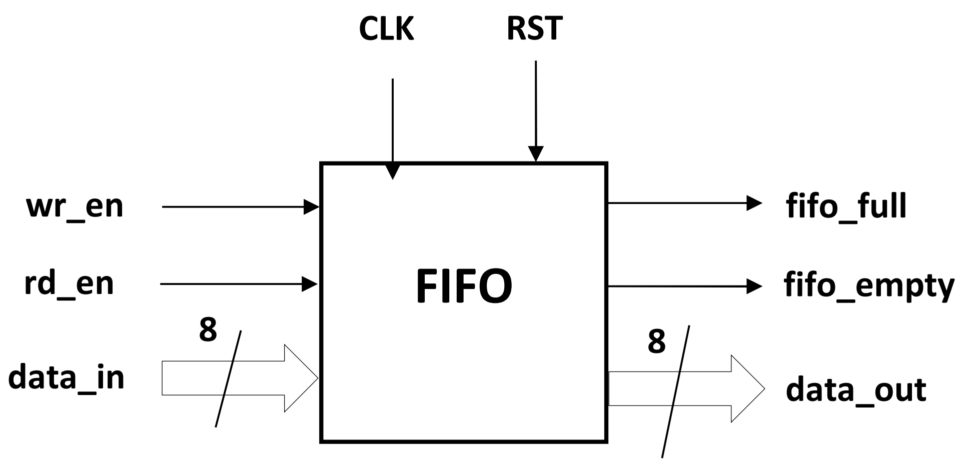
BLOCK DIAGRAM



Function Table:

CLK	RST	rd_en	wr_en	data_in	fifo_empty	fifo_full	data_out
↑	1	X	X	X	1	0	$(00)_{16}$
↑	0	0	1	$(XY)_{16}$	0	0	$(00)_{16}$
↑	0	1	0	$(XY)_{16}$	0	1	$(XY)_{16}$

VHDL MODEL / PROGRAM

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY FIFO is
generic (depth : integer := 16);
PORT (CLK : in std_logic;
        RST : in std_logic;
        rd_en : in std_logic;
        wr_en : in std_logic;
        data_in : in std_logic_vector(7 downto 0);
        data_out : out std_logic_vector(7 downto 0);
        fifo_empty : out std_logic;
        fifo_full : out std_logic
       );
END FIFO;

ARCHITECTURE FIFO_arch of FIFO is

type memory_type is array (0 to depth-1) of std_logic_vector(7 downto 0);
signal memory : memory_type :=(others => (others => '0'));
signal readptr,writeptr : integer := 0;
signal empty,full : std_logic := '0';

begin
                fifo_empty <= empty;
                fifo_full <= full;
                process(CLK,RST)
                variable num_elem : integer := 0;
                begin
                                if(RST = '1') then
                                        memory <= (others => (others=> '0'));
                                        data_out <= (others => '0');
                                        empty <= '1';
                                        full <= '0';
                                        readptr <= 0;
                                        writeptr <= 0;
                                        num_elem := 0;

                                elsif(rising_edge(CLK)) then
                                        if(rd_en = '1' and empty = '0') then  --read
                                                data_out <= memory(readptr);
                                                readptr <= readptr + 1;
                                                num_elem := num_elem-1;
                                        end if;

                                        if(wr_en ='1' and full = '0') then
                                                memory(writeptr) <= data_in;
```

```
                                                  writeptr <= writeptr + 1;
                                                  num_elem := num_elem+1;
                                          end if;

                                          if(readptr = depth-1) then
                                                  readptr <= 0;
                                          end if;

                                          if(writeptr = depth-1) then
                                                  writeptr <= 0;
                                          end if;

                                          if(num_elem = 0) then
                                                  empty <= '1';
                                          else
                                                  empty <= '0';
                                          end if;

                                          if(num_elem = depth) then
                                                  full <= '1';
                                          else
                                                  full <= '0';
                                          end if;
                                  end if;
                  end process;
  END FIFO_arch;
```

FIFO TVM

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY fifo_tb IS
END fifo_tb;

ARCHITECTURE behavior OF fifo_tb IS
   --Inputs and outputs
   signal Clk,reset,enr,enw,empty,full : std_logic := '0';
   signal data_in,data_out : std_logic_vector(7 downto 0) := (others => '0');
    --temporary signals
    signal i : integer := 0;
   -- Clock period definitions
   constant Clk_period : time := 10 ns;
    constant depth : integer := 16;  --specify depth of fifo here.

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: entity work.fifo generic map(depth => depth) PORT MAP
(clk,reset,enr,enw,data_in,data_out,empty,full);

   -- Clock process definitions
   Clk_process :process
   begin
        Clk <= '0';
        wait for Clk_period/2;
        Clk <= '1';
        wait for Clk_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
   begin
        reset <= '1';  --apply reset for one clock cycle.
        wait for clk_period;
        reset <= '0';
        wait for clk_period*3;  --wait for 3 clock periods(simply)
        enw <= '1';     enr <= '0';         --write 10 values to fifo.
      for i in 1 to 10 loop
            Data_In <= conv_std_logic_vector(i,8);
            wait for clk_period;
      end loop;
        enw <= '0';     enr <= '1';         --read 4 values from fifo.
      wait for clk_period*4;
        enw <= '0';     enr <= '0';
        wait for clk_period*10;  --wait for some clock cycles.
        enw <= '1';     enr <= '0';         --write 10 values to fifo.
    for i in 11 to 20 loop
        Data_In <= conv_std_logic_vector(i,8);
        wait for clk_period;
    end loop;
     enw <= '0';   enr <= '0';
     wait for clk_period*10;  --wait for some clock cycles.
     enw <= '0';   enr <= '1';       --read 4 values from fifo.
    wait for clk_period*4;
     enw <= '0';   enr <= '0';
     wait for clk_period;
     enw <= '0';   enr <= '1';       --read 4 values from fifo.
    wait for clk_period*8;
     enw <= '0';   enr <= '0';
     wait for clk_period;
     enw <= '0';   enr <= '1';       --read 8 values from fifo.
    wait for clk_period*4;
     enw <= '0';   enr <= '0';
     wait for clk_period;
     enw <= '0';   enr <= '1';       --read 4 values from fifo.
    wait for clk_period*4;
     enw <= '0';   enr <= '0';
     wait;
  end process;

END;
```