VHDL EXAMPLES, 2017

COUNTERS
SHIFTERS
RAMs
STATE MACHINE
DELAY LINES

1. 4-bit Unsigned Up Counter with Asynchronous Clear

The following table shows pin definitions for a 4-bit unsigned up counter with asynchronous clear.

IO Pins	Description
C	Positive-Edge Clock
CLR	Asynchronous Clear (active High)
Q[3:0]	Data Output

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port(C, CLR : in  std_logic;
       Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
      process (C, CLR)
        begin
          if (CLR='1') then
            tmp <= "0000";
          elsif (C'event and C='1') then
            tmp <= tmp + 1;
          end if;
      end process;
      Q <= tmp;
end archi;
```

2. 4-bit Unsigned Up/Down counter with Asynchronous Clear

The following table shows pin definitions for a 4-bit unsigned up/down counter with asynchronous clear.

IO Pins	Description
C	Positive-Edge Clock
CLR	Asynchronous Clear (active High)
UP_DOWN	up/down count mode selector
Q[3:0]	Data Output

VHDL Code

Following is the VHDL code for a 4-bit unsigned up/down counter with asynchronous clear.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port(C, CLR, UP_DOWN : in std_logic;
       Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (C, CLR)
      begin
        if (CLR='1') then
          tmp <= "0000";
        elsif (C'event and C='1') then
          if (UP_DOWN='1') then
            tmp <= tmp + 1;
          else
            tmp <= tmp - 1;
          end if;
        end if;
    end process;
    Q <= tmp;
end archi;
```

3. 4-bit Unsigned Up Counter with Asynchronous Load from Primary Input

The following table shows pin definitions for a 4-bit unsigned up counter with asynchronous load from primary input.

IO Pins	Description
C	Positive-Edge Clock

ALOAD	Asynchronous Load (active High)
D[3:0]	Data Input
Q[3:0]	Data Output

VHDL Code

Following is the VHDL code for a 4-bit unsigned up counter with asynchronous load from primary input.

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

entity counter is
  port(C, ALOAD : in  std_logic;
       D : in std_logic_vector(3 downto 0);
       Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (C, ALOAD, D)
      begin
        if (ALOAD='1') then
          tmp <= D;
        elsif (C'event and C='1') then
          tmp <= tmp + 1;
        end if;

    end process;
    Q <= tmp;
end archi;
```

4. 8-bit Shift-Left Register with Positive-Edge Clock, Asynchronous Clear, Serial In, and Serial Out

Note Because this example includes an asynchronous clear, XST will not infer SRL16.

The following table shows pin definitions for an 8-bit shift-left register with a positive-edge clock, asynchronous clear, serial in, and serial out.

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In

CLR	Asynchronous Clear (active High)
SO	Serial Output

VHDL Code

Following is the VHDL code for an 8-bit shift-left register with a positive-edge clock, asynchronous clear, serial in, and serial out.

```
library ieee;
use ieee.std_logic_1164.all;

entity shift is
  port(C, SI, CLR : in std_logic;
       SO : out std_logic);
end shift;
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C, CLR)
      begin
        if (CLR='1') then
          tmp <= (others => '0');
        elsif (C'event and C='1') then
          tmp <= tmp(6 downto 0) & SI;
        end if;
    end process;
    SO <= tmp(7);
end archi;
```

5. 8-bit Shift-Left/Shift-Right Register with Positive-Edge Clock, Serial In, and Parallel Out

Note For this example XST will not infer SRL16.

The following table shows pin definitions for an 8-bit shift-left/shift-right register with a positive-edge clock, serial in, and serial out.

IO Pins	Description
C	Positive-Edge Clock
SI	Serial In
LEFT_RIGHT	Left/right shift mode selector
PO[7:0]	Parallel Output

VHDL Code

Following is the VHDL code for an 8-bit shift-left/shift-right register with a positive-edge clock, serial in, and serial out.

```
library ieee;
use ieee.std_logic_1164.all;

entity shift is
port(C, SI, LEFT_RIGHT : in std_logic;
      PO : out std_logic_vector(7 downto 0));
end shift;
architecture archi of shift is
  signal tmp: std_logic_vector(7 downto 0);
  begin
    process (C)
      begin
        if (C'event and C='1') then
          if (LEFT_RIGHT='0') then
            tmp <= tmp(6 downto 0) & SI;
          else
            tmp <= SI & tmp(7 downto 1);
          end if;
        end if;
    end process;
    PO <= tmp;
end archi;
```

6. Logical shifter

The following table shows pin descriptions for a logical shifter.

IO pins	Description
D[7:0]	Data Input
SEL	shift distance selector
SO[7:0]	Data Output

VHDL

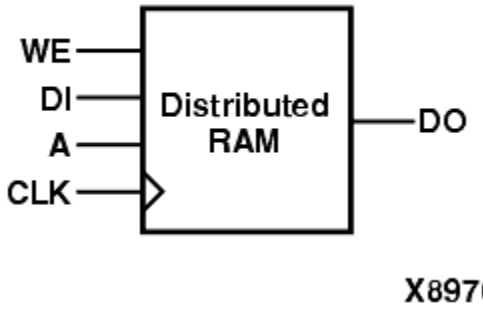
Following is the VHDL code for a logical shifter.

```
library ieee;
 use ieee.std_logic_1164.all;
 use ieee.numeric_std.all;

 entity lshift is
  port(DI : in unsigned(7 downto 0);
       SEL : in unsigned(1 downto 0);
        SO : out unsigned(7 downto 0));
 end lshift;
```

```
architecture archi of lshift is
  begin
    with SEL select
      SO <= DI when "00",
            DI sll 1 when "01",
            DI sll 2 when "10",
            DI sll 3 when others;
end archi;
```

7. Single-Port RAM with Asynchronous Read

The following descriptions are directly mappable onto *distributed RAM only*.



The following table shows pin descriptions for a single-port RAM with asynchronous read.

IO Pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
a	Read/Write Address
di	Data Input
do	Data Output

VHDL

Following is the VHDL code for a single-port RAM with asynchronous read.

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

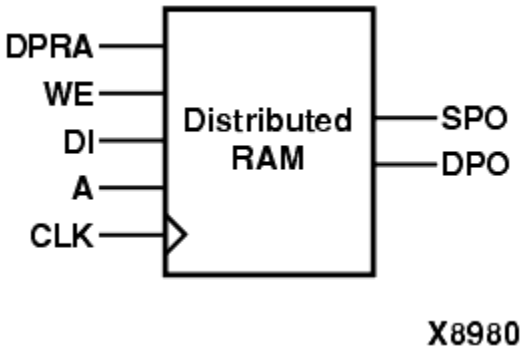
```
entity raminfr is
  port (clk : in std_logic;
        we  : in std_logic;
        a   : in std_logic_vector(4 downto 0);
        di  : in std_logic_vector(3 downto 0);
        do  : out std_logic_vector(3 downto 0));
end raminfr;
architecture syn of raminfr is
  type ram_type is array (31 downto 0)
        of std_logic_vector (3 downto 0);
  signal RAM : ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(a)) <= di;
      end if;
    end if;
  end process;
  do <= RAM(conv_integer(a));
end syn;
```

```
--------------------------------------------------------------------------------
-- File : ram_simple.vhd

--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ram_simple IS
PORT (
SIGNAL data : IN std_logic_vector(7 DOWNTO 0);
SIGNAL address : IN std_logic_vector(4 DOWNTO 0);
SIGNAL we, inclock, outclock : IN std_logic;
SIGNAL q : OUT std_logic_vector(7 DOWNTO 0));


END ram_simple;


ARCHITECTURE fe2 OF ram_simple IS

TYPE mem_type IS ARRAY ( 31 DOWNTO 0) OF std_logic_vector (7 DOWNTO 0);
SIGNAL mem : mem_type;
SIGNAL address_int : unsigned(4 DOWNTO 0);

BEGIN -- ex2
```

```
l0 : PROCESS (inclock,outclock, we, address)

BEGIN -- PROCESS
IF (inclock = '1' AND inclock'event) THEN
address_int <= unsigned(address);
IF we = '1' THEN
mem(To_integer(unsigned(address))) <= data;
END IF;
END IF;
IF (outclock = '1' AND outclock'event) THEN
q <= mem(to_integer(address_int));
END IF;

END PROCESS;

END fe2;
```

8. Dual-Port RAM with Asynchronous Read

The following example shows where the two output ports are used. It is directly mappable onto *Distributed RAM only*.



The following table shows pin descriptions for a dual-port RAM with asynchronous read.

IO pins	Description
clk	Positive-Edge Clock
we	Synchronous Write Enable (active High)
a	Write Address/Primary Read Address
dpra	Dual Read Address

di	Data Input
spo	Primary Output Port
dpo	Dual Output Port

VHDL

Following is the VHDL code for a dual-port RAM with asynchronous read.

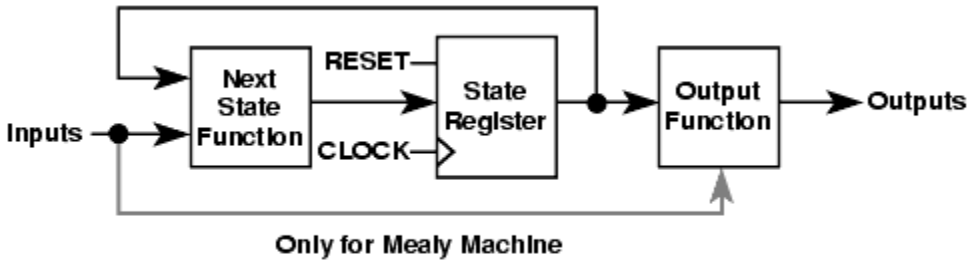
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
  port (clk  : in std_logic;
        we   : in std_logic;
        a    : in std_logic_vector(4 downto 0);
        dpra : in std_logic_vector(4 downto 0);
        di   : in std_logic_vector(3 downto 0);
        spo  : out std_logic_vector(3 downto 0);
        dpo  : out std_logic_vector(3 downto 0));
end raminfr;
architecture syn of raminfr is
  type ram_type is array (31 downto 0)
    of std_logic_vector (3 downto 0);
  signal RAM : ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(a)) <= di;
      end if;
    end if;
  end process;
  spo <= RAM(conv_integer(a));
  dpo <= RAM(conv_integer(dpra));
 end syn;
```

9. State Machine

XST proposes a large set of templates to describe Finite State Machines (FSMs). By default, XST tries to recognize FSMs from VHDL/Verilog code, and apply several state encoding techniques (it can re-encode the user's initial encoding) to get better performance or less area. However, you can disable FSM extraction using a **FSM_extract** design constraint.

Please note that XST can handle only synchronous state machines.

There are many ways to describe FSMs. A traditional FSM representation incorporates Mealy and Moore machines, as in the following figure:

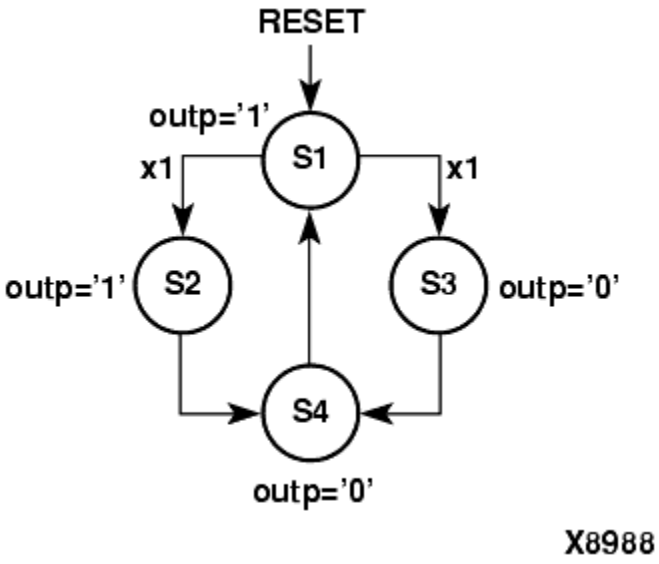


For HDL, process (VHDL) and always blocks (Verilog) are the most suitable ways for describing FSMs. (For description convenience Xilinx uses "process" to refer to both: VHDL processes and Verilog always blocks).

You may have several processes (1, 2 or 3) in your description, depending upon how you consider and decompose the different parts of the preceding model. Following is an example of the Moore Machine with Asynchronous Reset, "RESET".

- 4 states: s1, s2, s3, s4
- 5 transitions
- 1 input: "x1"
- 1 output: "outp"

This model is represented by the following bubble diagram:



Related Constraints

Related constraints are:

- **FSM_extract**
- **FSM_encoding**
- **FSM_fftype**
- **ENUM_encoding**

FSM with 1 Process

Please note, in this example output signal "outp" is a *register*.

VHDL

Following is the VHDL code for an FSM with a single process.

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

entity fsm is
  port ( clk, reset, x1 : IN std_logic;
                  outp : OUT std_logic);
end entity;
architecture beh1 of fsm is
  type state_type is (s1,s2,s3,s4);
  signal state: state_type ;
begin
  process (clk,reset)
  begin
    if (reset ='1') then
      state <=s1; outp<='1';
    elsif (clk='1' and clk'event) then
      case state is
        when s1 => if x1='1' then state <= s2;
                    else          state <= s3;
                    end if;
                    outp <= '1';
        when s2 => state <= s4; outp <= '1';
        when s3 => state <= s4; outp <= '0';
        when s4 => state <= s1; outp <= '0';
      end case;
    end if;
  end process;
end beh1;
```

10. DELAY LINE AND ARITHMETIC OPERATION

The function shows simple filter, produce an average value of two samples. Design VHDL code?