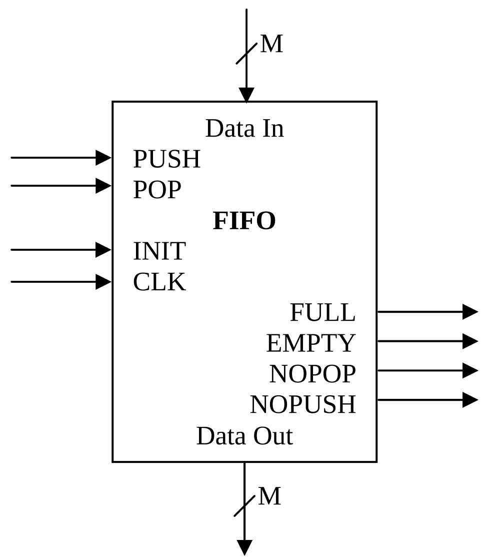
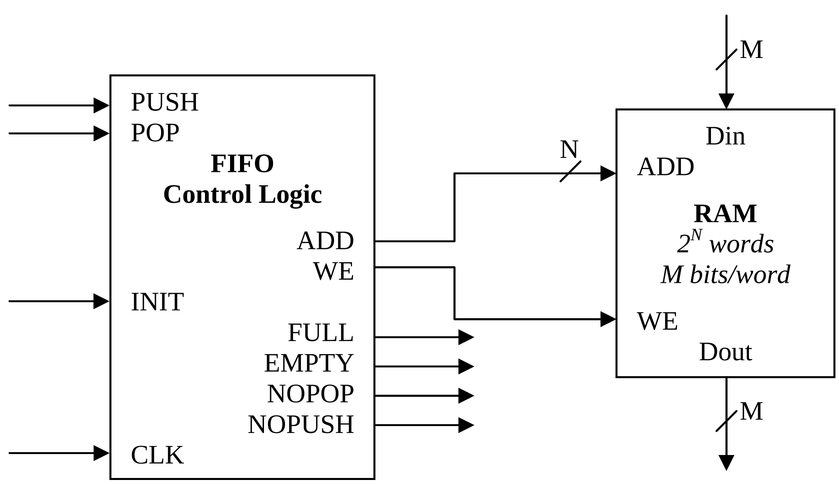
SEQUENTIAL LOGIC MODELING EXAMPLE
USING 2-PROCESS MODELING STYLE

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

A common problem in 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 a FIFO (using one process for sequential logic operations and one process for combinational logic operations). The VHDL model will implement the logic required to make a pipelined RAM operate as the FIFO. In this case, 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





SEQUENTIAL LOGIC MODELING EXAMPLE
USING 2-PROCESS MODELING STYLE

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity 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 entity FIFO_LOGIC;

architecture RTL of FIFO_LOGIC is
        signal WPTR, RPTR: 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 --
                        WPTR <= (others => '0');
                        RPTR <= (others => '0');
                        LASTOP <= '0';
                elsif (POP = '1' and EMPTY = '0') then  -- pop --
                        RPTR <= RPTR + 1;
                        LASTOP <= '0';
                elsif (PUSH = '1' and FULL = '0') then  -- push --
                        WPTR <= WPTR + 1;
                        LASTOP <= '1';
                end if;         -- otherwise all Fs hold their value --
        end if;
end process SYNC;

COMB: process (PUSH, POP, WPTR, RPTR, LASTOP, FULL, EMPTY) begin
-- full and empty flags --
        if (RPTR = WPTR) then
                if (LASTOP = '1') then
                        FULL <= '1';
                        EMPTY <= '0';
                else
                        FULL <= '0';
                        EMPTY <= '1';
                end if;
        else
                FULL <= '0';
                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
                        NOPOP <= '1';                   -- no red condition --
                end if;
        else                            -- push and pop at same time –
                if (EMPTY = ‘0’) then           -- valid pop --
                        ADD <= RPTR;
                        WE <= '0';
                        NOPUSH <= '1';
                        NOPOP <= '0';
                else
                        ADD <= wptr;
                        WE <= ‘1’;
                        NOPUSH <= ‘0’;
                        NOPOP <= ‘1’;
                end if;
        end if;
end process COMB;
end architecture RTL;
```

With a VHDL model for the RAM and FIFO control logic complete, we can generate a top-level hierarchical model of the complete FIFO:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity FIFO is
        generic (N: integer := 3; -- number of address bits for 2**N address locations
                M: integer := 5);  -- number of data bits to/from FIFO
        port (CLK, PUSH, POP, INIT: in std_logic;
                DIN: in std_logic_vector(N-1 downto 0);
                DOUT: out std_logic_vector(N-1 downto 0);
                FULL, EMPTY, NOPUSH, NOPOP: out std_logic);
end entity FIFO;

architecture TOP_HIER of FIFO is
signal WE: std_logic;
signal A: std_logic_vector(N-1 downto 0);

component FIFO_LOGIC is
        generic (N: integer); -- number of address bits
        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 component FIFO_LOGIC;

component RAM is
        generic (K, W: integer)    -- number of address and data bits
        port (WR: in std_logic;    -- active high write enable
                ADDR: in std _ logic _vector (W-1 downto 0);        -- RAM address
                DIN: in std _ logic _ vector (K-1 downto 0);        -- write data
                DOUT: out std _ logic _ vector (K-1 downto 0));     -- read data
end component RAM;

begin

-- example of component instantiation using positional notation
FL: FIFO_LOGIC generic map (N)
        port map (CLK, PUSH, POP, INIT, A, FULL, EMPTY, WE, NOPUSH, NOPOP);

-- example of component instantiation using keyword notation
R: RAM generic map (W => N, K => M)
        port map (DIN => DIN, ADDR => A, WR => WE, DOUT => DOUT);

end architecture TOP_HIER;
```