(7:0)

full

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use lEEE.NUMERIC\_STD.ALL;

MAIN VHDL MODEL ( MVM )

entity fifo is

generic (depth: integer:= 16); --depth of fifo port ( clk : in std\_logic; reset : in std\_logic;

enr: in std \_logic; --enable read,should be '0' when not in use. enw: in std\_logic; --enable write,should be 'O' when not in use. data\_in : in std\_logic\_vector (7 downto 0); --input data data\_out: out std\_logic\_vector(7 downto 0); -output data fifo\_empty: out std \_logic;

-set as'1' when the queue is empty

fifo\_full: out std\_logic --set as '1' when the queue is full

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')); -memory for queue. signal readptr,writeptr: integer:= 0; -read and write pointers. signal empty,full: std\_logic := '0';

begin

fifo\_empty <= empty; fifo\_full <= full;

process(Clk, reset)

* ﻿-this is the number of elements stored in fifo at a time.
* ﻿-this variable is used to decide whether the fifo is empty or full. variable num\_elem: integer := 0;

begin

if(reset = '1') then

- for i in 0 to depth-1 loop

memory(i)<=(others=>'0');

end loop;

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(enr = '1' and empty = '0) then -read

data\_out <= memory(readptr);

readptr <= readptr + 1; num\_elem:= num\_elem-1;

end if;

if(enw ='1' and full = '0') then -write

memory(writeptr) <= data\_in;

writeptr <= writept + 1; num\_elem := num\_elem+1;

end if;

-rolling over of the indices.

if(readptr = depth-1) then

- resetting read pointer.

readptr <= 0; end if;

if(writeptr = depth-1) then

-resetting write pointer.

writeptr <= 0; end it;

-setting empty and full flags.

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;

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.

DEP

istinguishe mpetent und Teleco

1 stimul

al inne ring p

ptabili

|  |  |
| --- | --- |
| 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 cik\_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;

rOg