Minor Project

Asynchronous FIFO

Submitted by: Irfan Ahmad

Student ID: 2023PVL0089

Email ID: 2023PVL0089@iitjammu.ac.in

Submitted to: Assistant Professor Satyadev Ahlawat

November 2023

Introduction

FIFO, or "First In, First Out," is a computing principle and data structure that follows the order of items based on their arrival sequence. In a FIFO system, the first element added is the first to be removed, akin to a queue where the first person in line is the first to be served. This concept is commonly implemented using a data structure called a queue, where elements are enqueued (added) at the rear and dequeued (removed) from the front. FIFO is widely applied in various computing domains, including operating systems for process scheduling, networking for data packet transmission, and hardware design for data buffering. It ensures that data or tasks are processed in the order they are received, contributing to system predictability and maintaining the integrity of sequential operations. The simplicity and effectiveness of FIFO make it a foundational principle in computer science, playing a key role in designing efficient.

Problem Statement

Design a SRAM based FIFO for the following specifications:

1. The word size should be 8bit for the SRAM memory that means you are reading or writing 1 Byte of data from the SRAM.

2. The number of address locations in your SRAM should Date of your birth in MB. For example if your D.O.B. is 25/08/2001 then the size of memory will be 25MB.

3. For synchronization of your address pointers you have to consider binary encoding only. You shall use a design approach which gives you best possible throughput for the binary encoding.

4. You should also keep the condition in consideration when the synchronizer takes 2 cycles to pass the value due to meta-stability condition.

5. Add adequate comments in your VHDL code for readability.

Approach

Since my DOB is 12th of November, so I have to construct a RAM of 12MB. For that taking a word length of 1 byte of memory its required to have an address with of 24 depth.

Block Diagram of FIFO

Here is the basic block diagram of the FIFO.

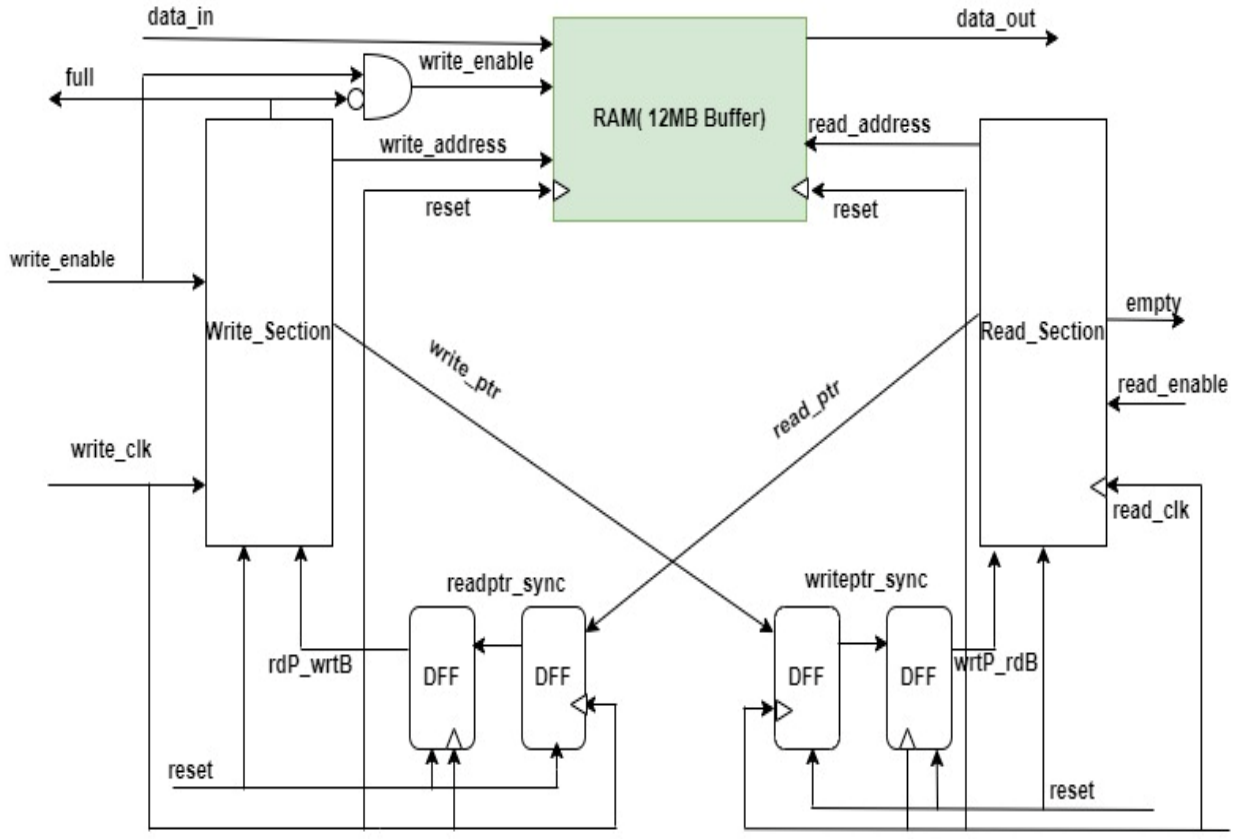

Figure 1:Basic Block Diagram of FIFO

VHDL code of the simulation of the FIFO

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

use ieee.std_logic_unsigned.all;

use ieee.std_logic_arith.all;


entity fifo is

generic(

word_width : integer := 8;

addr_bus : integer := 24);
```

```
port(
 ---------------------------- input data
write_clk : in std_logic;
reset : in std_logic;
write_enable : in std_logic;
full : out std_logic;
data_in : in std_logic_vector(word_width -1 downto 0);
------------------------------output data
read_clk : in std_logic;
read_enable : in std_logic;
empty : out std_logic;
data_out : out std_logic_vector(word_width -1 downto 0));
end fifo;
--Create SRAM
architecture Behavioral of fifo is
type memory is array((2**addr_bus) -1 downto 0) of
std_logic_vector(word_width-1 downto 0);
--Siganls  from write block
signal RAM : memory;
signal write_full : std_logic;
signal write_ptr_nxt : std_logic_vector (addr_bus downto 0);
signal write_addr_nxt : std_logic_vector(addr_bus-1 downto 0);
signal wrtP_rdB : std_logic_vector(addr_bus downto 0);
signal write_addr: std_logic_vector(addr_bus-1 downto 0);
```

```
signal write_ptr : std_logic_vector(addr_bus downto 0);
--signals  from the read block
signal read_empty : std_logic;
signal read_ptr_nxt: std_logic_vector (addr_bus downto 0);
signal read_addr_nxt : std_logic_vector(addr_bus-1 downto 0);
signal rdP_wrtB : std_logic_vector(addr_bus downto 0);
signal read_addr: std_logic_vector(addr_bus-1 downto 0);
signal read_ptr : std_logic_vector(addr_bus downto 0);
--signals form 2 ff sychronizer
signal wrtptr_sync : std_logic_vector(addr_bus downto 0);
signal rdptr_sync : std_logic_vector(addr_bus downto 0);

begin
--Writing the SRAM on the specific address
process(write_clk,write_enable)
begin
 if(write_clk'event and write_clk = '1')then
 if(write_enable ='1' and not write_full ='1')then
 RAM(conv_integer(write_addr)) <= data_in;
end if;
end if;
end process;
--incrementing write pointer and write address
process(write_clk,reset)
```

```
begin
 if(reset = '1')then
  write_ptr <= (others => '0');
  write_addr<= (others => '0');
 elsif(write_clk'event and write_clk='1')then
   write_ptr <= write_ptr_nxt;
   write_addr <= write_addr_nxt;
 end if;
end process;
-- increment of write pointer binary by 1
write_ptr_nxt <= ( write_ptr + 1) when write_enable ='1' and not write_full ='1' else write_ptr;
-- N-1 bits to address SRAM
write_addr_nxt <= write_ptr_nxt(addr_bus-1 downto 0);
--full condition : it is checked by comparing write pointer and snyced read pointer th the values
write_full <= '1' when write_ptr(addr_bus) /= rdP_wrtB(addr_bus) and write_ptr(addr_bus-1)=
rdP_wrtB(addr_bus-1)  else '0';
full <= write_full;


--sync write
process(write_clk,reset)
 begin
  if(reset ='1')then
   wrtptr_sync <= (others => '0');
   rdP_wrtB <= (others => '0'); -- rd_ptr_into_wrt_section  singifies read pointer which is
synced with write clock
```

```
 elsif (write_clk'event and write_clk ='1')then
   wrtptr_sync <= read_ptr;
   rdP_wrtB<= wrtptr_sync;
end if;
end process;
--Reading the data from SRAM
process(read_clk,read_enable)
 begin
  if(read_clk'event and read_clk = '1') then
   if(read_enable ='1' and (not read_empty) = '1') then
     data_out <= RAM(conv_integer(read_addr));
   end if;
  end if;
end process;
--assigning data calculated next value of read pointer and address into current read pointer and address
process(read_clk,reset)
begin
if(reset='1')then
 read_ptr <= (others => '0');
 read_addr<= (others => '0');
elsif(read_clk'event and read_clk='1')then
read_ptr <= read_ptr_nxt;
read_addr <= read_addr_nxt;
end if;
```

```
end process;
-- increment of read pointer binary by 1
 read_ptr_nxt <= ( read_ptr + 1) when read_enable ='1' and not read_empty ='1' else
 read_ptr;
-- N-1 bit to refer address of buffer
 read_addr_nxt <= read_ptr_nxt(addr_bus-1 downto 0);
 --generate empty condition
 read_empty <= '1' when  read_ptr = wrtP_rdB  else '0';
 empty <= read_empty;
-- 2FFs synchronizer to sync wrt pointer into read block with read clock
process (read_clk,reset)
begin
if(reset ='1')then
 rdptr_sync <= (others => '0');
 wrtP_rdB <= (others => '0');
elsif (read_clk'event and read_clk ='1')then
 rdptr_sync <= write_ptr;
wrtP_rdB<= rdptr_sync;
end if;
end process;
end Behavioral;
```

As we as supposed to make the FIFO with the buffer size of 12MB for which testing will be quite difficult so doing it for a 4 Byte memory so that every condition is to be checked.

For which address width is 2 bits and the word width will be of 1 Byte and the total memory locations will be 4. Here is the result for empty and full condition at different clock frequencies.

Testing Cases

1) Reading and writing into the memory with same read and write clock frequency.

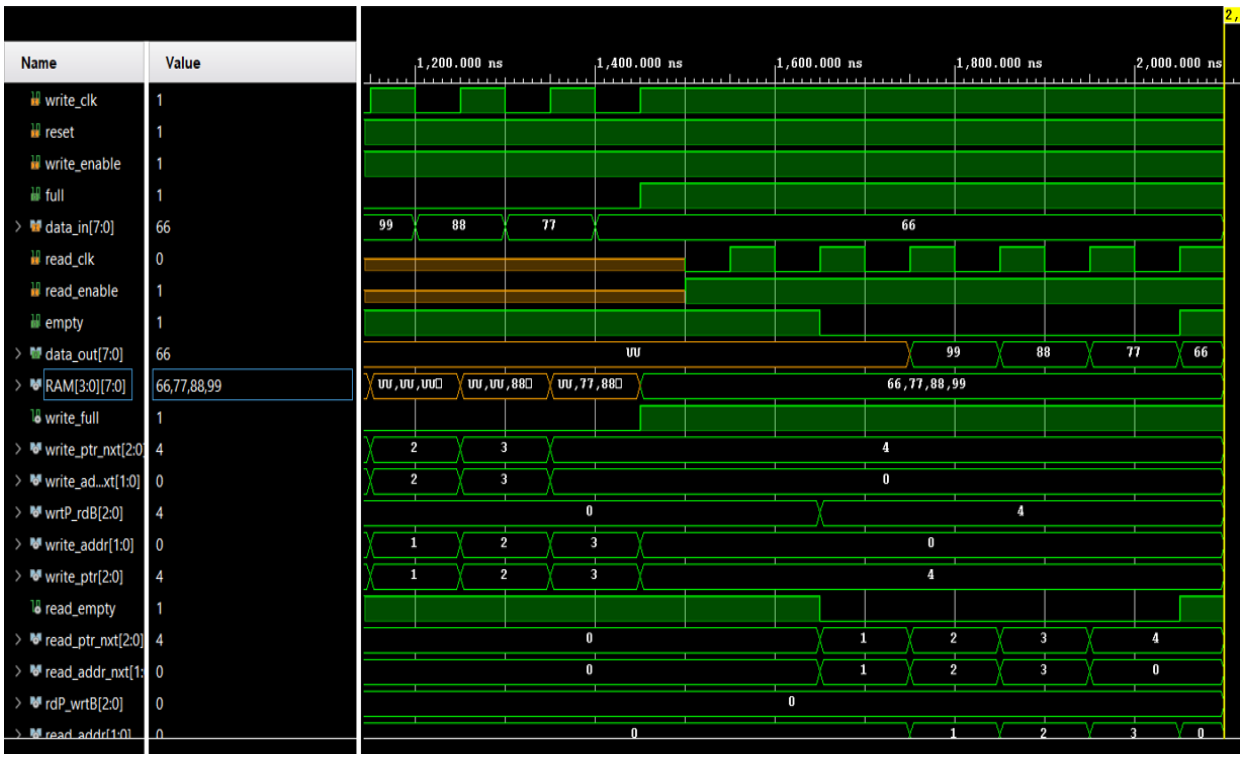

Figure2: Full goes high after writing the full memory and empty goes high after reading the entire memory where the write clock frequency is same as that of read clock frequency.

2) Reading and writing into the memory with reading clock frequency double as that of write clock frequency.

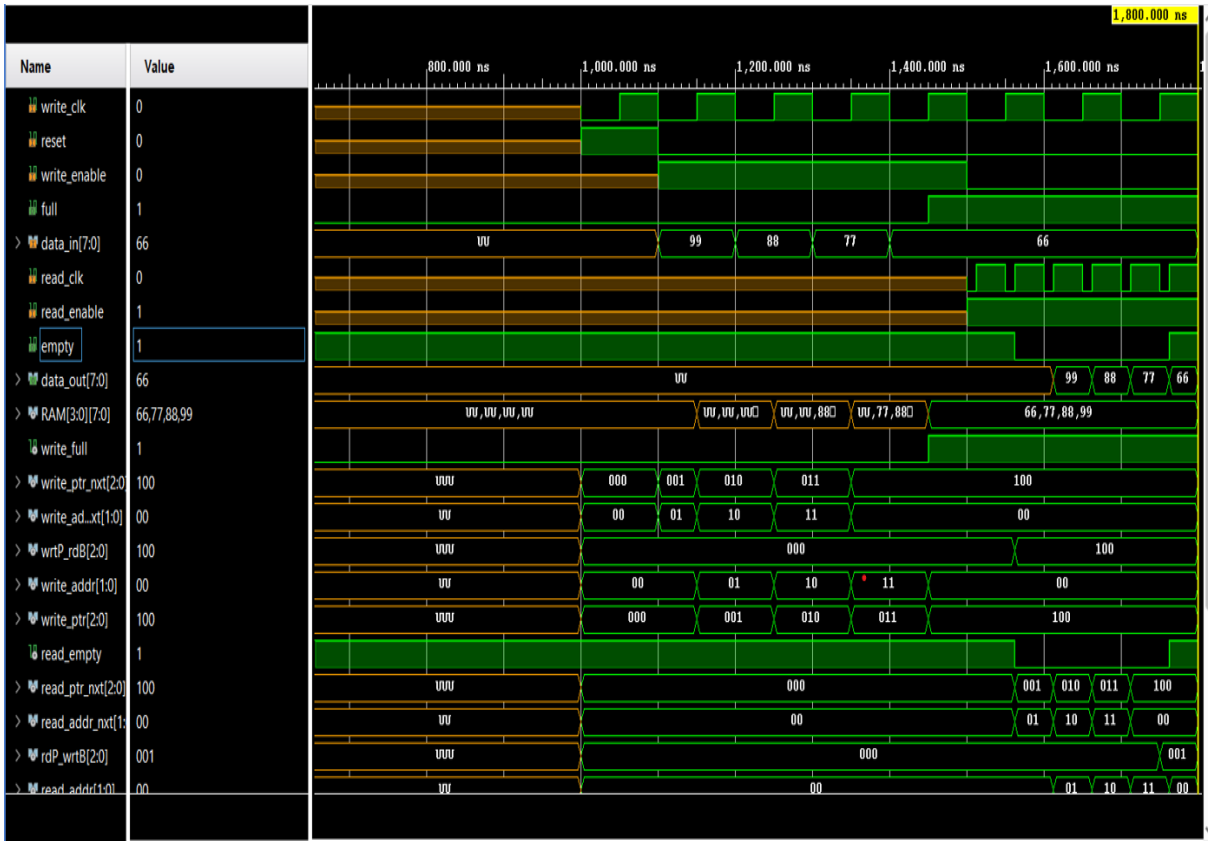

Figure3: Full goes high after writing the full memory and empty goes high after reading the entire memory where the write clock frequency is half as that of read clock frequency.

3) Reading and writing into the memory with reading clock frequency half as that of write clock frequency.

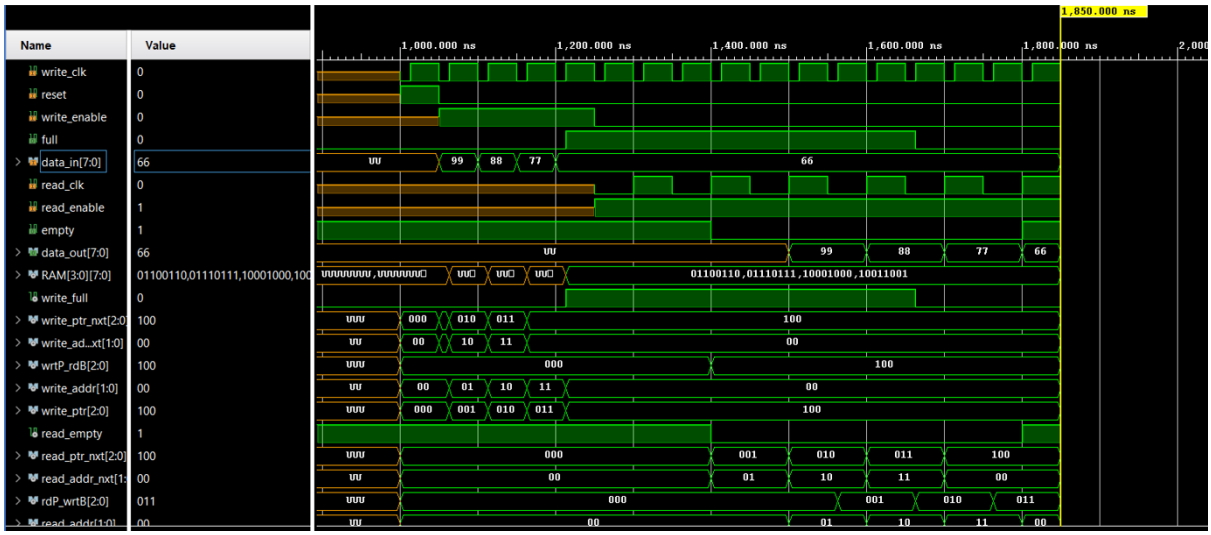

Figure4: Full goes high after writing the full memory and empty goes high after reading the entire memory where the write clock frequency is double as that of read clock frequency.

4) When both the read and write clocks are active, data is being pushed into the memory and reading is being done.

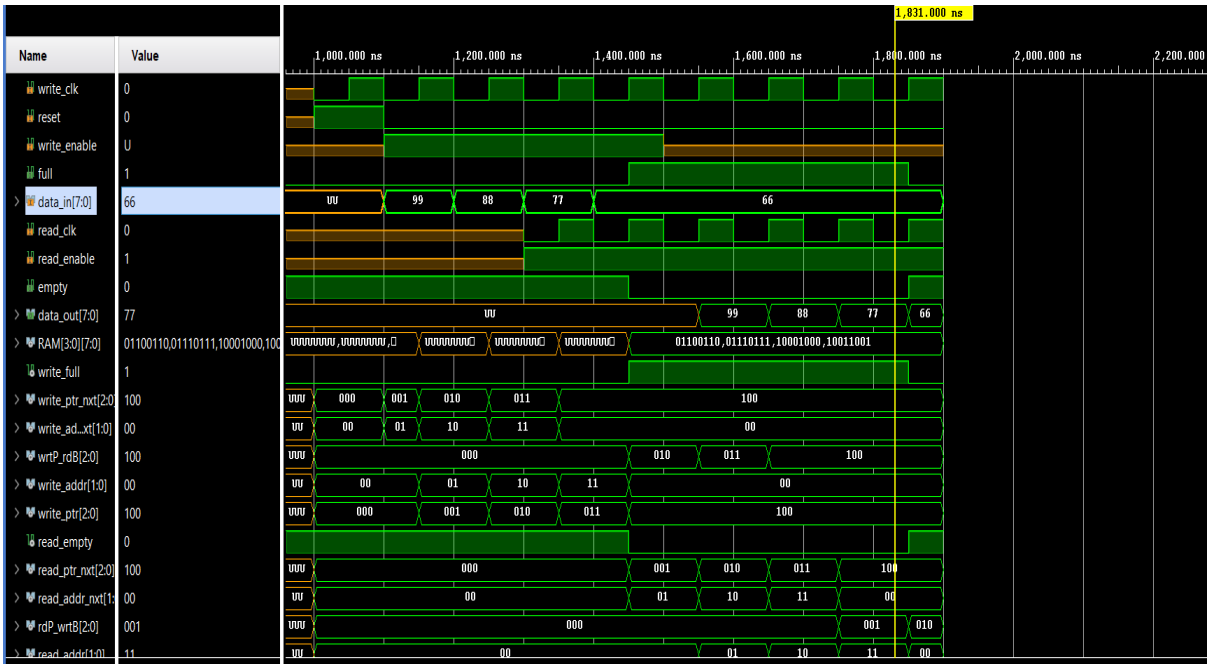

Figure 5: Data being written and read at the same time.

Conclusion

We are able to read and write the data from the memory without any loss of data when reading, writing clock frequencies are different or same.