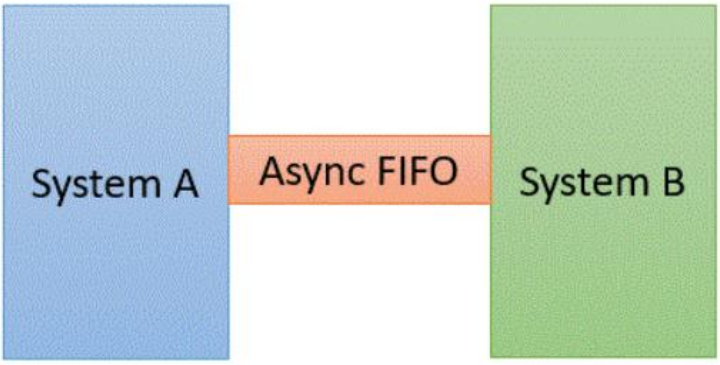
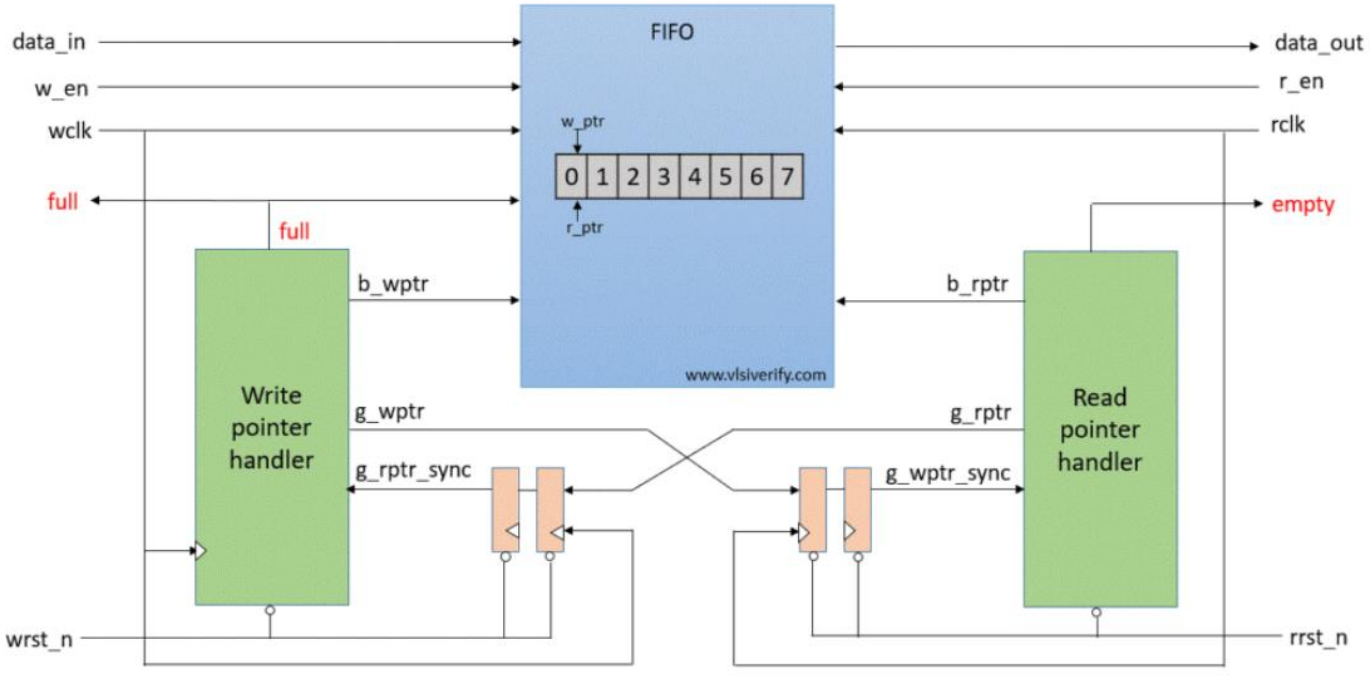
Asynchronous FIFO Description

- In asynchronous FIFO, data read and write operations use different clock frequencies. Since write and read clocks are not synchronized, it is referred to as asynchronous FIFO. Usually, these are used in systems where data needs to pass from one clock domain to another which is generally termed as 'clock domain crossing'. Thus, asynchronous FIFO helps to synchronize data flow between two systems working on different clocks.



- In the case of synchronous FIFO, the write and read pointers are generated on the same clock. However, in the case of asynchronous FIFO write pointer is aligned to the write clock domain whereas the read pointer is aligned to the read clock domain. Hence, it requires domain crossing to calculate FIFO full and empty conditions. This causes metastability in the actual design. In order to resolve this metastability, 2 flip flops or 3 flip flops synchronizer can be used to pass write and read pointers. For explanation, we will go with 2 flip-flop synchronizers. Please note that a single "2 FF synchronizer" can resolve metastability for only one bit. Hence, depending on write and read pointers multiple 2FF synchronizers are required.



Asynchronous FIFO

➢ Parameters

- **FIFO_WIDTH** → FIFO word width (default = 16 bits)
- **FIFO_DEPTH** → Number of FIFO locations (default = 512)
- **ADDR_SIZE** → Address line of FIFO (default = $\log_2(512) = 9\ bits$)

➢ Input & output ports

port	Width	Direction	Function
data_in	FIFO_WIDTH	Inputs	Write Data: The input data bus used when writing the FIFO.
wr_en	1		Write Enable: If the FIFO is not full, asserting this signal causes data (on data_in) to be written into the FIFO
rd_en	1		Read Enable: If the FIFO is not empty, asserting this signal causes data (on data_out) to be read from the FIFO
clk_wr	1		Clock signal for write port, used in the writing operation
clk_rd	1		Clock signal for read port, used in the reading operation
rst	1		Active high Asynchronous reset. It resets the data_out, internal write counter & internal read counters
data_out	FIFO_WIDTH	Outputs	Read Data: The output data bus used when reading from the FIFO.
full	1		Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is not destructive to the contents of the FIFO.
empty	1		Empty Flag: When asserted, this signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is not destructive to the FIFO.
overflow	1		Overflow flag: when asserted, this signal indicated that the FIFO is full & the write enable is also activated (sequential output in write domain)
underflow	1		underflow flag: when asserted, this signal indicated that the FIFO is empty & the read enable is also activated (sequential output in read domain)

❖ Additional Notes

1) Usage of Binary to gray converter in Asynchronous FIFO.

- Till now, we discussed how to get asynchronous write and read pointers in respective clock domains. However, we should not pass binary formatted write and read pointer values. Due to metastability, the overall write or read pointer value might be different, this problem is known as **Bus synchronization issue.**
- Example: When binary value wr_ptr = 4'b1101 at the write clock domain is transferred via 2FF synchronizer, at the read clock domain wr_ptr value may receive as 4'b1111 or any other value that is not acceptable. Whereas gray code is assured to have only a single bit change from its previous value. Hence, both write and read pointers need to convert first to their equivalent gray code in their corresponding domain and then pass them to an opposite domain. To check FIFO full and empty conditions in another domain.

2) Ignoring Data loss issue due to travelling from fast domain into slow domain as the FIFO concept is one of the most used techniques that handle this problem easily.