ASYNCHRONOUS FIFO DESIGN
(Dual Clock FIFO)

Gurram Manikanta (213070098)

July 30, 2022
Department of Electrical Engineering
IIT-Bombay

Contents

Objective:

Introduction:

Asynchronous FIFO : FIFOs are often used to safely pass data from one clock domain to another clock domain. An asynchronous FIFO refers to a FIFO design where data is written to a FIFO buffer from one clock domain and the same data is read from the FIFO buffer from another clock domain, where the two clock domains are asynchronous to each other.

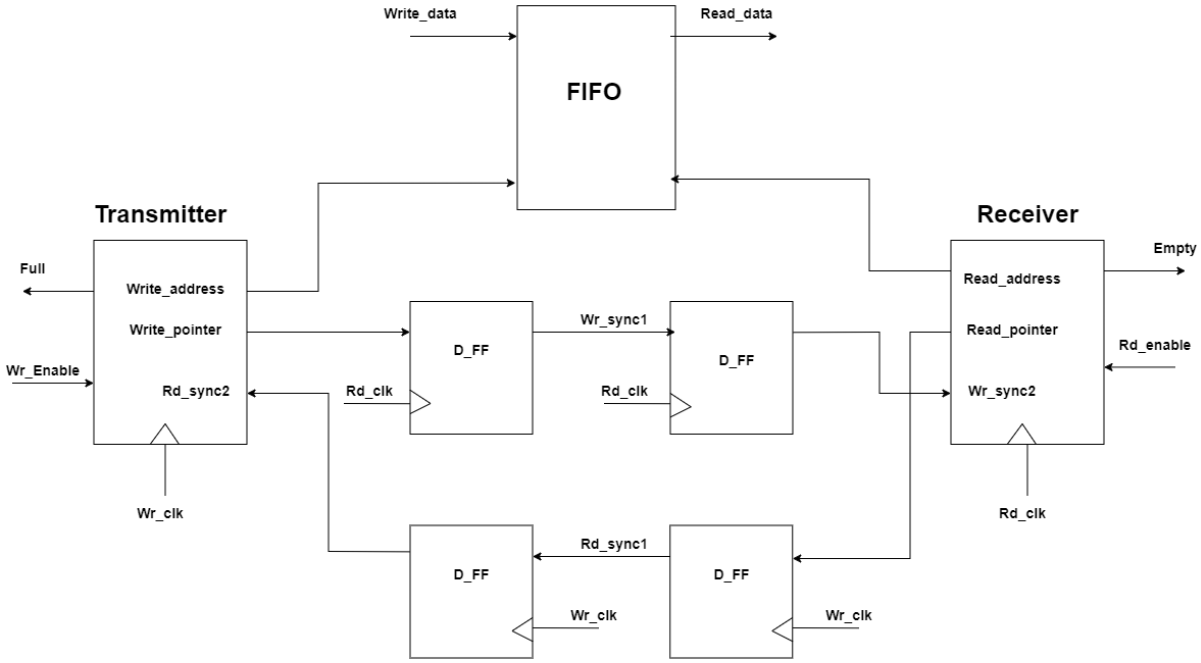


Fig_1: Asynchronous FIFO design

Blocks in FIFO design:

Read clock domain to write clock domain Synchronizer:

This module contains flip-flops that are synchronized to the write clock. Synchronize the write pointer into the read clock domain. This is because the write pointer is having metastability due to setup and hold time in the read clock. So, there is a synchronizer for this problem. The synchronized read pointer will be used by the Write control module to generate the **Write_Full** condition.

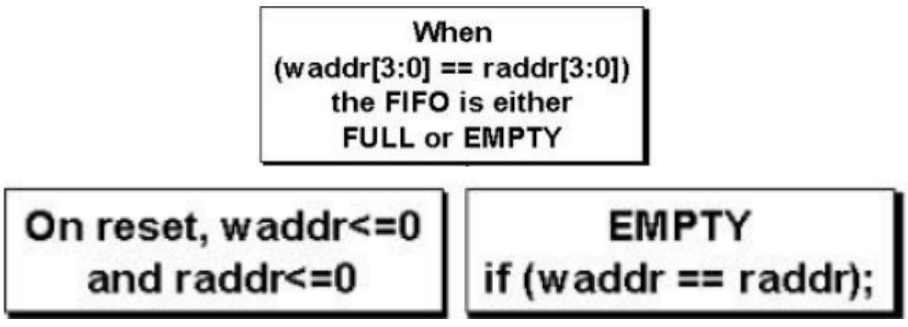
Write clock domain to read clock domain Synchronizer:

This module also contains flip-flops that are synchronized to the read clock , Similarly we will use one synchronizer to synchronize read pointer into the write clock domain, because read pointer, due to setup and hold time violation in the write clock will give the metastability. Further, this synchronized write pointer will be used to generate the **Read_empty** condition.

Logic to generate Read_empty signal:

This module is completely synchronous to the read-clock domain and contains the FIFO read pointer and empty-flag logic.

In our case we have used FIFO of depth 16 and address is of 4 bits, let us suppose that we have written some data at these 16 places. Now, write pointer will wrap up, again comes to 0th location and will start writing values. So, as to get to know whether the FIFO is full or empty, we will add one extra bit such that when FIFO is empty both the pointers will point to same location



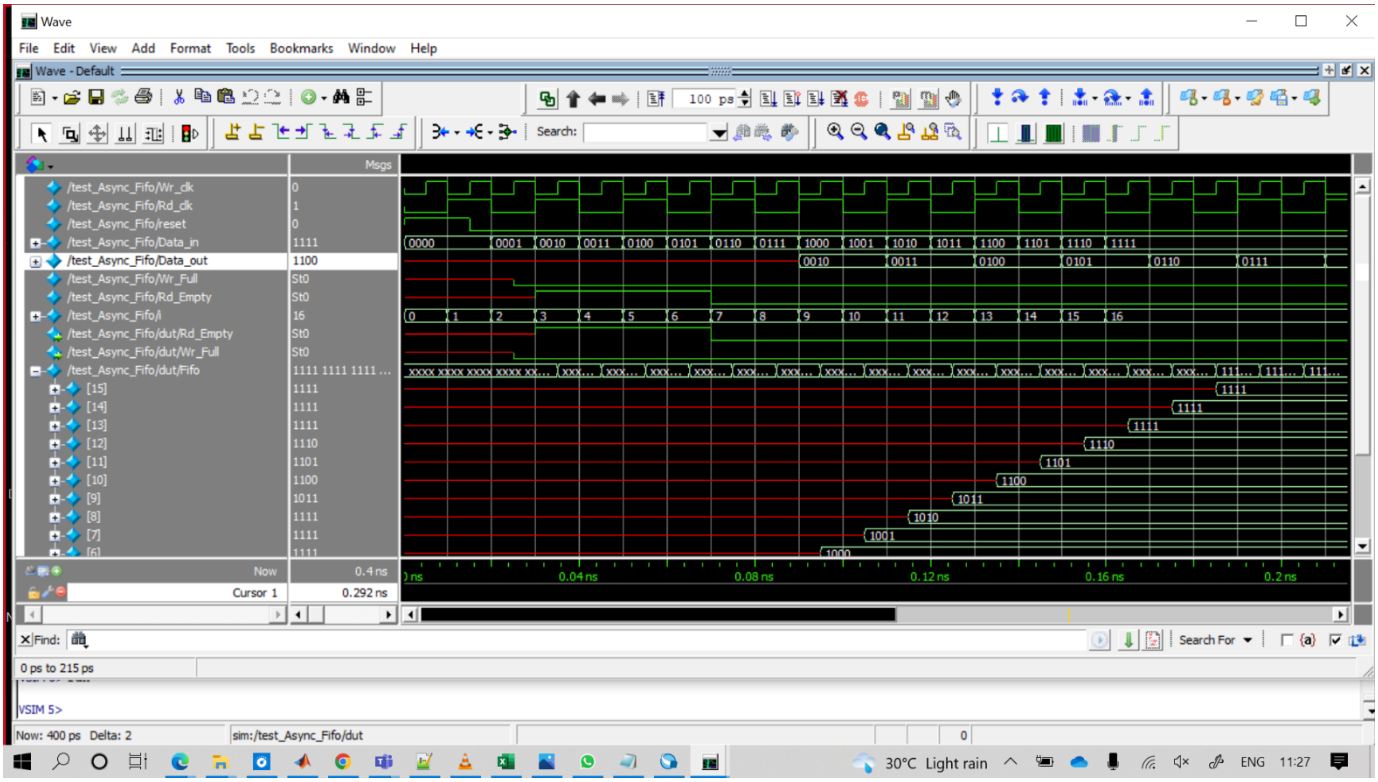
Logic to generate Write_Full signal:

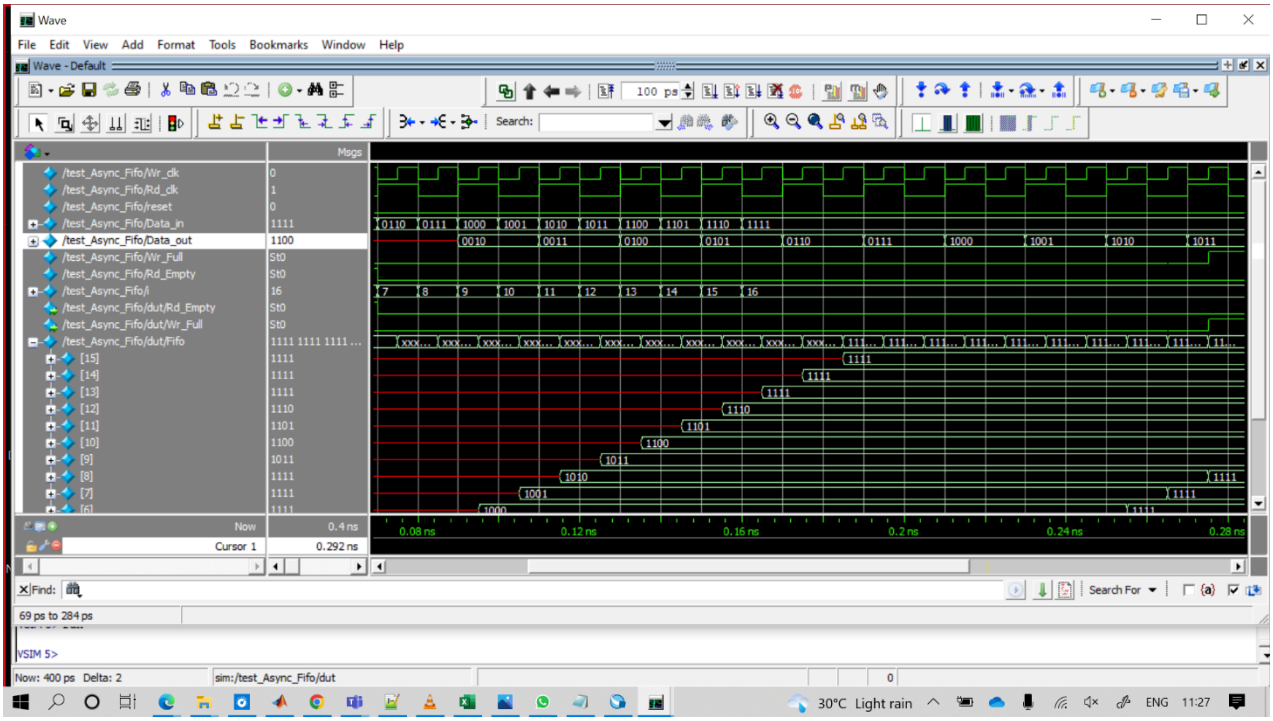
This module is completely synchronous to the write-clock domain and contains the FIFO write pointer and full-flag logic.

When the complement of MSB of write pointer equals the MSB of read pointer, and rest all the bits are same, then we can say that write pointer has traversed the FIFO once, i.e. it has already wrapped up and hence can generate the FULL condition logic.

```
FULL
if ({~waddr[4],waddr[3:0]} == raddr);
```

Simulation Results:





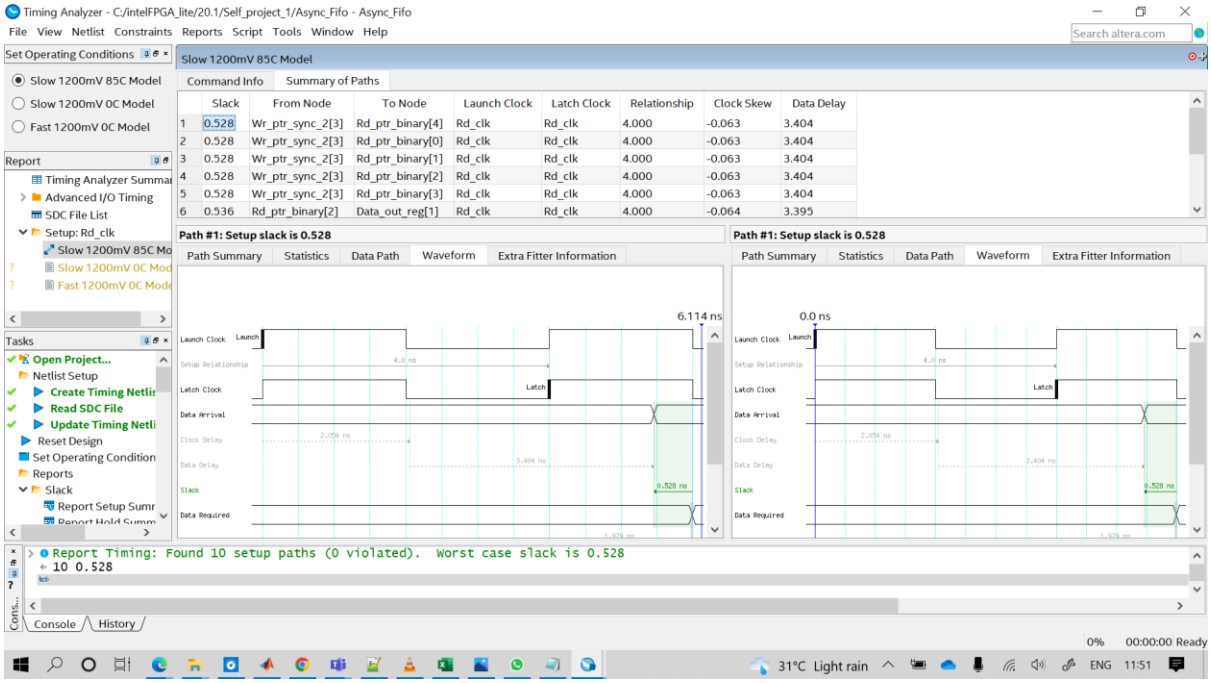
- Verilog code is written for Asynchronous FIFO and Testbench in Quartus Prime.
- Simulations are done in Modelsim using the written testbench.

Static Time Analysis:

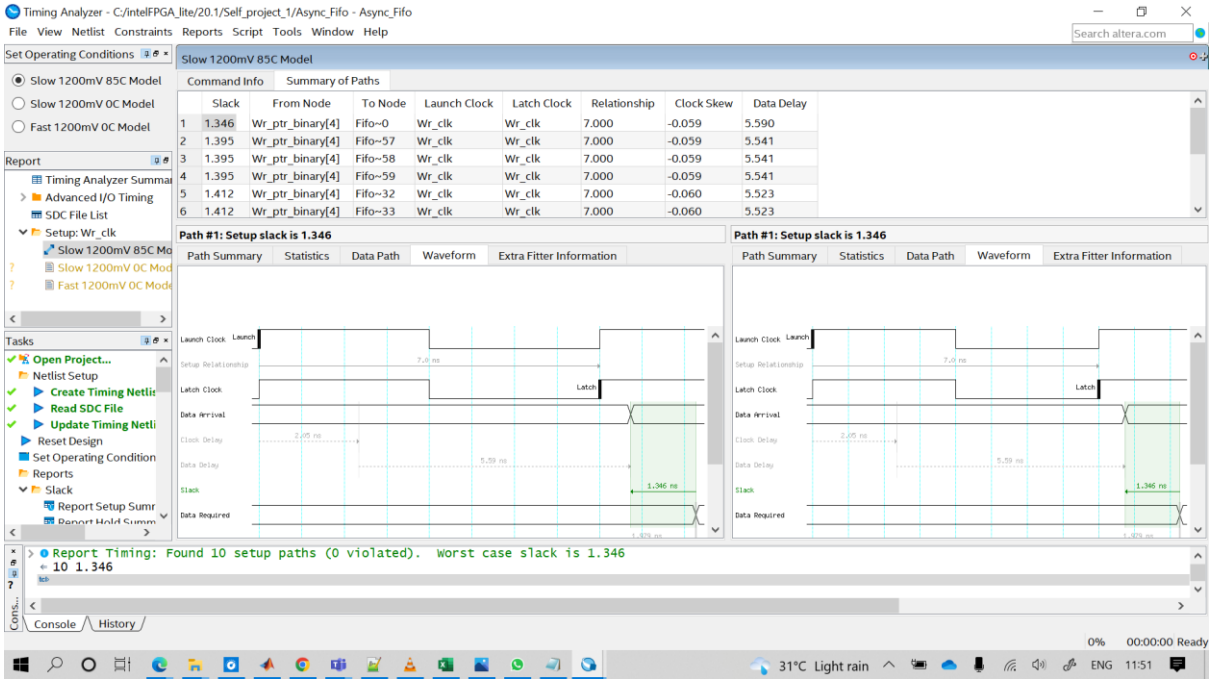
This is the process of analyzing delays in a logic circuit to determine the condions under which the circuit operates reliably.

For each path in the circuit, slack value represents the difference between the clock period constraint and the path delay. A positive slack means that delay is smaller than the constraint (i.e. Time required > Time of arrival). A negative slack represents a delay that is larger than constraint (i.e. Time required < Time of arrival).

For Read_clock:



For Write_clock:

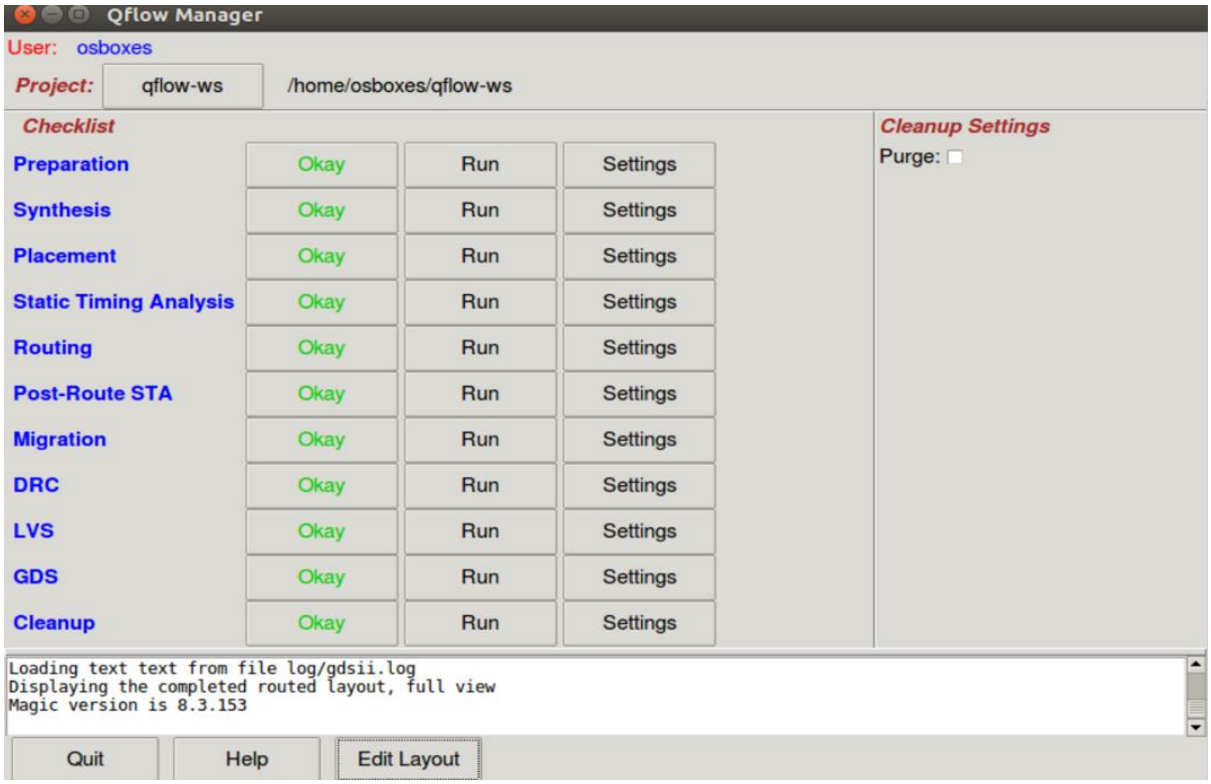


Here after giving Read clock time as 4ns, Write clock time as 7ns. I got a positive slack for both clocks.

Post Synthesis check and layout generation:

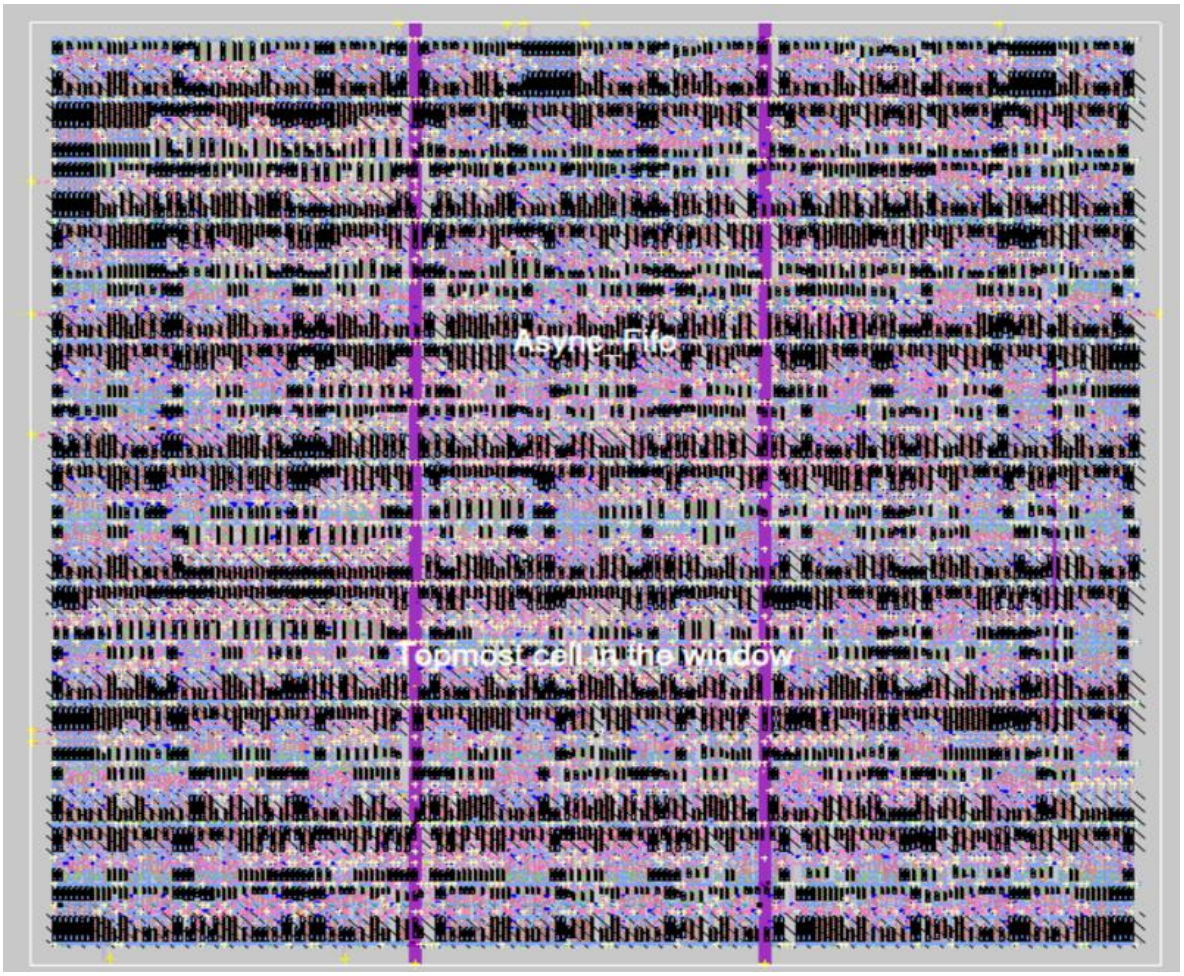
Gate level netlist is generated (using standard cells of osu035_stdcells) a er synthesis using QFLOW.

Equivalent layout is generated and is viewed in .gds format in MAGIC (technology file used is SCN4M_SUBM.20 – scalable CMOS N well sub micron technology)

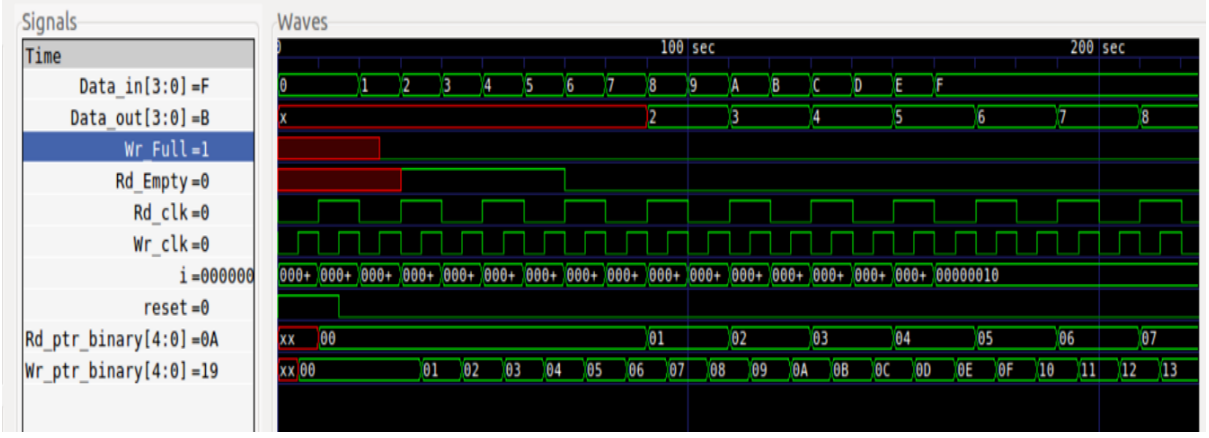


- **Qflow** was performed for the main module which contain all the essential blocks of asynchronous FIFO ,then steps were followed to extract the **GATE LEVEL NETLIST** as well as final layout of the asynchronous FIFO which was seen with the help of **Magic**.The netlist generated from the qflow was again simulated with the help of a test bench simulated using **iverilog** such that the delays are observed by comparing them with the testbench simulated using **Modelsim** from RTL simulation.

Layout view in magic:



Post-synthesis Gate level simulation :



We can see that data is read after a few ns delay. Hence the design yields fruitful results.