
700000 ps


Msgs
/fifo_test/reset
/fifo_test/read_clk
/fifo_test/write_clk
/fifo_test/data_out
/fifo_test/write_full
/fifo_test/read_empty
/fifo_test/read_clk_period
/fifo_test/write_clk_period
0
1
1
1
0
0
50000 ps
5000 ps
243
244
245
246
247
248
249
250
251
252
253
254
255
0
1
2
3
4
5
50000 ps
5000 ps
Now
Cursor 1
27769 ps
12200000 ps
12300000 ps
12400000 ps
12500000 ps
12600000 ps
12700000 ps
12800000 ps
12900000 ps
13000000 ps