Asynchronous FIFO

Self Project

BY

P.Nagasai Goud 203070094

DEPARTMENT OF ELECTRICAL ENGINEERING

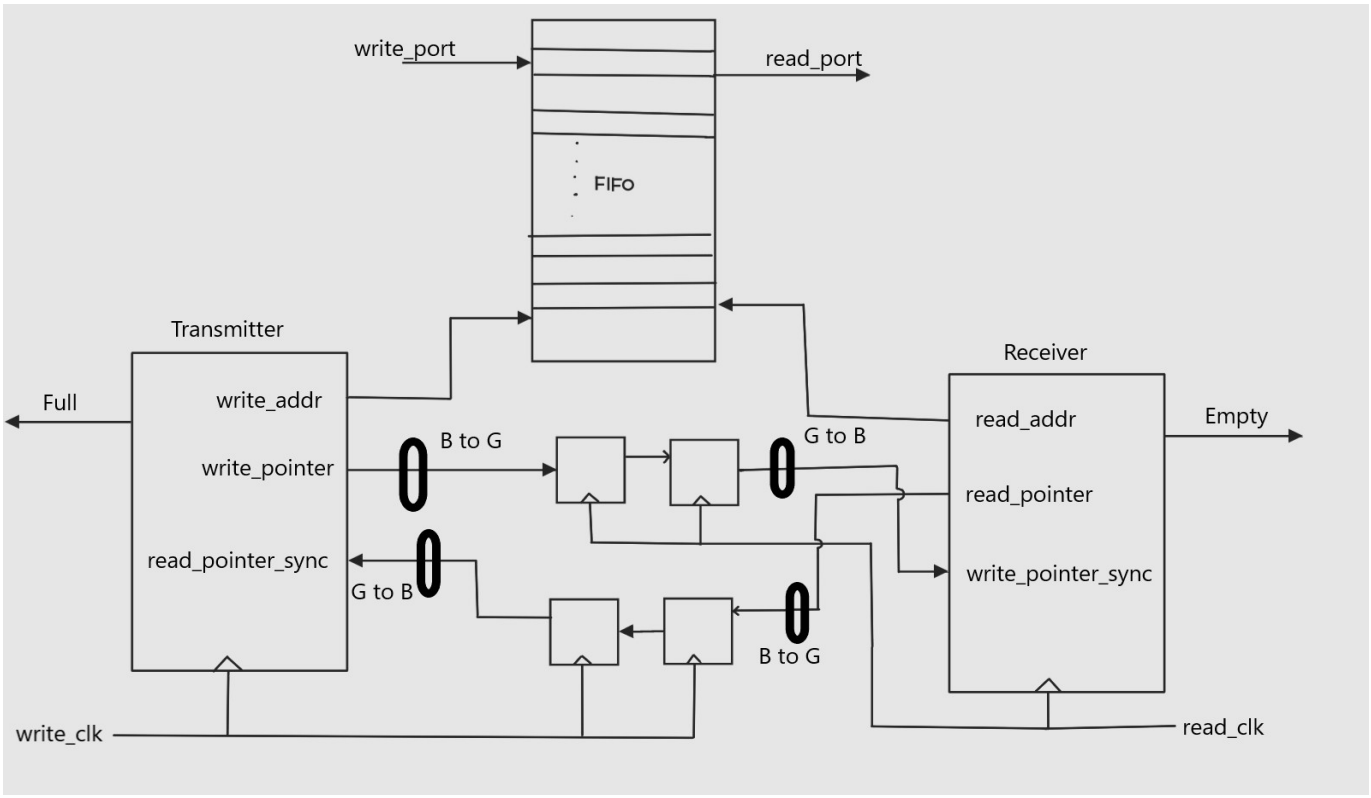
IIT BOMBAY

Main Gate Rd, IIT Area, Powai, Mumbai, Maharashtra 400076

Introduction

FIFO's are often used to safely pass data from one clock domain to other clock domain. Using a FIFO to pass data from one clock-domain to another clock domain requires multi-asynchronous clock design. An Asynchronous FIFO refers to FIFO design where data values are written to a buffer from one clock domain and the data values are read from the same FIFO buffer from another clock domain, where the two clock domain are asynchronous to each other.

Asynchronous FIFO Block Diagram :-

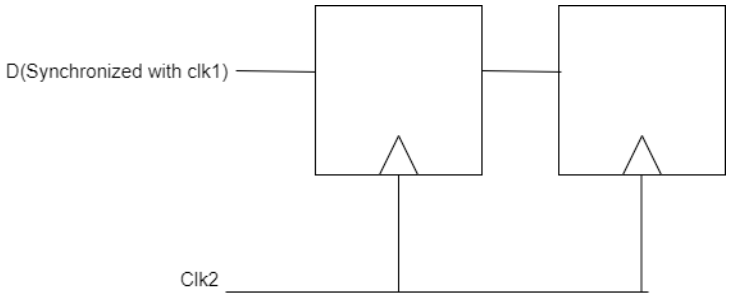


The transmitter hold the write port, the receiver controls read port. Thus the transmitter dump data into queue while the receiver reads it out. read_pointer is generated by receiver on read_clk, write_pointer is generated by transmitter on write_clk. The FIFO has two ports. One port is used exclusively for writing the other for reading there are two pointers in the FIFO, the read pointer and the write pointer. The read pointer indicates the next position we should read from .The write pointer is very similar, it indicates the next position to write to.

There are two flags derived from the read and write pointers. These flags are critical for correct operation of the FIFO. Notice that the write port should not write to a position that has not been read yet. Only a clear address can be written, and an address is "clear" only if it has been already read. But what if there are no positions that can be written to? If this happens, then all the FIFO is yet to be read, and we must wait until a read is performed before we can write. In this condition, we say the FIFO is full. When is a FIFO full? When all locations are unread. Note the read pointer indicates the next location to be read and the write pointer indicates the next position to be written.If the write pointer and read pointer is equal then we can say FIFO is full.

The other flag is the empty flag. When a FIFO is empty, it has no locations carrying unread information to be read. Thus, we have to wait for a new word to be written to the queue before we try to read. So, if we read a word and increment the address, then find that the read pointer is equal to the write pointer, then the FIFO is empty. This is because the next location to be read is also the next location to be written.

Note that the condition for empty and full is the same: The read and write pointers are equal. If we increase the read pointer and find equality, then the FIFO is empty. If we incremented the write pointer and find equality, then the FIFO is full. Thus, the last action before equality decides the state. To calculate the full flag, the read pointer is synchronized to write_clk by a two register synchronizer. This is then subtracted from the write pointer, to produce the flag. Similarly, to calculate the empty flag, the write pointer is synchronized to read_clk by a two register synchronizer before the receiver calculates the empty flag.



The synchronization allows the empty and full flags to be calculated reliably without being affected by metastability.

Using n-bit pointers where (n-1) is the number of address bits required to access the entire FIFO memory buffer, the FIFO is empty when both pointers, including the MSBs are equal. And the FIFO is full when both pointers, except the MSBs are equal.

The asynchronous FIFO is extremely efficient at transmitting long bursts of data. Its only delay overhead occurs when the empty or full flags are raised. In conditions where the receiver is much faster than the transmitter, the empty flag is raised often. In conditions where the transmitter is much faster than the receiver, the full flag is raised often. The latter case can be mitigated by increasing the size of the FIFO.

In conditions where the two clocks are close to each other in frequency, the communication can be very efficient, with the empty or full flags raised rarely. The read and write flags are both multi-bit words. Yet we still pass them through two register synchronizers. Buses should never be passed through synchronizers because the one cycle uncertainty of synchronization affects different bits differently.

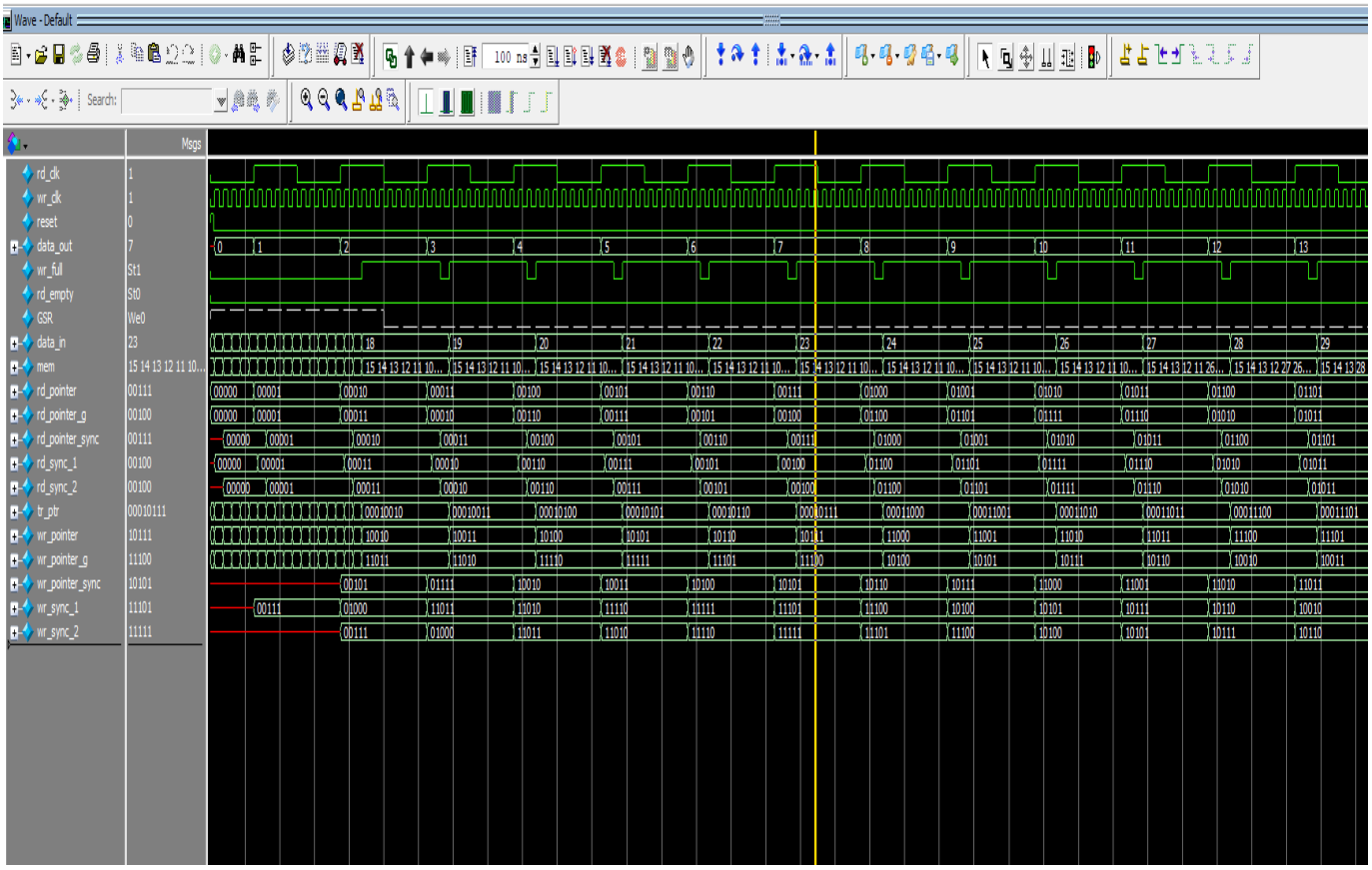
However, the read and write pointers are always Gray encoded. This means that they do not increment in binary order, but rather in a Gray-encoded order. Thus, there is at most a single bit difference in every cycle if the pointers increment. Thus, the synchronizers are effectively only synchronizing a single bit change, and they can be safely used.

Here FIFO Size is 16×8 and 5-bit pointers(MSB used for flag calculation and remaining 4-bits used for address)

Read clock Frequency : 20MHz

Write Clock Frequency : 200MHz

Transmitter sending data continuously 8 bits of data with frequency of 200MHz.


Wave - Default
Msgs
rd_clk
wr_clk
reset
data_out
wr_full
rd_empty
GSR
data_in
mem
rd_pointer
rd_pointer_g
rd_pointer_sync
rd_sync_1
rd_sync_2
tr_ptr
wr_pointer
wr_pointer_g
wr_pointer_sync
wr_sync_1
wr_sync_2