Design of Asynchronous FIFO

KUNAL KAUSHIK – 21307R020

July 25, 2023

Department of Electrical Engineering

IIT-Bombay

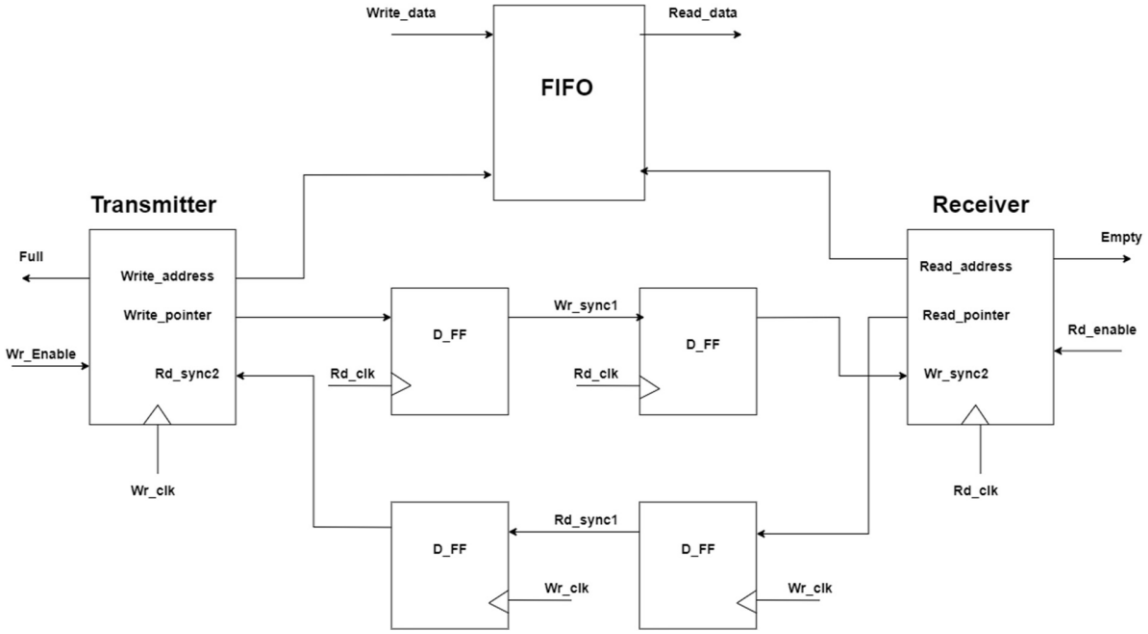
Contents

1. Introduction

Asynchronous FIFO is used whenever we want to transfer multiple data signals between two different clock domains. If there are two different clock domains then reading and writing may occur with different rates. Because of this there might be chances of reading wrong data before it was written completely or writing can happen in wrong locations which leads to data loss or metastability. To avoid such scenarios the reading and writing operations will be done using 2-bit synchronizers.

An Asynchronous FIFO is a First-In-First-Out queue which use *different clock pulses* for read and write so that it can receive and transfer data properly without any loss. The number of rows in FIFO is called as *FIFO Depth* and number of bits per each row is called as *FIFO Width*.

2. Block Diagram of an Asynchronous FIFO



2.1. Basic Working of FIFO

- Here the FIFO is taken as array with *Depth* number of slots with each slot of *Width* number of bits.
- Initially upon reset, both the write and read pointers are set to zero. Upon every write operation, write pointer is incremented and upon every read operation, read pointer is incremented. We shouldn't write to a FIFO when it is full and we shouldn't read from a FIFO if it is empty.

2.2. Full and Empty flags

- These flags will be set if both read and write pointers are pointing to same location.
- *Empty flag represents that the FIFO is empty.* It is controlled by circuit at reading end. It will be set when both read pointer and write pointer (which is send through synchronizer) are same.

- *Full flag represents that the FIFO is full.* It is controlled by circuit at writing end. It will be set when both write pointer and read pointer (which is send through synchronizer) are same.
- Consider a state where initially both the flags are zero. Therefore, we can say that FIFO is empty as both the read pointers and write pointers are same.
- Now consider all the locations except top of FIFO have been written. Now, write pointer points out to the top of FIFO. If the top location is also written, the write pointer wraps up and come to zero. The FIFO is full in this case. But read pointer is already there which implies the FIFO is empty. That is FIFO is full as well as empty, as both the read and write pointers are equal. Now we need to resolve this issue.
- To resolve this, we'll add one extra bit to the pointers which represents the full/empty flag and will differentiates whether the FIFO is full or empty flags.

Example: Consider the FIFO is of 16 X 8. Now we use 5 bits for write pointer as well as read pointer (instead of 4). Now, if all the locations have been written except the location, write pointer will point out to 5'b01111. Now if the top location has also been written, the write pointer will be 5'b10000. So, it can be observed that that the full flag will be set when MSB's of both write and read pointers are compliment of each other and the rest of the bits are equal to each other. Similarly, empty flag will be set when both write and read pointers are exactly the same.

2.3. Synchronizers

- It is difficult to compare read and write pointers directly because they are from different clocks.
- So, synchronizers are used one to synchronize write pointer at receiver side and another is to synchronize read pointer at the transmitter end.

2.4. Gray Conversion

- If we send the pointers in binary format then probability of error will be more as there will be more than one bit changes between the successive locations.
- For this we'll convert the pointers into gray code (because there is only one bit change between successive pointer values) before sending through synchronizer by using *Binary-to-Gray* code converter.
- Then we'll convert it back into binary using *Binary-to-Gray* converter then both the pointers will be compared.

2.5. Write and Read Pointers

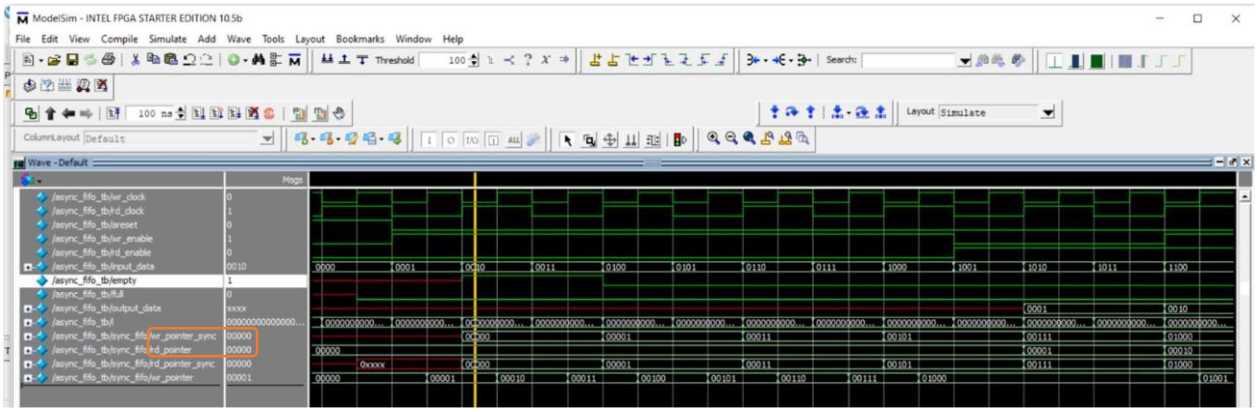
- *Write pointer* represents the *write address* in the FIFO where the *input data* should be written. After every write operation, write pointer will get incremented which represents the next location in FIFO where next write should happen.
- *Read pointer* represents the *read address* in the FIFO where the *output data* should be read. After every read operation, read pointer will get incremented which represents the next location in FIFO where next read should be done.

2.6. Write and Read Operations

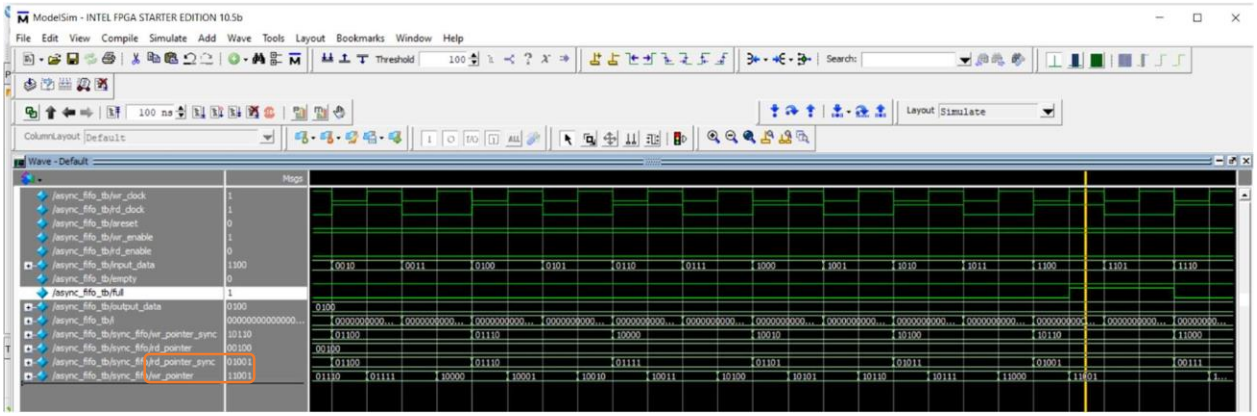
- A write operation should happen only if *write enable* is high and FIFO is *not full*. Data that should be written will be the input to FIFO and will be written at the corresponding write pointer location.
- A read operation should happen only if *read enable* is high and FIFO is *not empty*. Data that should be read will be the output from FIFO and will be read from the corresponding read pointer location.

3. Simulation Outputs

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



- ✓ Here we can see that at the beginning when there is no write happened, empty flag is set.
- ✓ We can check the condition also that the read pointer and the write pointer (after coming from the synchronizer) are same (00000).



- ✓ After writing the data into the FIFO we can see that the full flag is set.
- ✓ We can check the condition also that the write pointer (with MSB negated) and the read pointer (after coming from the synchronizer) are same (01001).

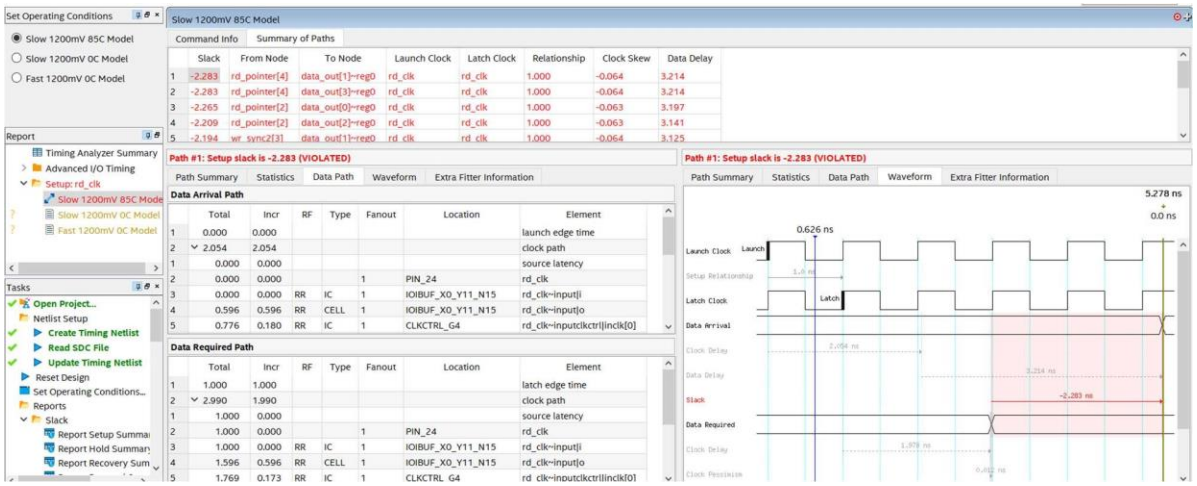
4. Static Timing Analysis

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

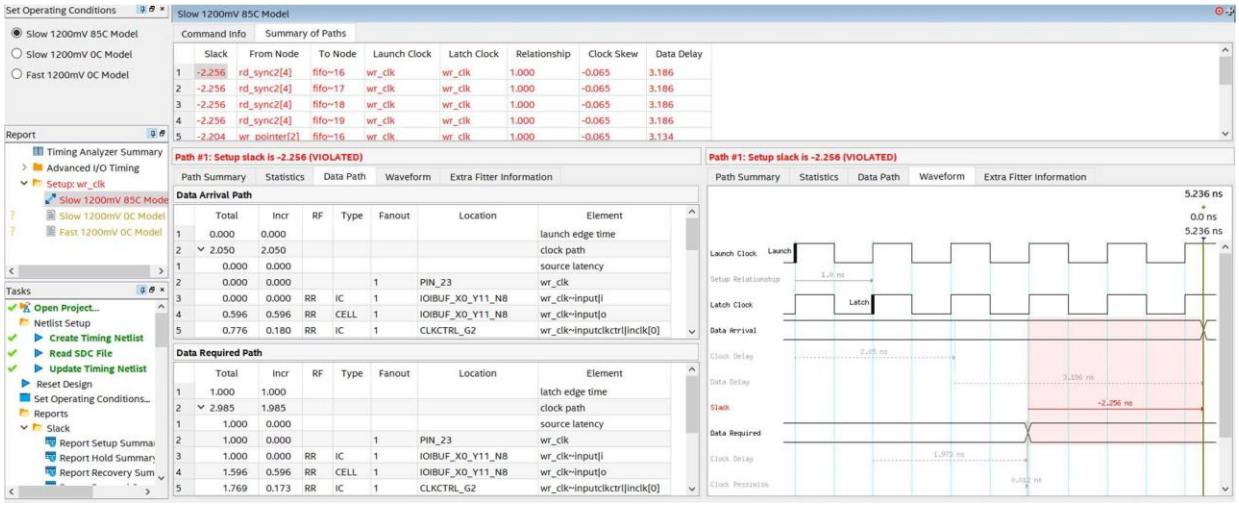
the clock period constraint and the path delay. A positive slack means that delay is smaller than the constraint (i.e. Time required > Time of arrival). A negative slack represents a delay that is larger than constraint (i.e. Time required < Time of arrival).

Default Timing Analysis:

Read Clock:



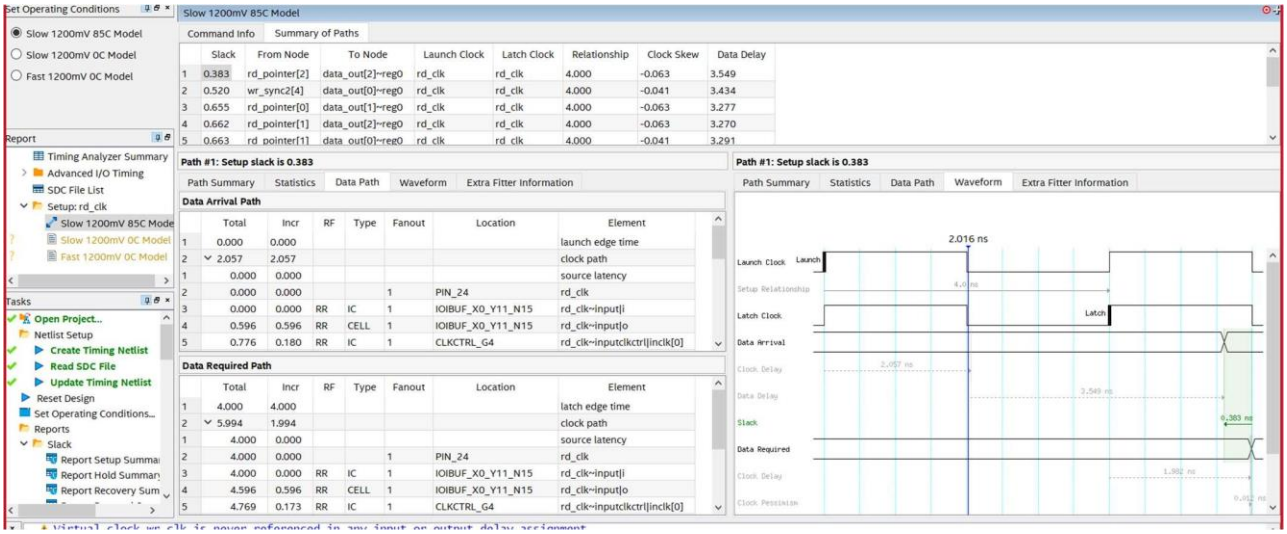
Write Clock:



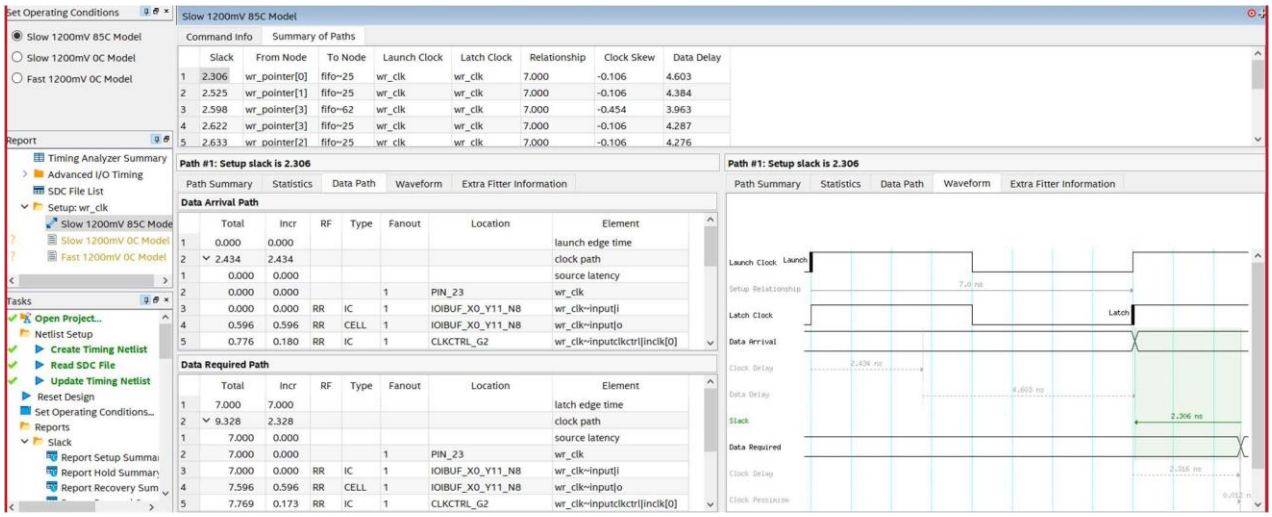
- Here we are getting negative slack so to get positive slack we'll change the clock period of read and write clocks.
-

Regenerated Timing Analysis:

Read Clock: 4ns



Write Clock: 7ns



5. References

- Asynchronous FIFO : – Tutorials in Verilog & SystemVerilog: (systemverilogdesign.com)