First-In First-Out (FIFO) Control Logic VHDL Modeling Example

A common problem in ASIC design is constructing a FIFO from a RAM by designing the control logic to generate the address (ADD) and write enable (WE) to the RAM so that the first data word written into the RAM is also the first data word retrieved from the RAM. Therefore, we want to write a parameterized VHDL model for an N-bit FIFO. The VHDL model will implement the logic required to make a pipelined RAM operate as the FIFO. As discussed in class, the RAM is assumed to have separate data inputs and outputs, an N-bit address bus (ADD) and an active high write enable (WE). The inputs to the FIFO/Stack logic include PUSH, POP, INIT (all active high) in addition to the rising edge triggered CLK input. The FIFO logic will not only supply the address and write enable to the RAM, but will also supply active high flags for FULL, EMPTY, NOPOP and NOPUSH conditions. The NOPOP and NOPUSH flags indicate that no FIFO read or write operation was executed due to one of the following conditions:

1. simultaneous assertion of both PUSH and POP - the POP takes priority => NOPUSH
2. assertion of PUSH when the FIFO is full => NOPUSH
3. assertion of POP when the FIFO is empty => NOPOP
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

design FIFO_LOGIC is
  generic (N: integer := 3);
  port (clk,push,pop,init: in std_logic;
        add: out std_logic_vector(N-1 downto 0);
        full,empty,we,nopush,nopop: buffer std_logic);
end FIFO_LOGIC;

architecture RTL of FIFO_LOGIC is
signal wp, rp, lastop: std_logic_vector(N-1 downto 0);
signal lastop: std_logic;
begin
  sync: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (init = '1') then  -- initialization --
        wp <= (others => '0');
        rp <= (others => '0');
        lastop <= '0';
      elseif (pop = '1' and empty = '0') then  -- pop --
        rp <= rp + 1;
        lastop <= '0';
      elseif (push = '1' and full = '0') then  -- push --
        wp <= wp + 1;
        lastop <= '1';
      end if;  -- otherwise all F's hold their value --
    end if;
  end process sync;

  comb: process (push,pop,wp,rp,lastop,full,empty) begin
  -- full and empty flags --
  if (rp = wp) then
    if (lastop = '1') then
      full <= '1';
      empty <= '0';
    else
      full <= '0';
      empty <= '1';
    end if;
  else
    full <= '0';
  end if;
end process comb;

C. Stroud Dept. of ECE, Auburn Univ.
empty <= '0';
end if;
-- address, write enable and nopush/nopop logic --
if (pop = '0' and push = '0') then
  -- no operation --
  add <= rptr;
  we <= '0';
  nopush <= '0';
nopop <= '0';
elsif (pop = '0' and push = '1') then
  -- push only --
  add <= wptr;
  nopop <= '0';
  if (full = '0') then
    -- valid write condition --
    we <= '1';
    nopush <= '0';
  else
    -- no write condition --
    we <= '0';
    nopush <= '1';
  end if;
elsif (pop = '1' and push = '0') then
  -- pop only --
  add <= rptr;
  nopush <= '0';
  we <= '0';
  if (empty = '0') then
    -- valid read condition --
    nopop <= '0';
  else
    -- no read condition --
    nopop <= '1';
  end if;
else
  -- push and pop at same time --
  add <= rptr;
  we <= '0';
  nopush <= '1';
  nopop <= '0';
end if;
end process comb;
end architecture RTL;

Note: If you synthesize this model and simulate the synthesized circuit, you may see cases where the value for “lastop” is inverted compared to the values indicated in the VHDL model. Since this is an internal node and not a primary output of the circuit, then a given synthesis tool may take the liberty of inverting the node values to help minimize the resultant logic utilization. The important thing is that the circuit works properly from the standpoint of the I/O behavior (so you wouldn’t know that lastop was inverted if you didn’t monitor that node during simulation). Synthesis tools may also eliminate internal nodes of the VHDL model as well as introduce new internal nodes all in the name of design optimization.