Stack

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  entity stack is  port(  clk, reset: in std\_logic;  full, empty: out std\_logic;  push, pop: in std\_logic;  w\_addr, r\_addr: out std\_logic\_vector(1 downto 0)  );    end stack;  architecture stack\_arch of stack is  constant N: natural := 1;  signal addr\_reg, addr\_next: unsigned(N downto 0);  signal full\_flag, empty\_flag: std\_logic;    begin  process(clk,reset)  begin  if (reset='1') then  addr\_reg <= (others=>'0');  elsif(clk'event and clk='1') then  addr\_reg <= addr\_next;  end if;  end process;  addr\_next <= addr\_reg+1 when push='1' and full\_flag='0' else  addr\_reg-1 when pop='1' and empty\_flag='0' else  addr\_reg;  full\_flag <= '1' when addr\_reg="11" else  '0';  empty\_flag <= '1' when addr\_reg="00" else  '0';  w\_addr <= std\_logic\_vector(addr\_reg+1);  r\_addr <= std\_logic\_vector(addr\_reg);  full <= full\_flag;  empty <= empty\_flag;  end stack\_arch; |

FIFO 2048x8

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  entity fifo2048\_8 is  port(  clk, reset: in std\_logic;  full, empty: out std\_logic;  wr, rd: in std\_logic;  w\_data: in std\_logic\_vector(7 downto 0);  r\_data: out std\_logic\_vector(7 downto 0)  );    end fifo2048\_8;  architecture fifo\_arch of fifo2048\_8 is  component RAMB16\_S9\_S9  port(  DOA: out std\_logic\_vector(7 downto 0);  DOB: out std\_logic\_vector(7 downto 0);  DOPA: out std\_logic\_vector(0 downto 0);  DOPB: out std\_logic\_vector(0 downto 0);  ADDRA: in std\_logic\_vector(10 downto 0);  ADDRB: in std\_logic\_vector(10 downto 0);  CLKA: in std\_ulogic;  CLKB: in std\_ulogic;  DIA: in std\_logic\_vector(7 downto 0);  DIB: in std\_logic\_vector(7 downto 0);  DIPA: in std\_logic\_vector(0 downto 0);  DIPB: in std\_logic\_vector(0 downto 0);  ENA: in std\_ulogic;  ENB: in std\_ulogic;  SSRA: in std\_ulogic;  SSRB: in std\_ulogic;  WEA: in std\_ulogic;  WEB: in std\_ulogic  );  end component;    constant N: natural := 11;  signal w\_ptr\_reg, w\_ptr\_next: unsigned(N downto 0);  signal r\_ptr\_reg, r\_ptr\_next: unsigned(N downto 0);  signal full\_flag, empty\_flag: std\_logic;  signal addra, addrb: std\_logic\_vector(N downto 0) := (others=>'0');  signal w\_en: std\_logic;  signal clk\_en: std\_logic := '1';    signal dib: std\_logic\_vector(7 downto 0) := (others=>'0');  signal dipa,dipb: std\_logic\_vector(0 downto 0) := (others=>'0');  signal web: std\_ulogic := '0';      begin  myram: RAMB16\_S9\_S9  port map(DOA=>open, DOB=>r\_data, DOPA=>open, DOPB=>open,  ADDRA=>addra, ADDRB=>addrb,  CLKA=>clk, CLKB=>clk, DIA=>w\_data, DIB=>dib,  DIPA=>dipa, DIPB=>dipb, ENA=>clk\_en, ENB=>clk\_en,  SSRA=>reset, SSRB=>reset, WEA=>w\_en, WEB=>web  );    process(clk,reset)  begin  if (reset='1') then  w\_ptr\_reg <= (others=>'0');  r\_ptr\_reg <= (others=>'0');  elsif(clk'event and clk='1') then  w\_ptr\_reg <= w\_ptr\_next;  r\_ptr\_reg <= r\_ptr\_next;  end if;  end process;  -- write next logic  w\_en <= '1' when wr='1' and full\_flag='0' else  '0';  w\_ptr\_next <= w\_ptr\_reg + 1 when wr='1' and full\_flag='0' else  w\_ptr\_reg;  full\_flag <= '1' when r\_ptr\_reg(N) /= w\_ptr\_reg(N) and  r\_ptr\_reg(N-1 downto 0) = w\_ptr\_reg(N-1 downto 0) else  '0';  addra <= std\_logic\_vector(w\_ptr\_reg(N-1 downto 0));  -- read next logic  r\_ptr\_next <= r\_ptr\_reg + 1 when rd='1' and empty\_flag='0' else  r\_ptr\_reg;  empty\_flag <= '1' when r\_ptr\_reg = w\_ptr\_reg else  '0';  addrb <= std\_logic\_vector(r\_ptr\_reg(N-1 downto 0));    -- output logic  full <= full\_flag;  empty <= empty\_flag;  end fifo\_arch; |

Inferred FIFO 2048x8

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity fifoinfer2048\_8 is  port(  clk, reset: in std\_logic;  full, empty: out std\_logic;  wr, rd: in std\_logic;  w\_data: in std\_logic\_vector(7 downto 0);  r\_data: out std\_logic\_vector(7 downto 0)  );    end fifoinfer2048\_8;  architecture fifo\_arch of fifoinfer2048\_8 is    type ram\_type is array(2047 downto 0) of std\_logic\_vector(7 downto 0);  signal Ram: ram\_type := (others=>(others=>'0'));  constant N: natural := 11;  signal w\_ptr\_reg, w\_ptr\_next: unsigned(N downto 0);  signal r\_ptr\_reg, r\_ptr\_next: unsigned(N downto 0);  signal full\_flag, empty\_flag: std\_logic;  signal addra, addrb: std\_logic\_vector(N downto 0) := (others=>'0');  signal w\_en: std\_logic;  signal read\_a: std\_logic\_vector(7 downto 0) := (others=>'0');  begin  process(clk)  begin  if(clk'event and clk='1') then  if(w\_en='1') then  RAM(conv\_integer(addra)) <= w\_data;  end if;  end if;  read\_a <= RAM(conv\_integer(addrb));  end process;  r\_data <= read\_a;    process(clk,reset)  begin  if (reset='1') then  w\_ptr\_reg <= (others=>'0');  r\_ptr\_reg <= (others=>'0');  elsif(clk'event and clk='1') then  w\_ptr\_reg <= w\_ptr\_next;  r\_ptr\_reg <= r\_ptr\_next;  end if;  end process;  w\_en <= '1' when wr='1' and full\_flag='0' else  '0';  w\_ptr\_next <= w\_ptr\_reg + 1 when wr='1' and full\_flag='0' else  w\_ptr\_reg;  full\_flag <= '1' when r\_ptr\_reg(N) /= w\_ptr\_reg(N) and  r\_ptr\_reg(N-1 downto 0) = w\_ptr\_reg(N-1 downto 0) else  '0';  addra <= std\_logic\_vector(w\_ptr\_reg(N-1 downto 0));  r\_ptr\_next <= r\_ptr\_reg + 1 when rd='1' and empty\_flag='0' else  r\_ptr\_reg;  empty\_flag <= '1' when r\_ptr\_reg = w\_ptr\_reg else  '0';  addrb <= std\_logic\_vector(r\_ptr\_reg(N-1 downto 0));    -- output logic  full <= full\_flag;  empty <= empty\_flag;  end fifo\_arch; |