Design of Asynchronous FIFO

V Joseph Jeevan reddy-203070049

July 26, 2021

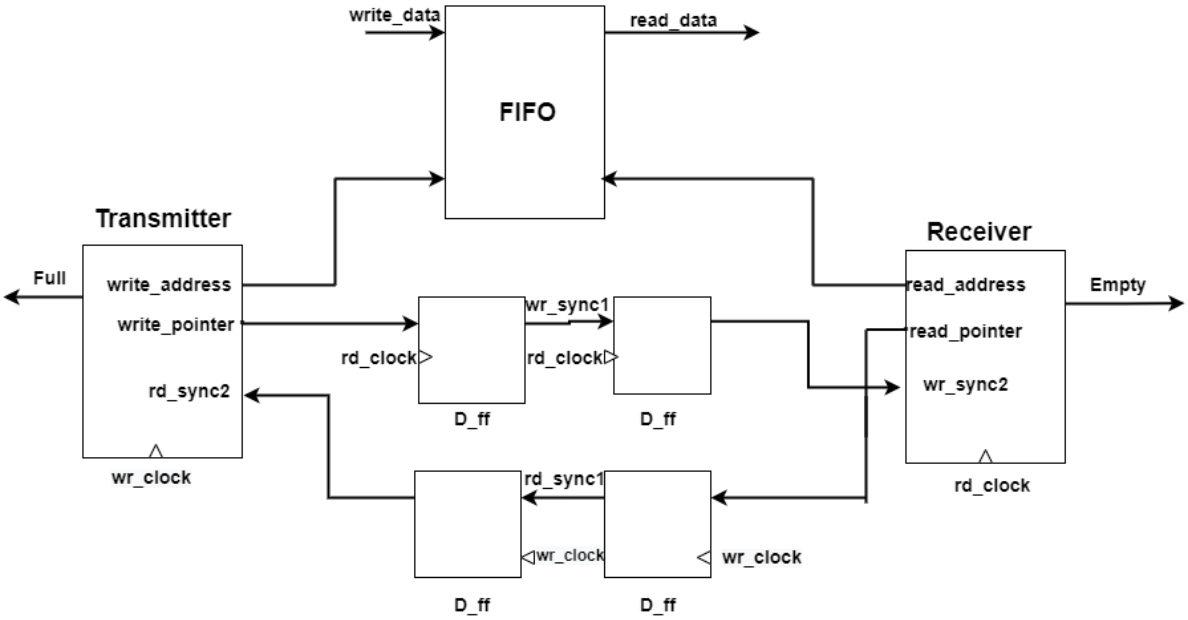
Department of Electrical Engineering

IIT-Bombay

Introduction:

It is often required to send the data between two modules. If the two modules are operating at the same frequency then it is not much of an issue. Then the transfer can be easy with a simple synchronous FIFO. But if both the modules are operating at different frequencies then a simple data transfer is not sufficient. Because the sending data is synchronous with the transmitter clock but it is asynchronous with the receiver clock. So, there may be a probability that the data may change within the period of metastability i.e. it may change within the interval of {Tset,Thold}. Thus, we may receive the wrong data. So, it is very important to design a structure in which we can efficiently communicate between two clock domains. Asynchronous FIFO is one such structure.

Block diagram of implemented asynchronous FIFO:



Consider a FIFO of dimensions n X m (depth X width). It consists of read and write pointer. The transmitter writes the data to FIFO through write pointer and the receiver reads the data from the FIFO through read pointer. Let's consider that the transmitter is operating at a clock of wr_clock and receiver is operating at a clock of rd_clock. The write pointer always points to the next data to be written and the

read pointer points to the data word to be read. The main challenge is the determination of empty and full flags.

Initially upon reset, both the write pointer and read pointer 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. 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. Now, if the top location is also written, the write pointer wraps up and come to zero. The FIFO is full in this case. But if we consider the empty and full flags, they are equal in this case also. So, now in both the cases where FIFO is full as well as empty, both the read and write pointers are equal. Now we need to resolve this issue.

If the FIFO is of 'n' bit depth, we only need log n_2 bits to address the FIFO. But if we use only these number of bits for read and write pointers, the determination of full and empty flags become difficult. So we add one extra bit to resolve the issue generated in determining the full and empty flags. But we only use log n_2 bits to access the FIFO as the depth of FIFO remains the same.

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.

Now, the next challenge lies in the comparison of read and write pointers. They can't be compared directly as they are operating on different clocks. So, they must be passed through a synchroniser.

As, the write pointer is controlled by the transmitter, it should be responsible for generating the full flag. Similarly, read pointer is controlled by the receiver, so the empty flag should be generated by the receiver. So, the read pointer is passed through a two-stage synchroniser and compared with the write pointer for the generation of full flag. The write pointer is passed through a two-stage synchroniser and compared with the read pointer for the generation of empty flag.

Now, the read and write pointers can be send in the binary format. But the problem is there are many bits change in the successive transmission. Suppose if the write pointer is 3'b101, if the data is written to that location, the write pointer will be incremented to 3'b110. So, the possibility of error will be more as two bits are changing here. So, instead we will first convert to grey code before sending and while comparing it will again be converted back to binary. This way, the circuit will be less prone to errors.

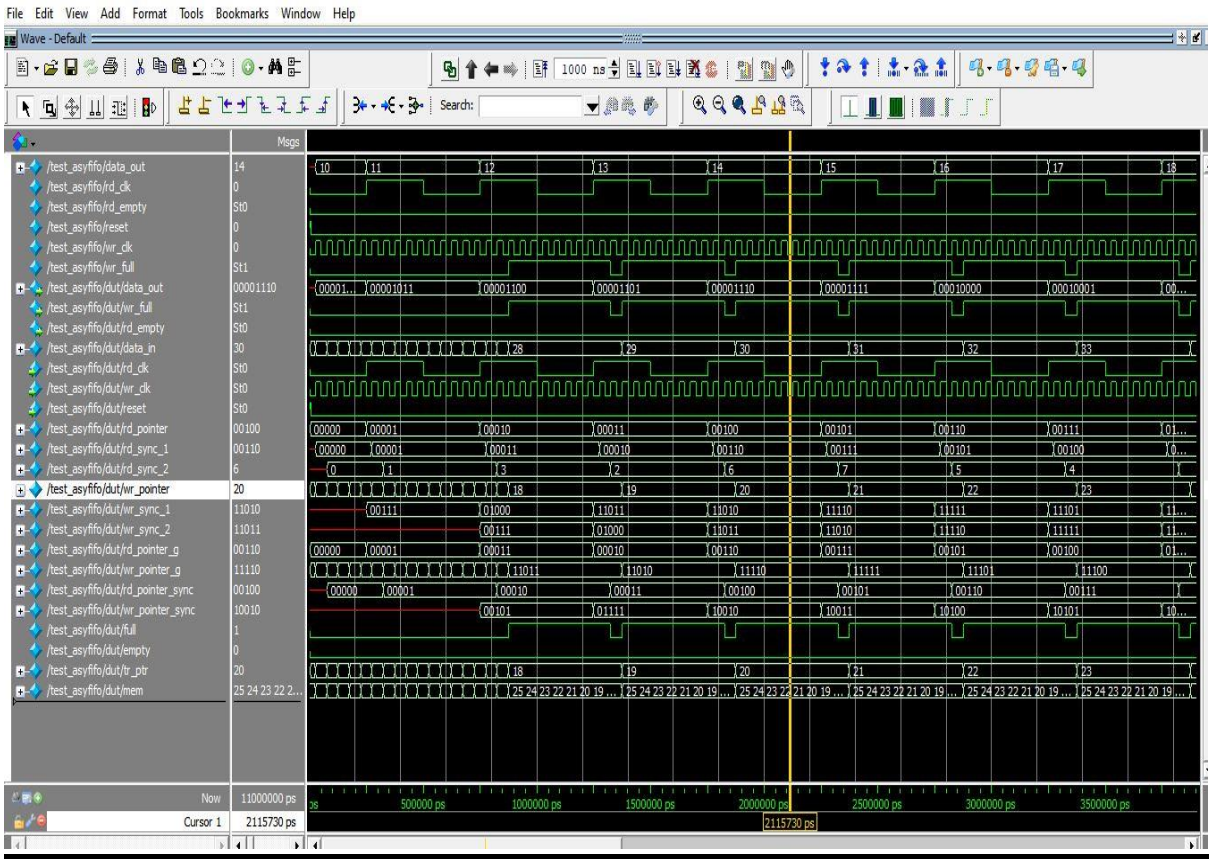
Simulation:

The simulation is done in Verilog. A FIFO of 16 X 8 is considered. For transmitting the data, a ROM of 128 X 8 is considered. The data will be sent in sequence to the FIFO and will be stopped when full flag is set and it will resume sending when full flag will become zero again.

The following parameters are taken in the test bench.

	Frequency(MHz)
Write-clock	20
Read-clock	2

Waveform:



Conclusion:

Thus a synthesisable asynchronous FIFO has been successfully implemented and tested in Verilog.