**FIFO CODE**

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 := 8

);

Port(

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

WriteEn : in STD\_LOGIC;

Dataln : 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

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

DataOut<= Memory(Tail);

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

Memory(Head) := Dataln;

if (Head = FIFO\_DEPTH - 1) then

Head := 0;

Looped := true;

else

Head := Head + 1;

end if;

end if;

end if;

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;

**TEST BENCH**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.NUMERIC\_STD.ALL;

entity tb\_STD\_FIFO is

end tb\_STD\_FIFO;

architecture Behavioral of tb\_STD\_FIFO is

-- Constants for the test bench

constant DATA\_WIDTH : positive := 8;

constant FIFO\_DEPTH : positive := 8;

-- Signals for the DUT

signal CLK : STD\_LOGIC := '0';

signal RST : STD\_LOGIC := '0';

signal WriteEn : STD\_LOGIC := '0';

signal Dataln : STD\_LOGIC\_VECTOR (DATA\_WIDTH - 1 downto 0) := (others => '0');

signal ReadEn : STD\_LOGIC := '0';

signal DataOut : STD\_LOGIC\_VECTOR (DATA\_WIDTH - 1 downto 0);

signal Empty : STD\_LOGIC;

signal Full : STD\_LOGIC;

-- Instantiate the DUT

component STD\_FIFO

Generic (

DATA\_WIDTH : positive;

FIFO\_DEPTH : positive

);

Port (

CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

WriteEn : in STD\_LOGIC;

Dataln : 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 component;

begin

-- Instantiate the FIFO

DUT: STD\_FIFO

Generic map (

DATA\_WIDTH => DATA\_WIDTH,

FIFO\_DEPTH => FIFO\_DEPTH

)

Port map (

CLK => CLK,

RST => RST,

WriteEn =>WriteEn,

Dataln =>Dataln,

ReadEn =>ReadEn,

DataOut =>DataOut,

Empty => Empty,

Full => Full

);

-- Clock generation

CLK\_process: process

begin

while true loop

CLK <= '0';

wait for 10 ns;

CLK <= '1';

wait for 10 ns;

end loop;

end process;

-- Test process

stim\_proc: process

begin

-- Reset the FIFO

RST <= '1';

wait for 20 ns;

RST <= '0';

wait for 20 ns;

-- Write data to FIFO

for i in 0 to 7 loop

WriteEn<= '1';

Dataln<= std\_logic\_vector(to\_unsigned(i, DATA\_WIDTH));

wait for 20 ns;

end loop;

WriteEn<= '0'; -- Stop writing

wait for 20 ns;

-- Read data from FIFO

for i in 0 to 7 loop

ReadEn<= '1';

wait for 20 ns; -- Allow time for the read operation

ReadEn<= '0';

wait for 20 ns;

end loop;

-- Attempt to read from an empty FIFO

ReadEn<= '1';

wait for 20 ns; -- Allow time for the read operation

ReadEn<= '0';

wait for 20 ns;

-- Write more data to check full condition

for i in 8 to 15 loop

WriteEn<= '1';

Dataln<= std\_logic\_vector(to\_unsigned(i, DATA\_WIDTH));

wait for 20 ns;

end loop;

WriteEn<= '0'; -- Stop writing

wait for 20 ns;

-- Read all data to check FIFO behavior

for i in 0 to 7 loop

ReadEn<= '1';

wait for 20 ns; -- Allow time for the read operation

ReadEn<= '0';

wait for 20 ns;

end loop;

-- Reset the FIFO again to see if it clears correctly

RST <= '1';

wait for 20 ns;

RST <= '0';

wait for 20 ns;

-- Finalize simulation

wait;

end process;

end Behavioral;