

## Self Project

At the core of this FIFO design lies a memory with a capacity of **16 locations** in terms of depth and **8 bits** in terms of width. The design incorporates several input signals, including:

- **Write Data (8 bits)**
- **Write Enable**
- **Read Enable**
- **Write Clock**
- **Write Address (4 bits)**
- **Read Address (4 bits)**

An output signal is also present, which is the Read Data (8 bits).

The process of using this FIFO involves the following steps:

1. **Writing Data:** Data to be written and the address where it should be written are fed into the inputs, specifically the Write Data and Write Address ports. Upon the positive edge of the Write Clock signal, if the Write Enable signal is active, the provided data is written into the FIFO memory.
2. **Reading Data:** To retrieve data from the FIFO memory, the Read Enable signal needs to be activated. Additionally, the address from which the data should be read is specified through the Read Address input.

In summary, this encapsulates the memory operations. To operate the memory in accordance with the FIFO's requirements, the following must be adhered to:

- Writing data involves supplying data and its designated address through the Write Data and Write Address inputs. On the positive edge of the Write Clock, with Write Enable active, data is stored in the FIFO memory.
- Reading data requires activating the Read Enable signal and providing the relevant address through the Read Address input. This initiates the process of extracting data from the FIFO memory.

- The aim is to orchestrate the memory operations to ensure compliance with the FIFO's functioning principles.

## 2. BINARY & GRAY COUNTER :

- We need to design a counter which can give Binary and Gray output's, the need for Binary counter is to address the FIFO MEMORY i.e. Write and Read address. And the need of Gray counter is for addressing Read and Write pointers. Once the counter with binary and Gray code output is designed it is then Port mapped with Memory's Read address, write address, Read pointer, Write Pointer.
- The Use Full and Empty logic for addressing the memory Empty: the counter takes Empty signal and increments the Read address depending on this.
- Full: when ever the Full signal is high the counter should not increment write address
- If ( $\sim$ EMPTY)                      If ( $\sim$ FULL)  
Increment Read Address    Increment Write Address  
Else No increment              Else No Increment

## 3. SYNCHRONIZER'S:

- Synchronizers are very simple in operation; they are made of 2 D Flip Flop's. As the FIFO is operating at 2 different clock domains so there is a need to synchronize the Write and Read pointers for generating empty and full logic which in turn is used for addressing the FIFO memory.
- The Figure below shows how synchronization takes place; the logic behind this is very simple. What we are trying to do over here is , passing the Write Pointer to a D Flip Flop which is driven by the Read clock and in the same manner the Read pointer is fed to a D Flip Flop which is driven by Write Clock, so as a result of this we get Read Pointer (which is operating under Read clock) and Synchronized Write Pointer which is also operating under Read clock, and the same with Write pointer and Synchronized Read Pointer, so now we can compare them and derive a logic for Generating

Empty and Full conditions, which is the most important design part of this FIFO

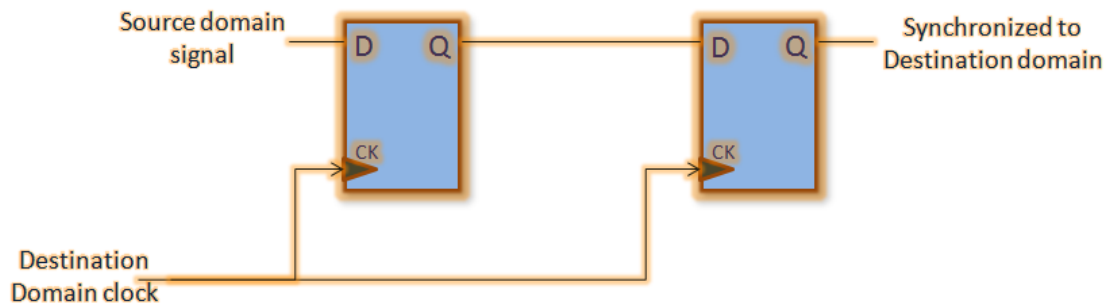


Fig: Dual Synchronizer

### Empty Logic generation:

- When the FIFO is full, the write pointer must stop writing values into it and when the FIFO is empty, the read pointer cannot read any value from it
- So, 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 0<sup>th</sup> 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
- This module is completely synchronous to the read-clock domain and contains the FIFO read pointer and empty-flag logic.

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

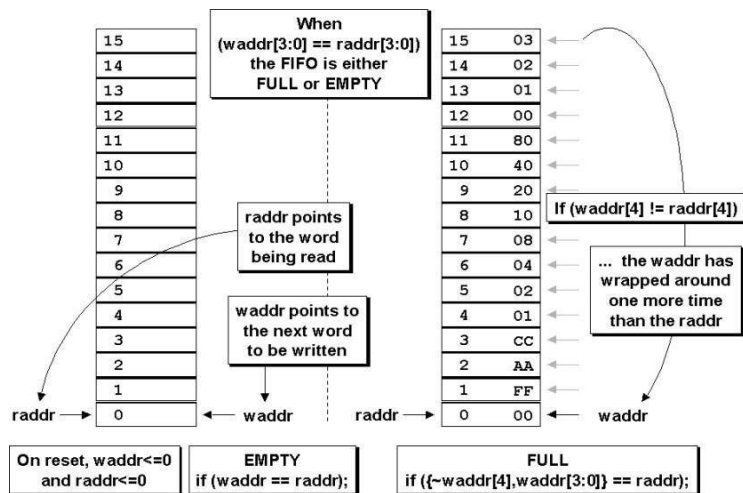


Fig : Full and empty generation logic

## Results:

- The whole design was implemented in Quartus using Verilog and simulated in ModelSim with proper testbenches. The below waveforms shows the functionality of **16X8 Asynchronous FIFO**.

## RTL View:

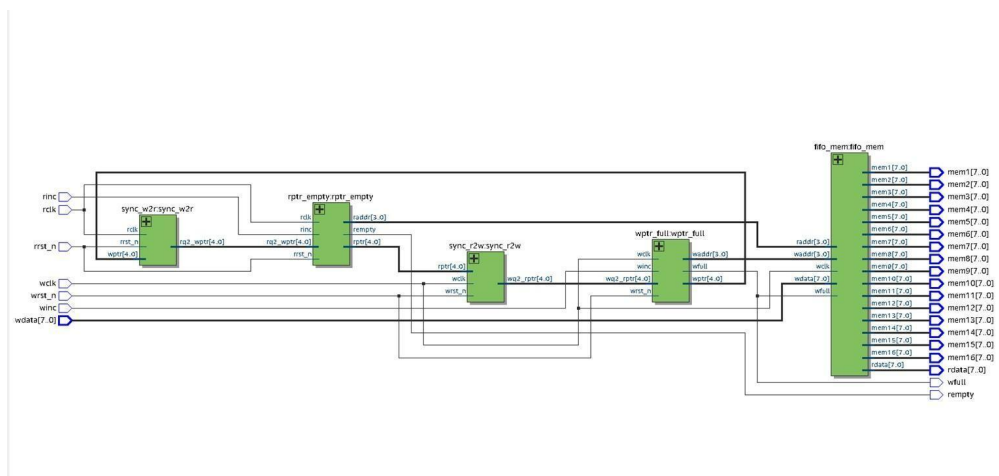
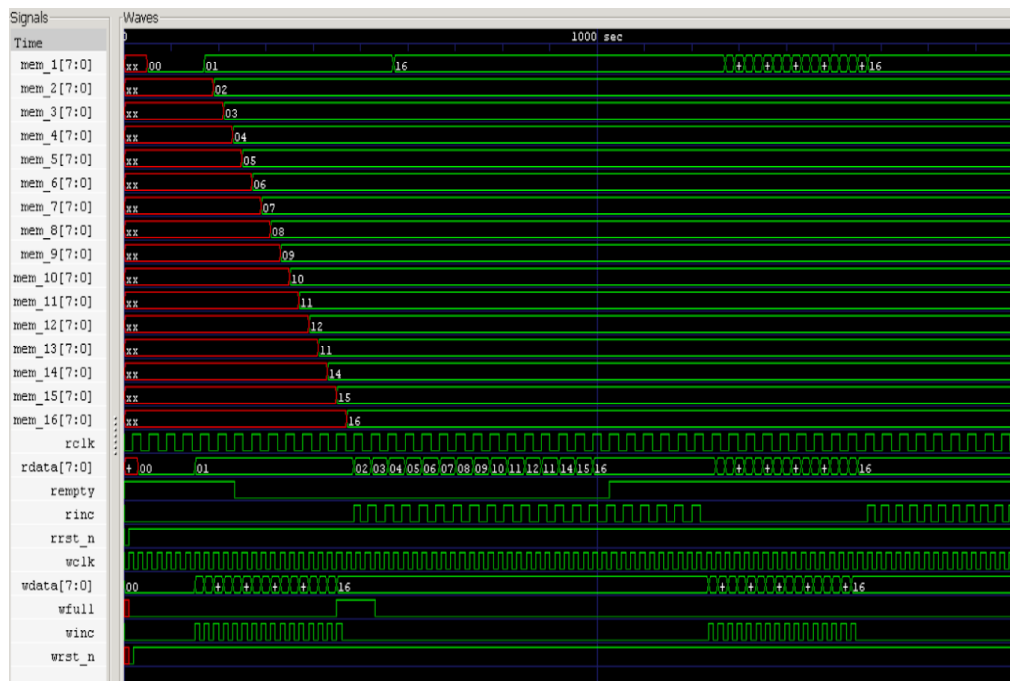


Fig : RTL view of the Asynchronous FIFO

### TestBench 1 and it's Result:

- Here we have given winc signal '1' such that FIFO is written by all 16 addresses and then we can see that the wfull flag is high ,then after some amount of delay we have read all those 16 words from the FIFO then we have observed the rempty signal is high.



### TestBench 2 and it's Result:

- Here, we have first written 14 times to the FIFO and then read all those 14 words from the FIFO, so from this, we can conclude even if FIFO is about to be full then the *wfull* signal is not lit and then after reading all the 14 words from the FIFO the empty signal starts hitting high

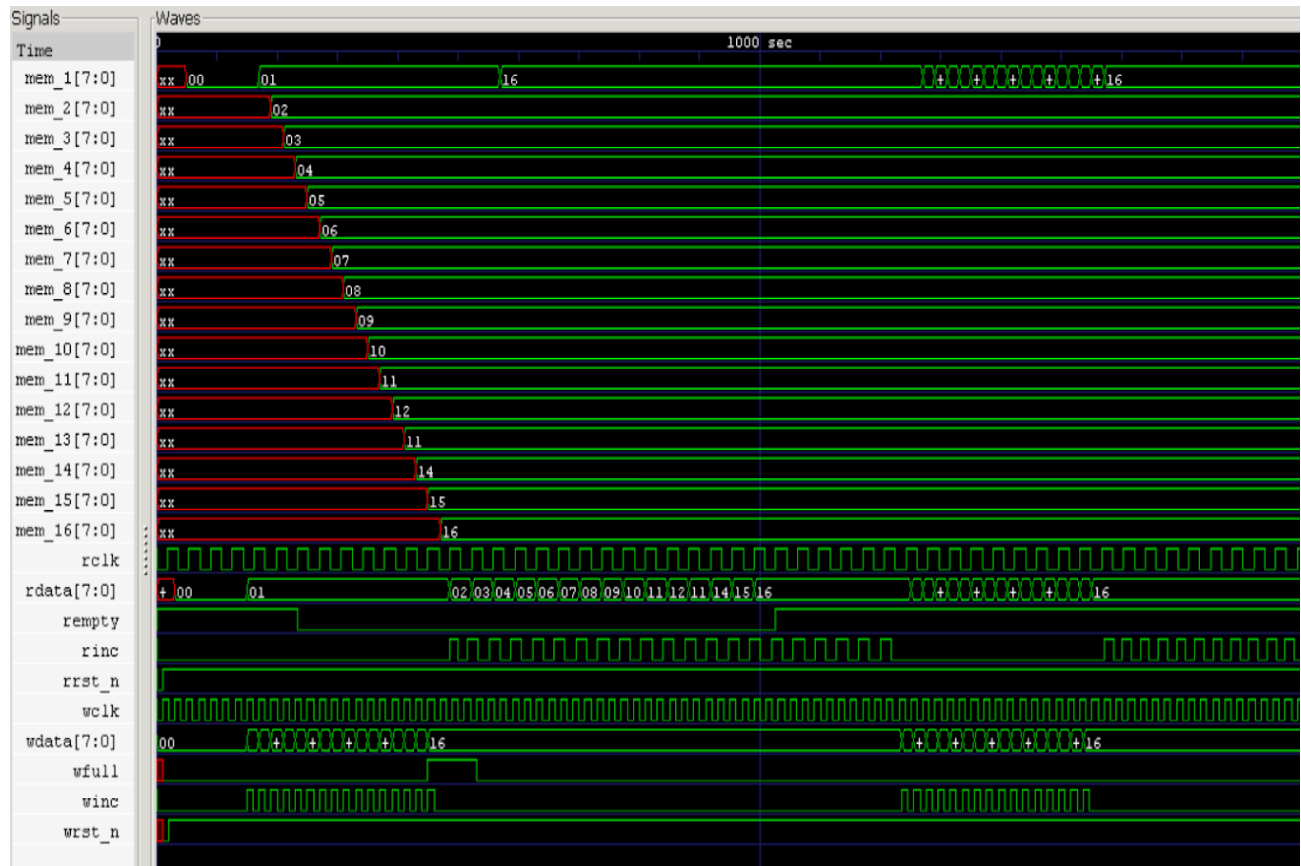


Fig : partial write and read completely for rempty signal function to be validated

### TestBench 3 and it's Result:

- Then in this case we are writing sequentially 16 words and reading 16 words from the FIFO after a certain amount of delay again we are writing completely to the FIFO and reading completely from the FIFO sequentially and verified its behavior

