Standard FIFO Memory

(Project Report)

Under the guidance of :

Dr. Vipin Kamble

Parija Malgaonkar : BT18ECE040

Anuj Nakade: BT18ECE041

Divyanshu Rahate: BT18ECE043

Index

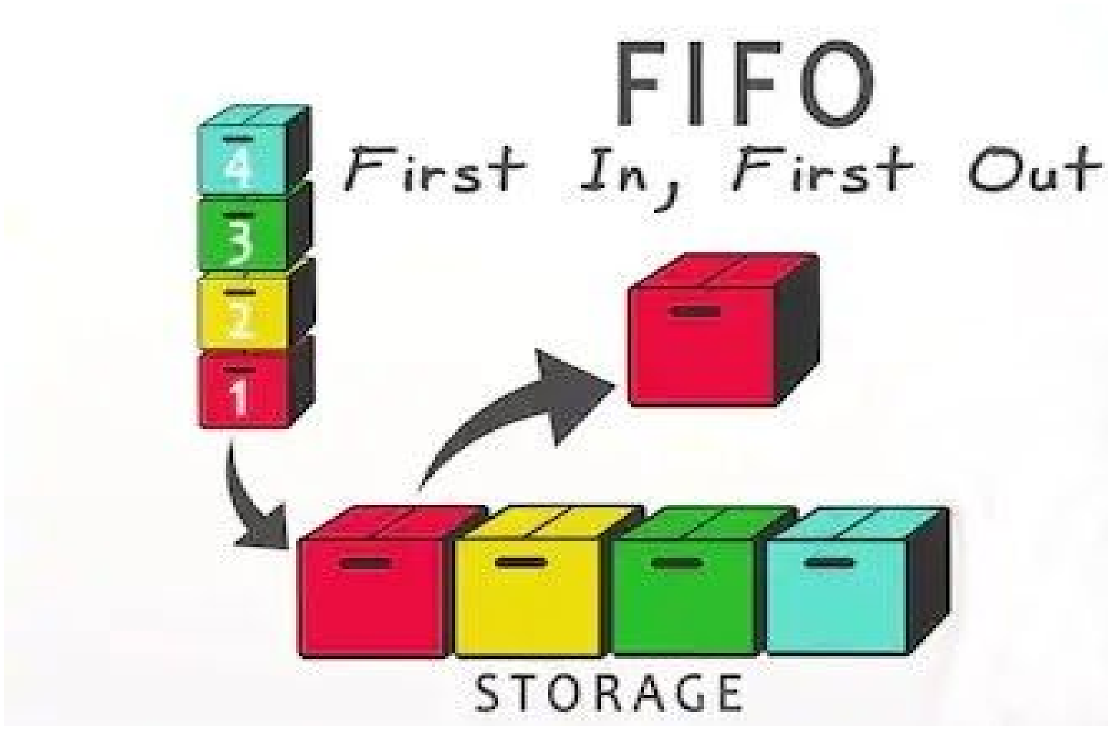
Introduction	**3**
Theory	**4**
Code	**6**
Output	**8**
Observation	**9**
References	**10**

Introduction

A buffer, also called **buffer memory**, is a portion of a computer's memory that is set aside as a temporary holding place for data that is being sent to or received from an external device. There are several types to these memory buffers, one such type is the **FIFO Memory Buffer**.

FIFOs (First In, First Out) are essentially memory buffers used to temporarily store data until another process is ready to read it. As their name suggests the first byte written into a FIFO will be the first one to appear on the output.

To understand the concept of FIFO, you can imagine a queue of people waiting to get on the bus at a bus stand. The first person to arrive at the bus stand will be the first person in the queue who will get on the bus first.



Theory

The FIFO Module used in the code has two settings that can be altered. The first one is **DATA_WIDTH** that can be used to configure the width of the FIFO, that is, the **DataIn** and **DataOut** buses so that you can write different sizes of bytes if needed. The second one is the **FIFO_DEPTH** variable that adjusts how big the internal memory of the FIFO is going to be.

The Head and Tail variables both start from 0 when the module is Reset, that is **RST** = 1. That's when the **Full** flag becomes 0 and the **Empty** flag becomes 1. This is when we know that the FIFO module has been reset and that we can push data onto it and read the pushed data.

Write Process:

In order to write data into the FIFO first push the data onto the **DataIn** bus and then strobe the **WriteEn** input high for one clock cycle. This will write whatever is on **DataIn** into the FIFOs internal memory. The code checks whether Reset is set to 0, which makes Looped= false, or it checks whether Head is not equal to Tail, only then it carries out the pushing of that data.

As we write data onto the **DataIn** bus, we make sure that the **Head** pointer is updated and moved forward along the size of the memory, while the **Tail** pointer stays on 0 as we will use that to read the data that has been pushed.

When the **Full** flag goes high, this means that the FIFO’s memory is full and will not accept any more writes until data is read using the **ReadEn** input. If data is written while the **Full** flag is high it will be ignored.

Read Process:

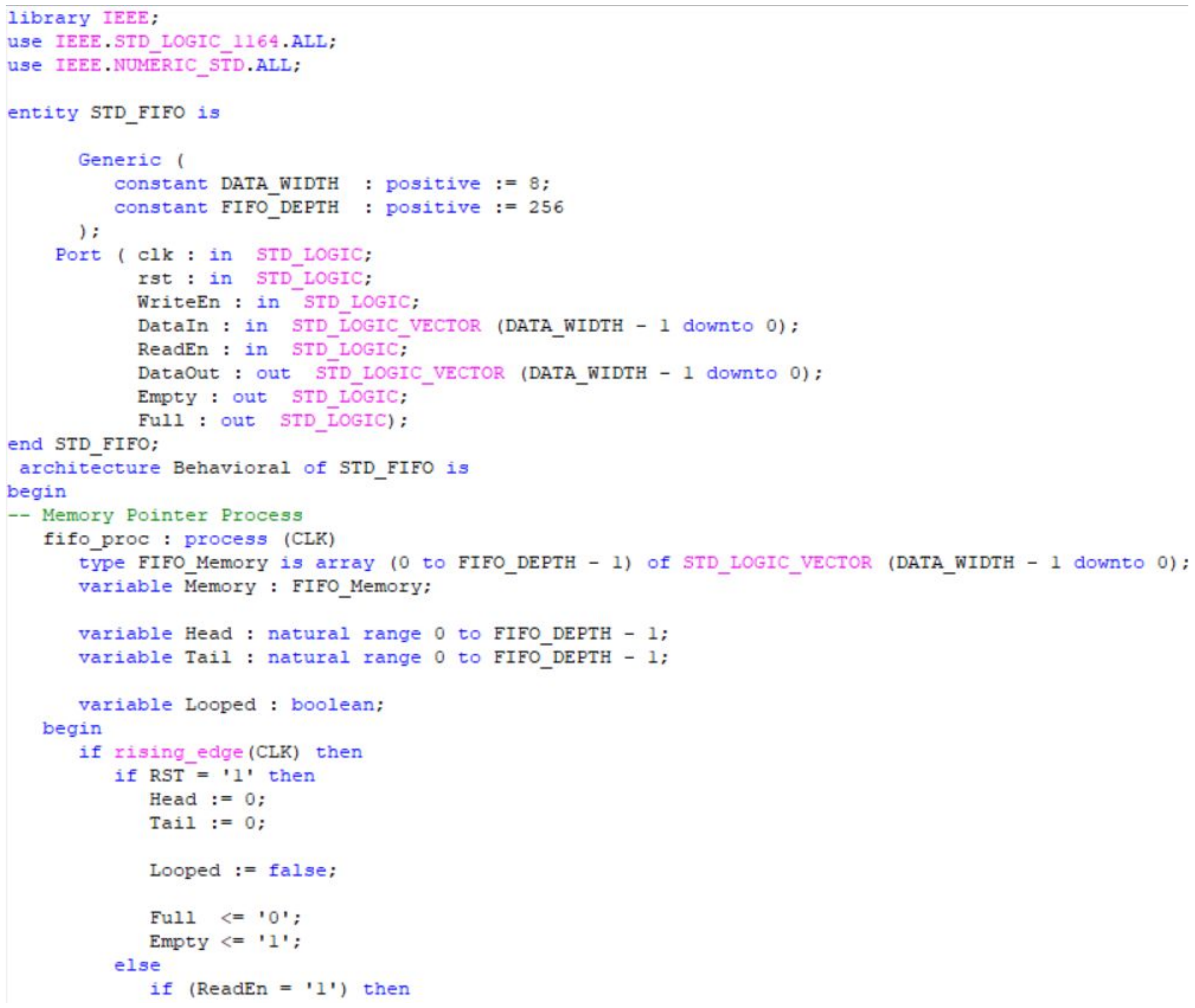
For the read process to be started **ReadEn** has to be set to 1. For a standard FIFO when you write the first byte into the FIFO nothing happens on the **DataOut** bus until the **ReadEn** signal is pulsed high for at-least one clock cycle.

Just as it goes for the Write process, where we update the Head pointer as the data is pushed, we keep the Tail pointer at the beginning since the read process was never started. When the **ReadEn** signal is pulsed high for at-least one clock cycle, we update the position of the **Tail** pointer and move it forward as and when it reads the data.

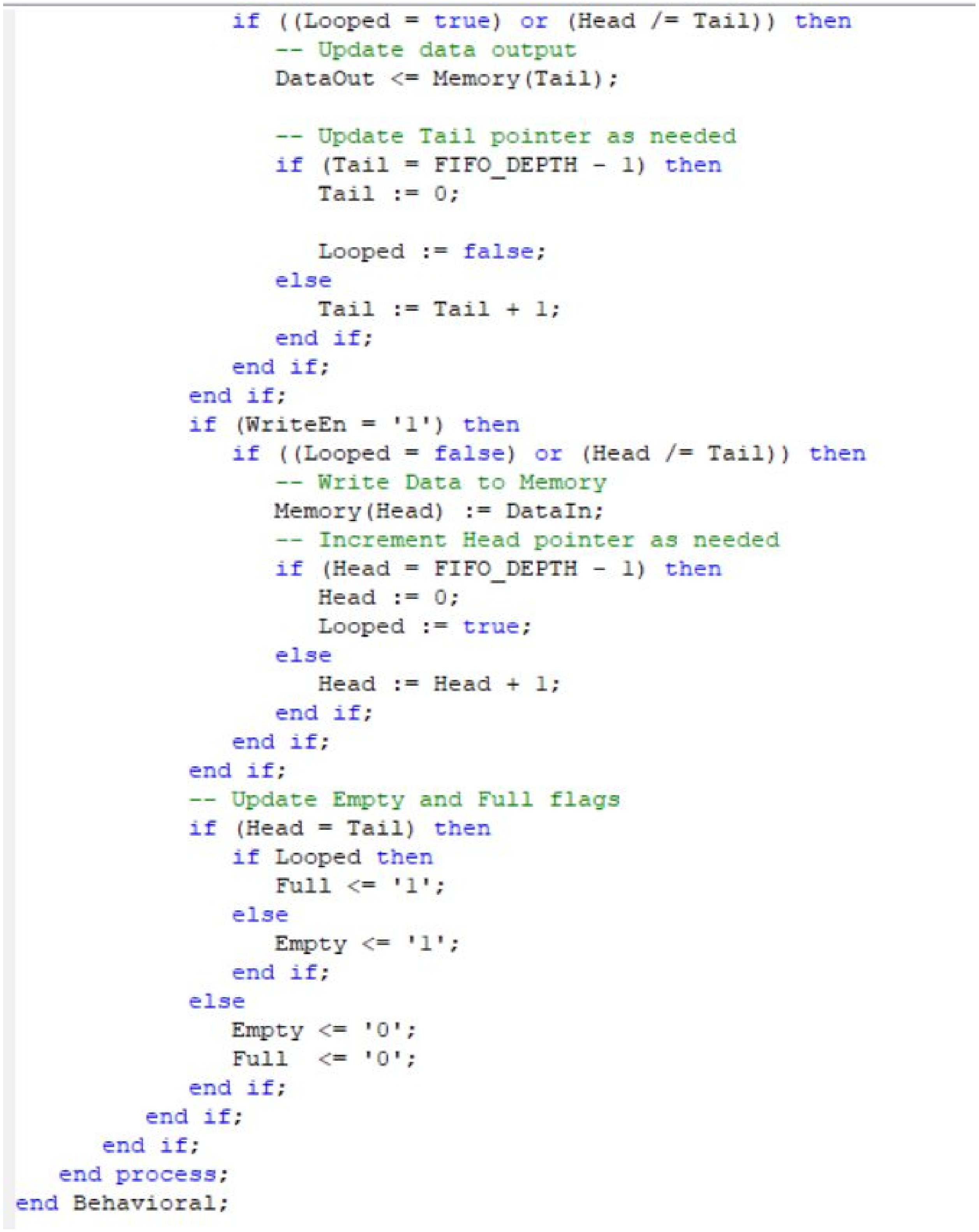
Once a byte has been written into the FIFO the **Empty** flag will go low. To read the next byte from the FIFO strobe the **ReadEn** signal high for one clock cycle and the next byte of data will be available to read on the next clock cycle. When the last byte of data is pushed onto the **DataOut** bus the **Empty** flag will go high.

Code

FIFO Memory:

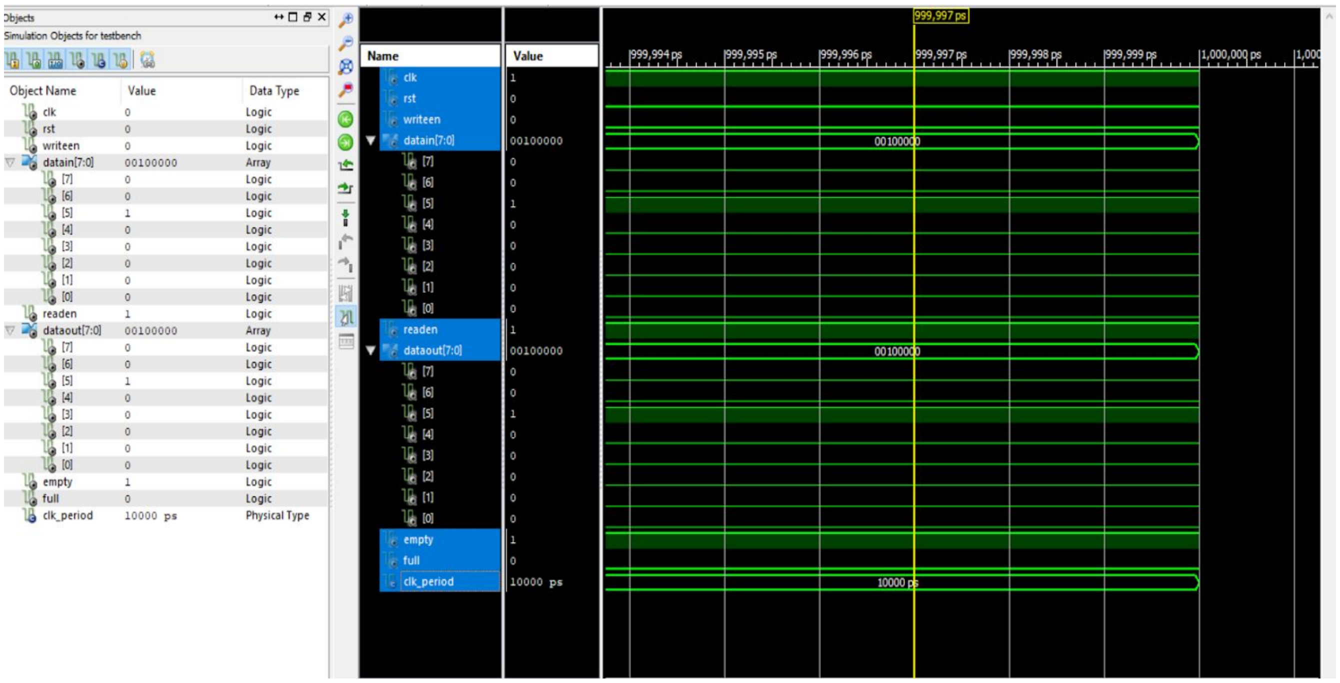


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity STD_FIFO is

      Generic (
         constant DATA_WIDTH  : positive := 8;
         constant FIFO_DEPTH  : positive := 256
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           WriteEn : in  STD_LOGIC;
           DataIn : in  STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
           ReadEn : in  STD_LOGIC;
           DataOut : out  STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
           Empty : out  STD_LOGIC;
           Full : out  STD_LOGIC);
end STD_FIFO;
 architecture Behavioral of STD_FIFO is
begin
-- Memory Pointer Process
   fifo_proc : process (CLK)
      type FIFO_Memory is array (0 to FIFO_DEPTH - 1) of STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
      variable Memory : FIFO_Memory;

      variable Head : natural range 0 to FIFO_DEPTH - 1;
      variable Tail : natural range 0 to FIFO_DEPTH - 1;

      variable Looped : boolean;
   begin
      if rising_edge(CLK) then
         if RST = '1' then
            Head := 0;
            Tail := 0;

            Looped := false;

            Full  <= '0';
            Empty <= '1';
         else
            if (ReadEn = '1') then
```



```
               if ((Looped = true) or (Head /= Tail)) then
                  -- Update data output
                  DataOut <= Memory(Tail);

                  -- Update Tail pointer as needed
                  if (Tail = FIFO_DEPTH - 1) then
                     Tail := 0;

                     Looped := false;
                  else
                     Tail := Tail + 1;
                  end if;
               end if;
            end if;
            if (WriteEn = '1') then
               if ((Looped = false) or (Head /= Tail)) then
                  -- Write Data to Memory
                  Memory(Head) := DataIn;
                  -- Increment Head pointer as needed
                  if (Head = FIFO_DEPTH - 1) then
                     Head := 0;
                     Looped := true;
                  else
                     Head := Head + 1;
                  end if;
               end if;
            end if;
            -- Update Empty and Full flags
            if (Head = Tail) then
               if Looped then
                  Full <= '1';
               else
                  Empty <= '1';
               end if;
            else
               Empty <= '0';
               Full  <= '0';
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

Output



Objects

Simulation Objects for testbench

Object Name	Value	Data Type
clk	0	Logic
rst	0	Logic
writeen	0	Logic
datain[7:0]	00100000	Array
[7]	0	Logic
[6]	0	Logic
[5]	1	Logic
[4]	0	Logic
[3]	0	Logic
[2]	0	Logic
[1]	0	Logic
[0]	0	Logic
readen	1	Logic
dataout[7:0]	00100000	Array
[7]	0	Logic
[6]	0	Logic
[5]	1	Logic
[4]	0	Logic
[3]	0	Logic
[2]	0	Logic
[1]	0	Logic
[0]	0	Logic
empty	1	Logic
full	0	Logic
clk_period	10000 ps	Physical Type

Name	Value
clk	1
rst	0
writeen	0
datain[7:0]	00100000
[7]	0
[6]	0
[5]	1
[4]	0
[3]	0
[2]	0
[1]	0
[0]	0
readen	1
dataout[7:0]	00100000
[7]	0
[6]	0
[5]	1
[4]	0
[3]	0
[2]	0
[1]	0
[0]	0
empty	1
full	0
clk_period	10000 ps

Observation

From the given code we can observe that the STD_FIFO module used gives us the proper output, following the process of how a FIFO memory works.

We push the data using the DataIn bus when the WriteEn signal is high and we receive the same data that we pushed as a result once the ReadEn signal is high, using the DataOut bus.
The data we recieve is in the same sequence that we pushed it in, in. That is, in example of the code, we push in **00100000** in the DataIn bus and we receive **00100000** in the DataOut bus.

We can conclude by saying that the First-In First-Out standard of the FIFO memory was followed by the mentioned STD_FIFO module.

References

- https://surf-vhdl.com/what-is-a-fifo/
- https://www.fpga4student.com/2017/01/vhdl-code-for-fifo-memory.html?m=1
- https://embdev.net/topic/466010
- https://github.com/DeathByLogic/HDL
- https://www.digikey.com/eewiki/pages/viewpage.action?pageId=20939499