

Objective: To design a Asynchronous FIFO to avoid the issues with CDC metastability.

In Asynchronous FIFO data is written in FIFO in one clock domain(wr_clk) & the data is read from the same clock domain in another clock domain(rd_clk). Both the clock is asynchronous.

Modules in Asynchronous FIFO:

- 1) **Synchronizer Module:** This module is used to synchronize the read pointer in write clock domain to check the FIFO full condition & to synchronize the write pointer in read clock domain to check the FIFO empty condition. This module is used to avoid the metastability condition in the design.
- 2) FIFO Empty block: This module takes synchronized write pointer from write clock domain and read pointer of read clock domain to produce the FIFO empty condition. This module operates at read clock.
- 3) FIFO Full block: This module takes synchronized read pointer from read clock domain and write pointer of write clock domain to produce the FIFO Full condition. This module operates at writeclock.
- 4) FIFO Memory Block: This is dual port memory block where data is written into the memory and data is read from memory.

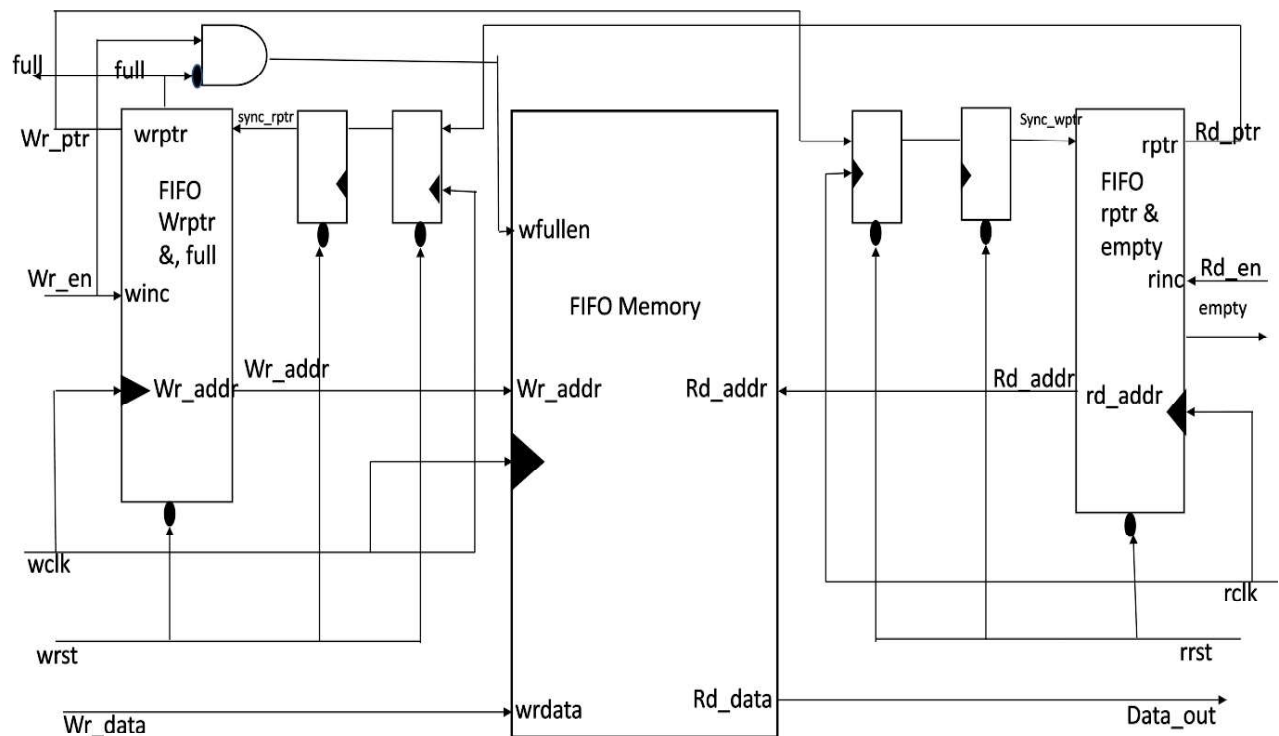


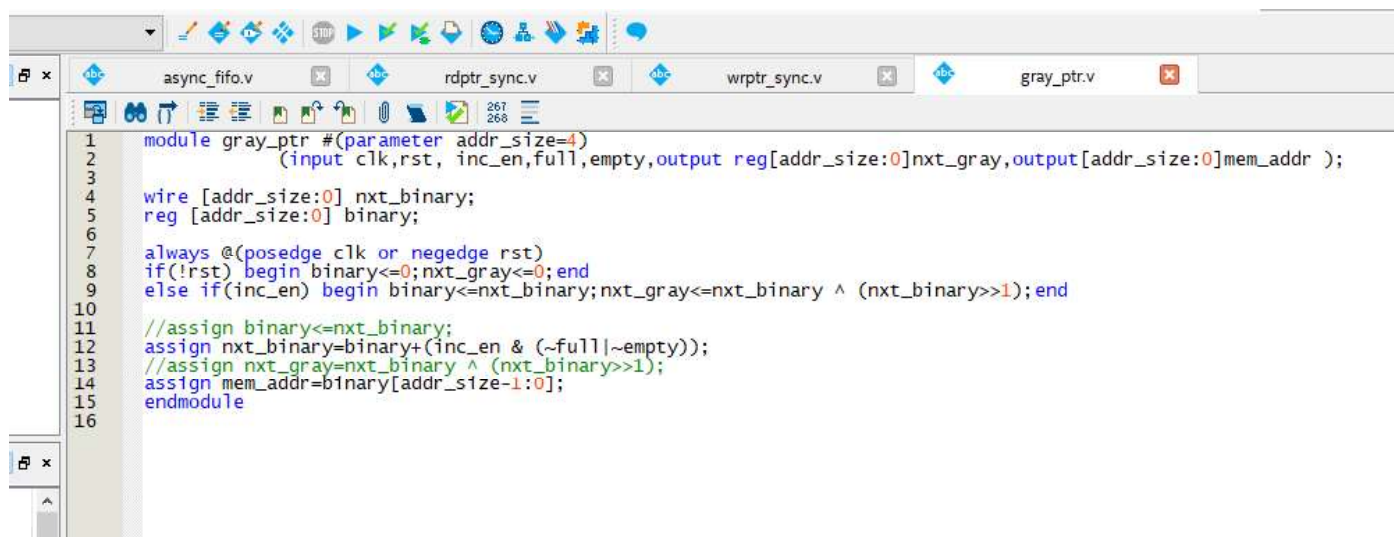
Fig: Asynchronous FIFO

Use of Gray counter: When we are passing the binary values through the synchronizer, due to changing multiple bits there is high chance of metastability. In Gray counter only 1-bit changes so there is less chance of metastability to occur.

For example, there is 2 bits change in Binary 5 (0101) to binary 6(0110). So due to metastability output can be 0111/0110/0101.

Use of n-bit read/write pointer in (n-1) bit memory address:

When the FIFO is empty then write address and read address is at same location. Similarly, when FIFO is full then our address is wrapped one time but the read and write address is again same. So, to differentiate it we have used one more extra bit which will indicate one time wrap of FIFO.



```
1 module gray_ptr #(parameter addr_size=4)
2     (input clk,rst, inc_en,full,empty,output reg[addr_size:0]nxt_gray,output[addr_size:0]mem_addr );
3
4     wire [addr_size:0] nxt_binary;
5     reg [addr_size:0] binary;
6
7     always @(posedge clk or negedge rst)
8     if(!rst) begin binary<=0;nxt_gray<=0;end
9     else if(inc_en) begin binary<=nxt_binary;nxt_gray<=nxt_binary ^ (nxt_binary>>1);end
10
11     //assign binary<=nxt_binary;
12     assign nxt_binary=binary+(inc_en & (~full|~empty));
13     //assign nxt_gray=nxt_binary ^ (nxt_binary>>1);
14     assign mem_addr=binary[addr_size-1:0];
15 endmodule
16
```

In this module gray and binary values are incremented only when FIFO is not full while writing in memory or FIFO is not empty while reading from memory.

Both read and write pointer will use this module for generation of its next pointer value.

Condition to check full and empty FIFO:

- If synchronized write pointer is equal to read pointer then empty status flag will be high.
- To check full condition:
 - i) The write pointer and the synchronized read pointer MSB's are not equal. (Because the write pointer must have wrapped one more time than the read pointer)

ii) The write pointer and the synchronized read pointer 2nd MSB's are not equal (because an inverted 2nd MSB from one pointer must be tested against the un-inverted 2nd MSB from the other pointer, which is required if the MSB's are also inverses of each other

iii) All other write pointer and synchronized read pointer bits must be equal.

2- Flip Flop Synchronizer: To avoid the issue for CDC metastability, we have implemented a 2FF synchronizer. When the data is changing either 0 to 1 or 1 to 0 at rising edge of clock then after one clock it will get stable at the output of first FF. In the next clock we achieve correct result at output of second FF.

This module is required for sending read pointer to the write clock domain and write pointer to read clock domain.

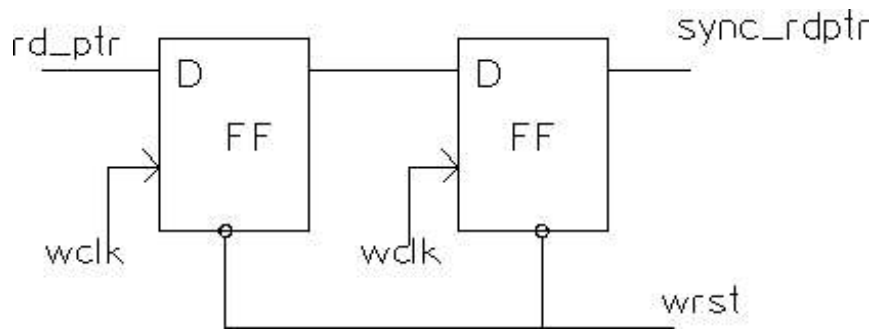


Fig: 2 - FF synchronizer for read pointer

We check FIFO Full condition on write clock domain. So, this module work on rising edge of write clock. To check this condition, wptr and sync_rdptr is compared on wclk.

```

1 module rdptr_sync #(parameter addr_size=4)(input wclk,wrst,input[addr_size:0]rd_ptr,output reg[addr_size:0]sync_rdptr)
2
3   reg [addr_size:0]temp;
4   always@(posedge wclk or negedge wrst)//write pointer work on read clock
5   begin
6     if(!wrst)begin sync_rdptr<=0;
7       temp<=0; end
8   else begin
9     temp<=rd_ptr;
10    sync_rdptr<=temp;
11  end
12 end
13 endmodule
14

```

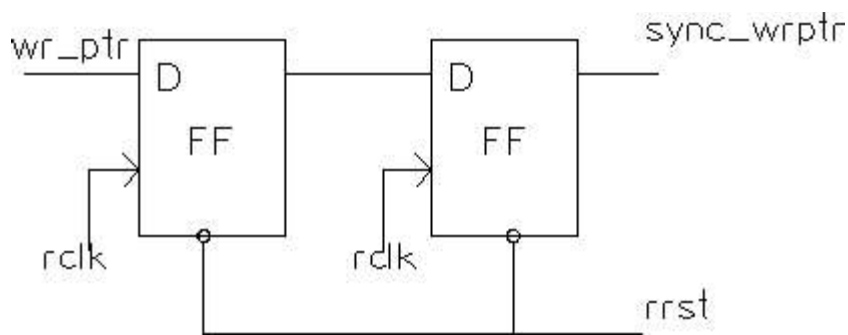


Fig: 2-FF synchronizer for write pointer

We check FIFO Empty condition on read clock domain. So, this module work on rising edge of read clock. To check this condition, rdptr and sync_rdptr is compared on wclk.

```

1 module wrptr_sync #(parameter addr_size=4)(input rclk,rrst,input[addr_size:0]wr_ptr,output reg[addr_size:0]sync_wptr);
2
3   reg [addr_size:0]temp;
4   always@(posedge rclk or negedge rrst)//write pointer work on read clock
5   begin
6     if(!rrst)begin sync_wptr<=0;
7       temp<=0; end
8   else begin
9     temp<=wr_ptr;
10    sync_wptr<=temp;
11  end
12 end
13 endmodule
14

```

Results & Simulation:

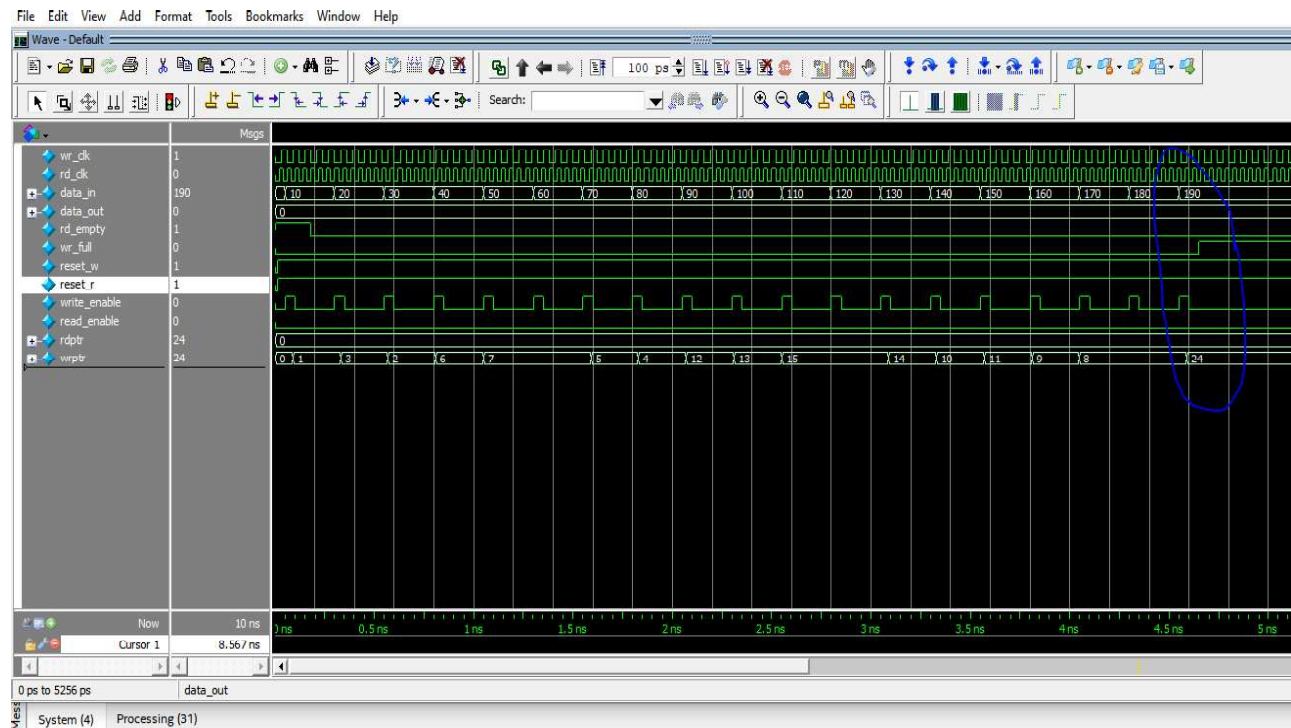


Fig: writing in fifo and fifo full

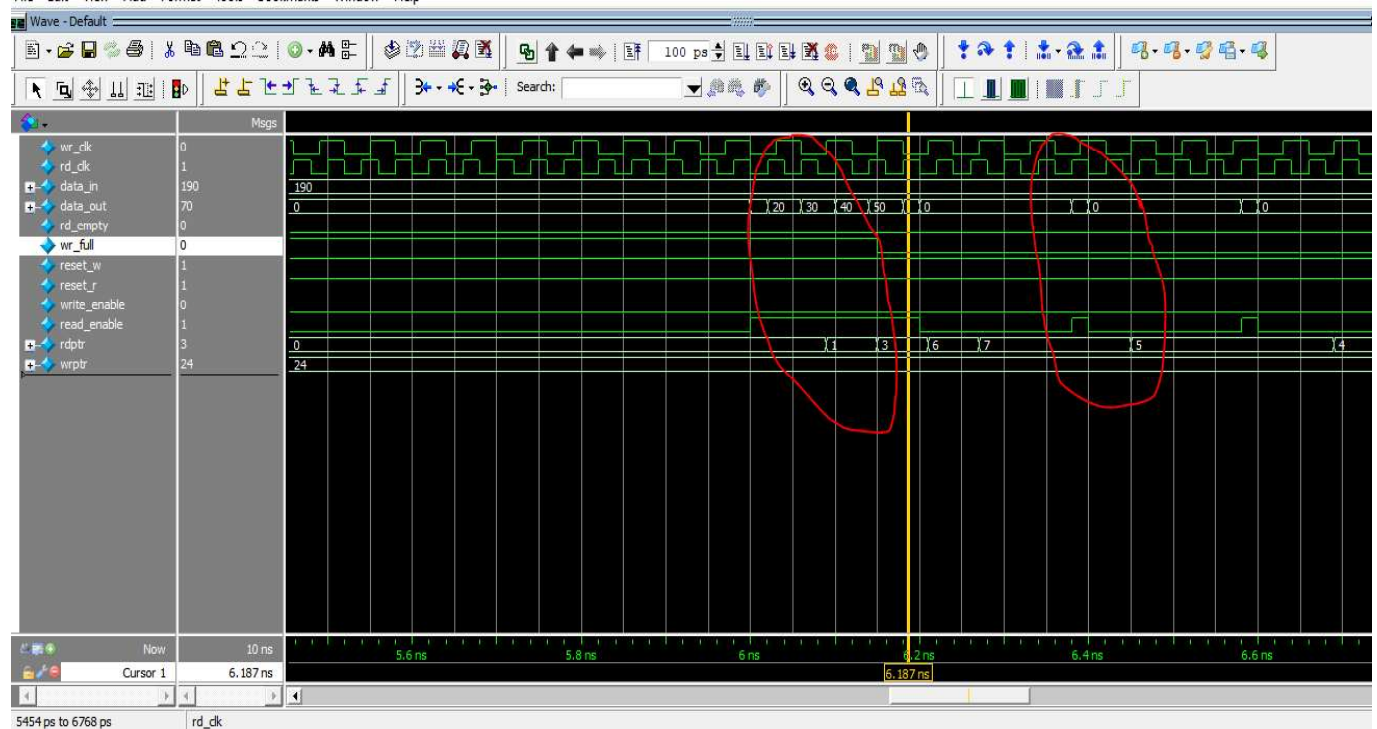


Fig: Read fifo when read enable high

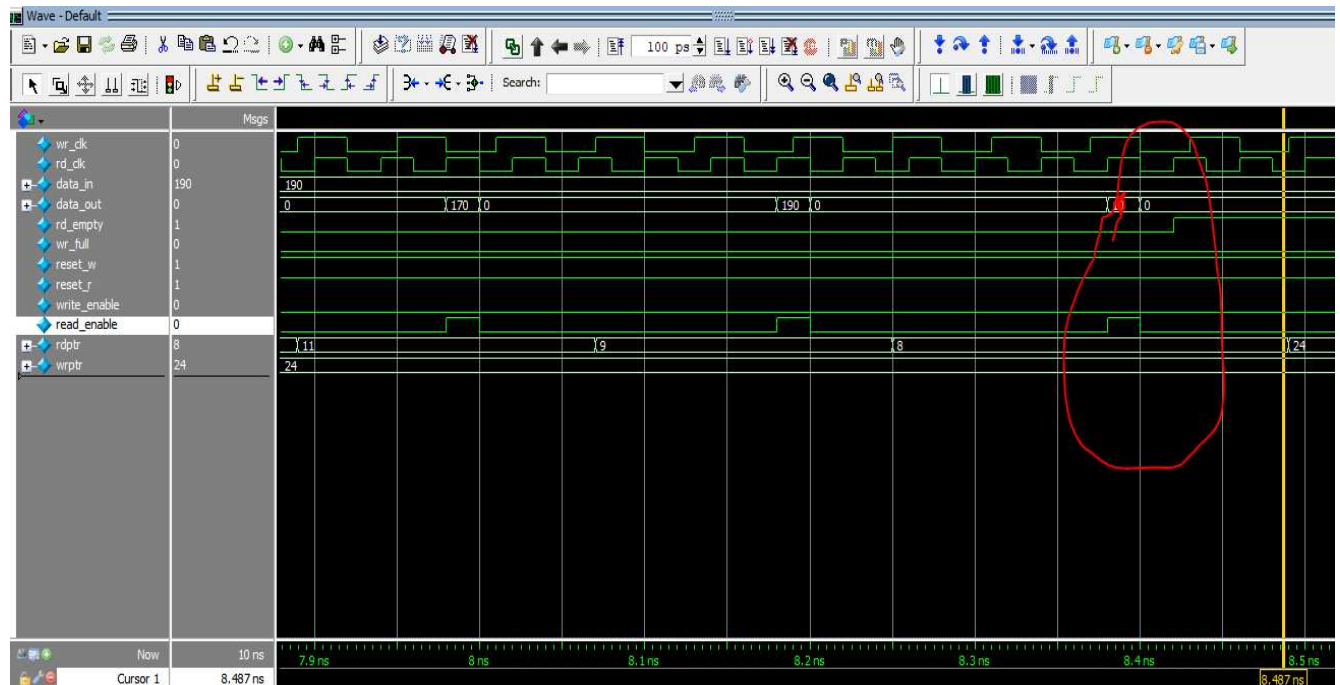


Fig: fifo empty signal high

Data is read only when fifo read signal is high. There is two clock delay due to 2-ff synchronizer for producing empty and full condition,