How To Read VHDL

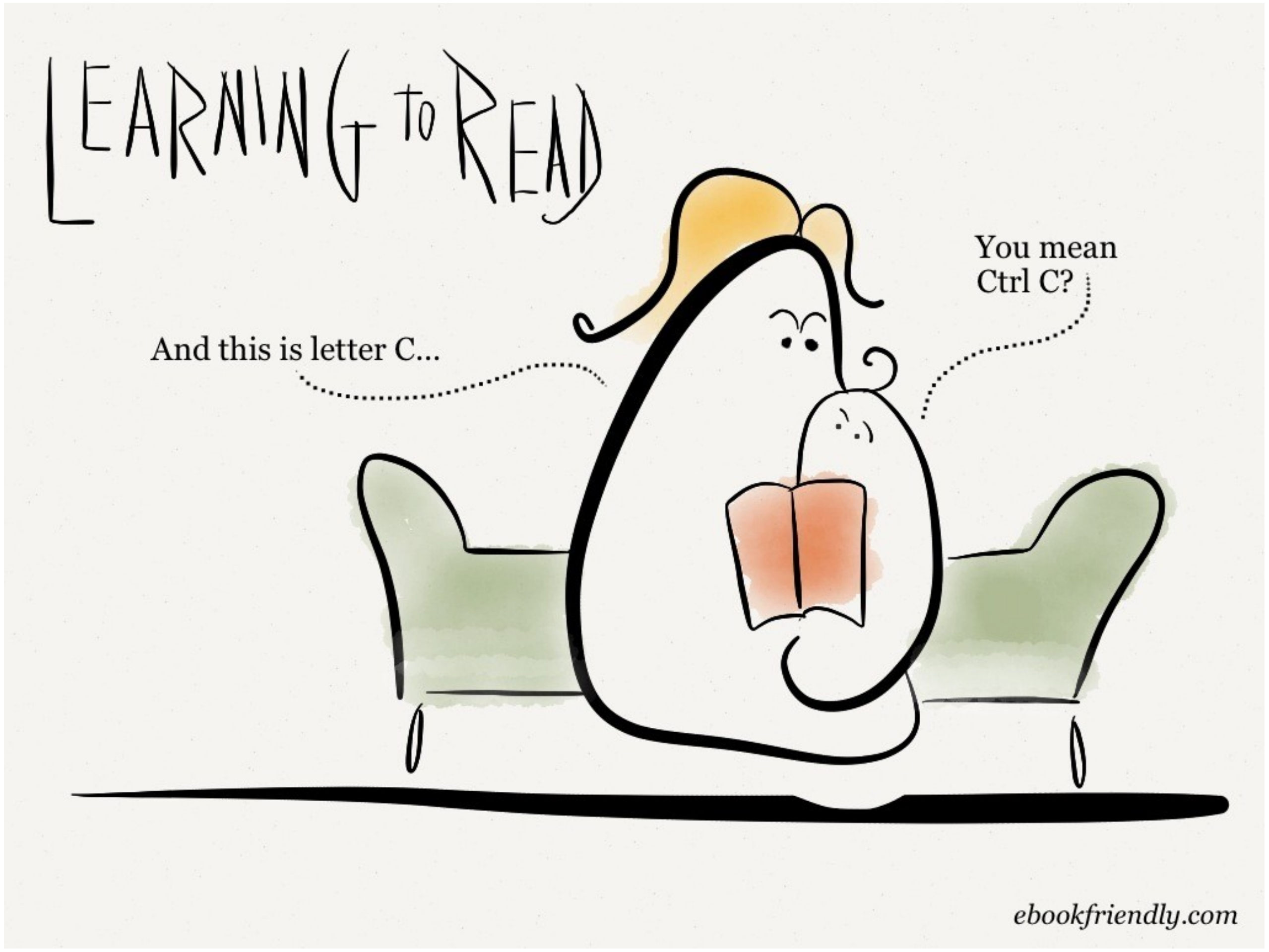


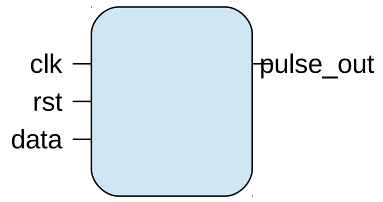
Image credit: Piotr Kowalczyk

by Iain Waugh

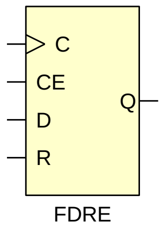
Key VHDL Structure

```
----------------------------------------
-- The outline of VHDL files
----------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity outline is
  generic (
    -- Some synthesis-time parameters
    );
  port(
    -- Interface definition
    );
end outline;

architecture outline_rtl of outline is
  -- Internal signals

begin  -- outline_rtl

  -- Core functionality

end outline_rtl;
```

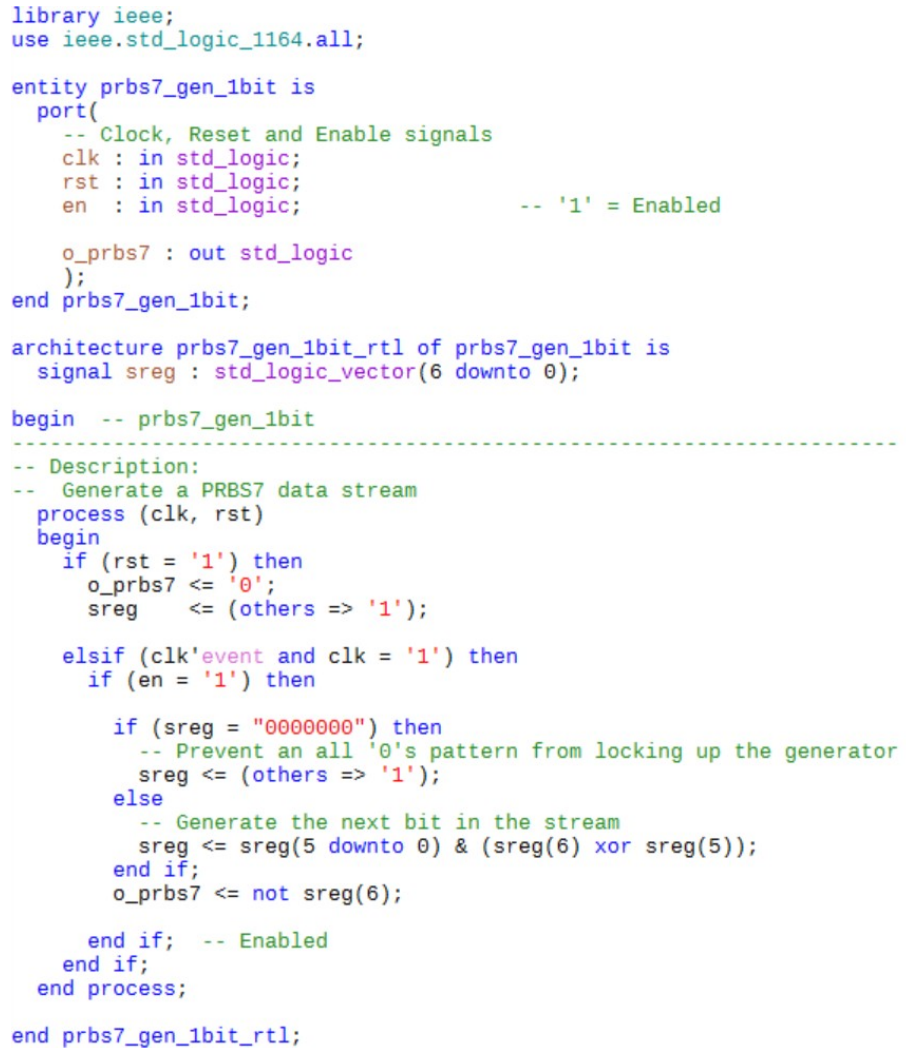
- Header
- Included libraries
- Generics and I/O ports



- Architecture (the guts)
- May be in a separate file



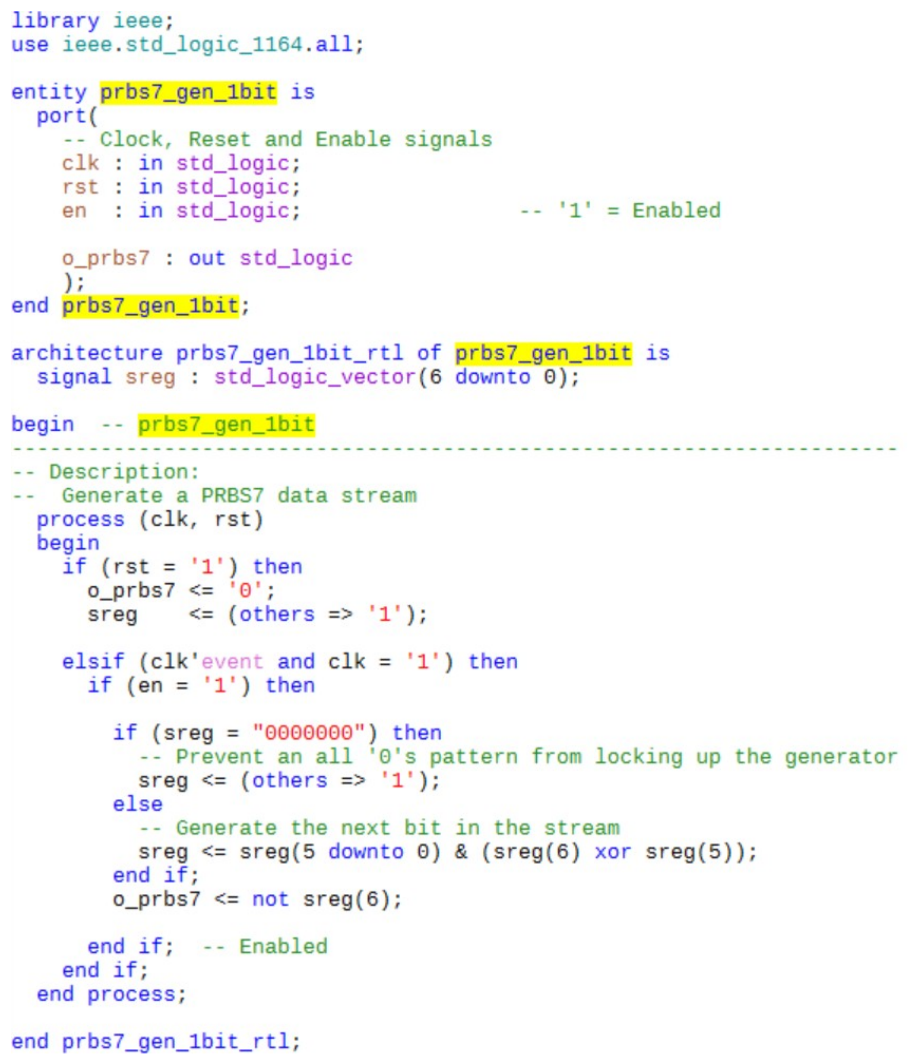
How To Read VHDL



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

entity prbs7_gen_1bit is
  port(
    -- Clock, Reset and Enable signals
    clk : in std_logic;
    rst : in std_logic;
    en  : in std_logic;                    -- '1' = Enabled

    o_prbs7 : out std_logic
    );
end prbs7_gen_1bit;

architecture prbs7_gen_1bit_rtl of prbs7_gen_1bit is
  signal sreg : std_logic_vector(6 downto 0);

begin  -- prbs7_gen_1bit
----------------------------------------------------------------------
-- Description:
--   Generate a PRBS7 data stream
  process (clk, rst)
  begin
    if (rst = '1') then
      o_prbs7 <= '0';
      sreg    <= (others => '1');

    elsif (clk'event and clk = '1') then
      if (en = '1') then

        if (sreg = "0000000") then
          -- Prevent an all '0's pattern from locking up the generator
          sreg <= (others => '1');
        else
          -- Generate the next bit in the stream
          sreg <= sreg(5 downto 0) & (sreg(6) xor sreg(5));
        end if;
        o_prbs7 <= not sreg(6);

      end if;  -- Enabled
    end if;
  end process;

end prbs7_gen_1bit_rtl;
```

- Here's a PRBS 7 generator
- It generates a random-looking sequence of 0's and 1's
- (see Xilinx XAPP052 for more)

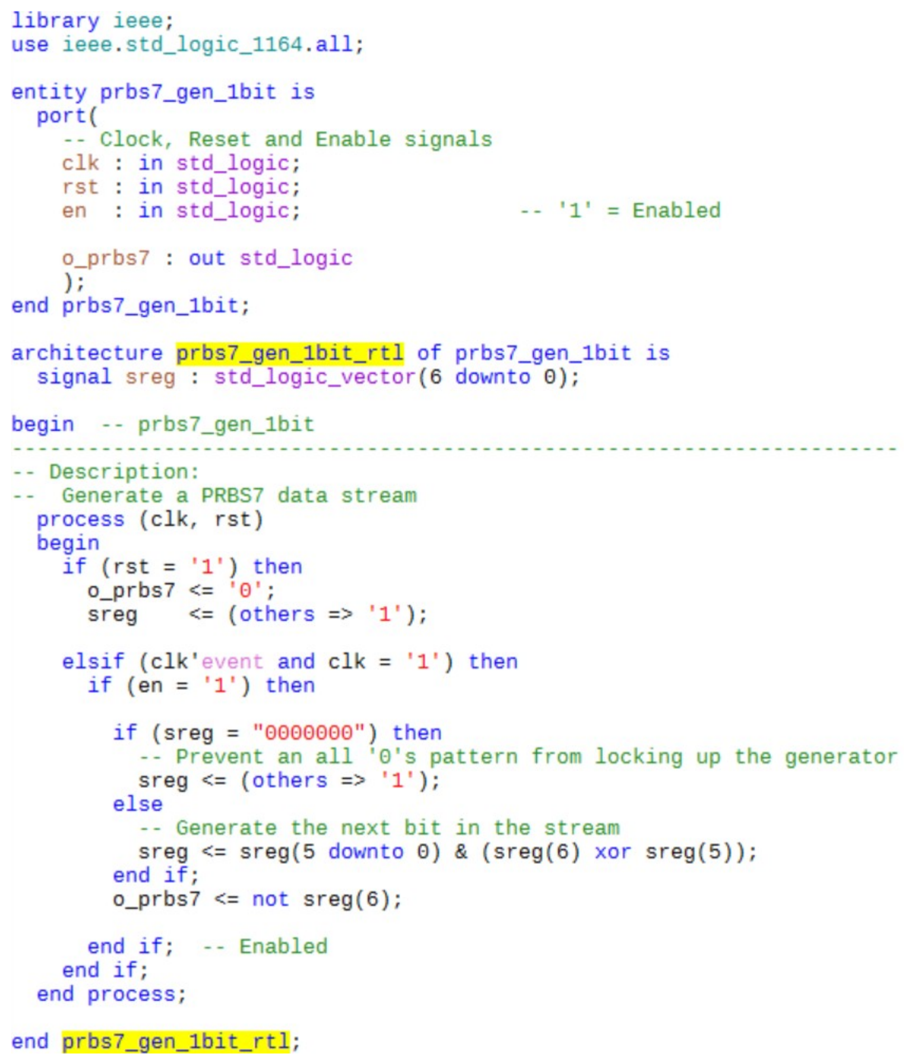
Here’s the Entity Name



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

entity prbs7_gen_1bit is
  port(
    -- Clock, Reset and Enable signals
    clk : in std_logic;
    rst : in std_logic;
    en  : in std_logic;                    -- '1' = Enabled

    o_prbs7 : out std_logic
    );
end prbs7_gen_1bit;

architecture prbs7_gen_1bit_rtl of prbs7_gen_1bit is
  signal sreg : std_logic_vector(6 downto 0);

begin  -- prbs7_gen_1bit
-----------------------------------------------------------------------
-- Description:
--  Generate a PRBS7 data stream
  process (clk, rst)
  begin
    if (rst = '1') then
      o_prbs7 <= '0';
      sreg    <= (others => '1');

    elsif (clk'event and clk = '1') then
      if (en = '1') then

        if (sreg = "0000000") then
          -- Prevent an all '0's pattern from locking up the generator
          sreg <= (others => '1');
        else
          -- Generate the next bit in the stream
          sreg <= sreg(5 downto 0) & (sreg(6) xor sreg(5));
        end if;
        o_prbs7 <= not sreg(6);

      end if;  -- Enabled
    end if;
  end process;

end prbs7_gen_1bit_rtl;
```

- With luck, it’s the same as the filename

Here's the Architecture Name



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

entity prbs7_gen_1bit is
  port(
    -- Clock, Reset and Enable signals
    clk : in std_logic;
    rst : in std_logic;
    en  : in std_logic;                   -- '1' = Enabled

    o_prbs7 : out std_logic
    );
end prbs7_gen_1bit;

architecture prbs7_gen_1bit_rtl of prbs7_gen_1bit is
  signal sreg : std_logic_vector(6 downto 0);

begin  -- prbs7_gen_1bit
----------------------------------------------------------------------
-- Description:
--   Generate a PRBS7 data stream
  process (clk, rst)
  begin
    if (rst = '1') then
      o_prbs7 <= '0';
      sreg    <= (others => '1');

    elsif (clk'event and clk = '1') then
      if (en = '1') then

        if (sreg = "0000000") then
          -- Prevent an all '0's pattern from locking up the generator
          sreg <= (others => '1');
        else
          -- Generate the next bit in the stream
          sreg <= sreg(5 downto 0) & (sreg(6) xor sreg(5));
        end if;
        o_prbs7 <= not sreg(6);

      end if;  -- Enabled
    end if;
  end process;

end prbs7_gen_1bit_rtl;
```

- Some people just write 'rtl', 'arch' or 'beh'

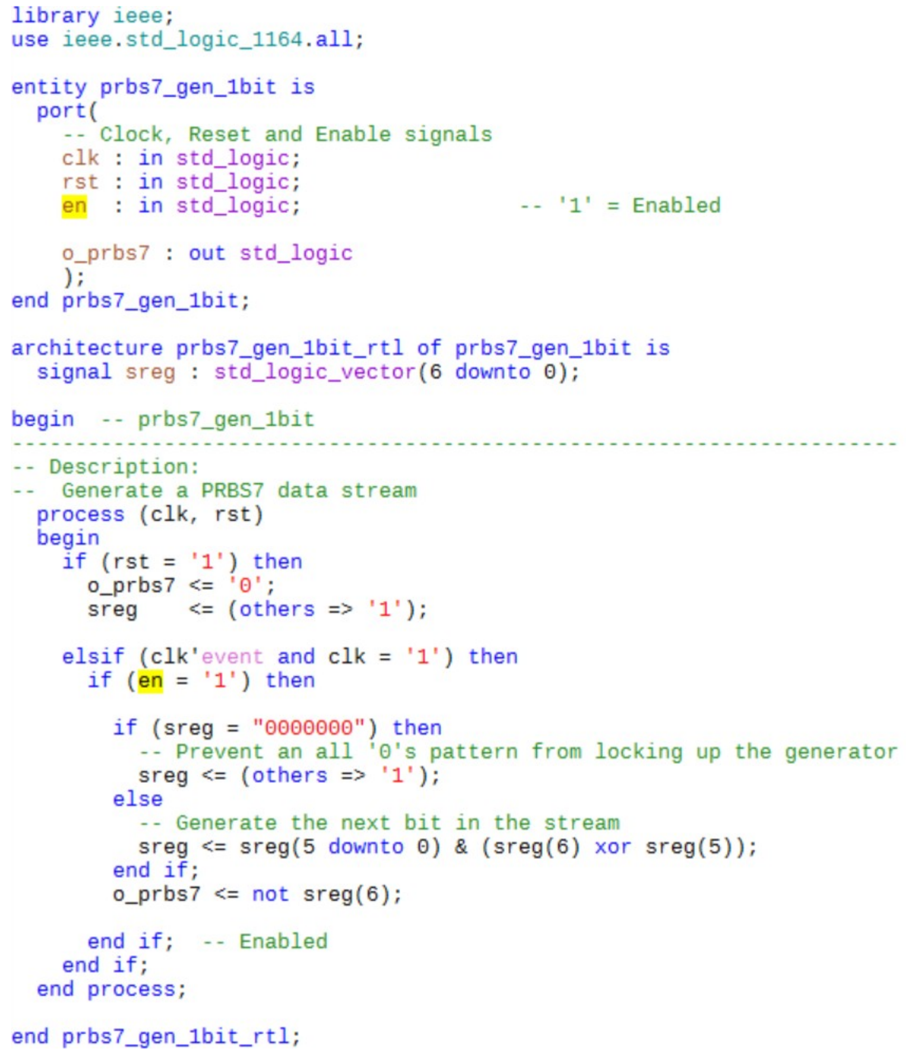
  'rtl' = functional

  'arch' = connectivity

  'beh' = sim model

  (these are very, very loose conventions)

You Can Trace The Inputs...

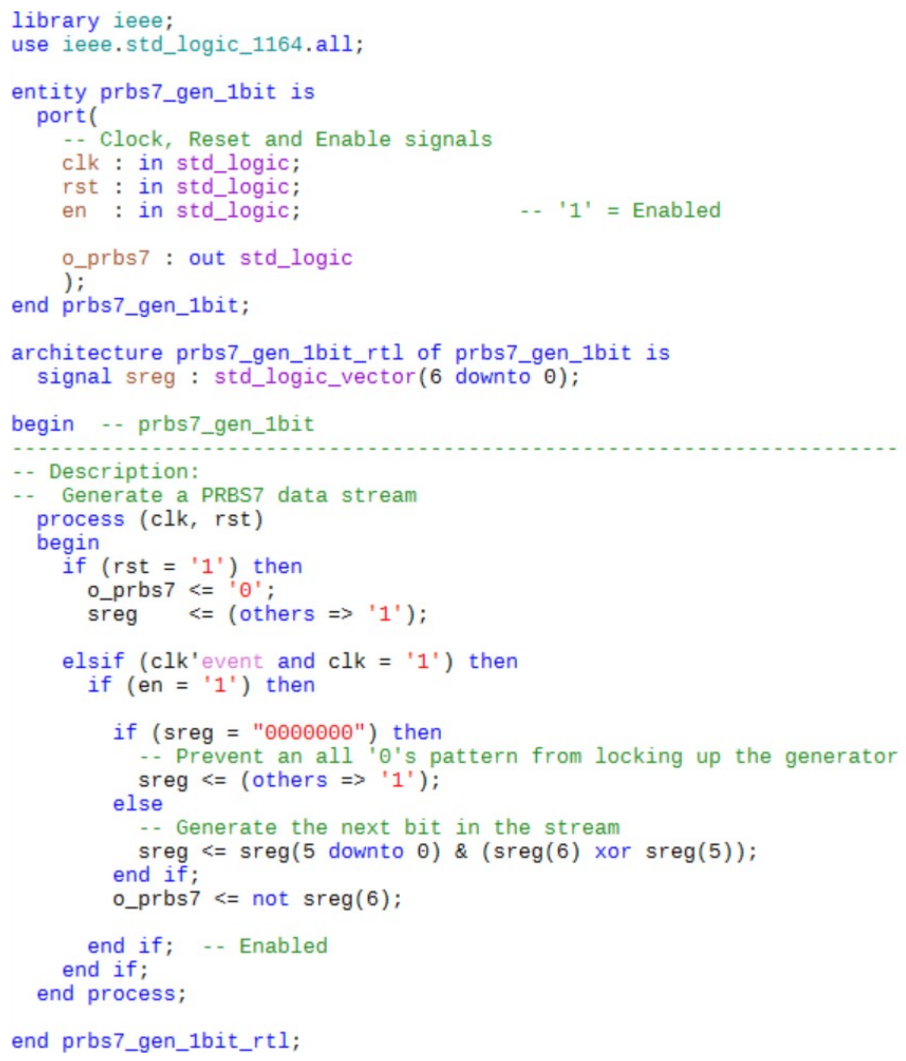


```
library ieee;
use ieee.std_logic_1164.all;

entity prbs7_gen_1bit is
  port(
    -- Clock, Reset and Enable signals
    clk : in std_logic;
    rst : in std_logic;
    en  : in std_logic;                 -- '1' = Enabled

    o_prbs7 : out std_logic
    );
end prbs7_gen_1bit;

architecture prbs7_gen_1bit_rtl of prbs7_gen_1bit is
  signal sreg : std_logic_vector(6 downto 0);

begin  -- prbs7_gen_1bit
-----------------------------------------------------------------------
-- Description:
--  Generate a PRBS7 data stream
  process (clk, rst)
  begin
    if (rst = '1') then
      o_prbs7 <= '0';
      sreg    <= (others => '1');

    elsif (clk'event and clk = '1') then
      if (en = '1') then

        if (sreg = "0000000") then
          -- Prevent an all '0's pattern from locking up the generator
          sreg <= (others => '1');
        else
          -- Generate the next bit in the stream
          sreg <= sreg(5 downto 0) & (sreg(6) xor sreg(5));
        end if;
        o_prbs7 <= not sreg(6);

      end if;  -- Enabled
    end if;
  end process;

end prbs7_gen_1bit_rtl;
```

Or Trace The Outputs

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

entity prbs7_gen_1bit is
  port(
    -- Clock, Reset and Enable signals
    clk : in std_logic;
    rst : in std_logic;
    en  : in std_logic;                  -- '1' = Enabled

    o_prbs7 : out std_logic
    );
end prbs7_gen_1bit;

architecture prbs7_gen_1bit_rtl of prbs7_gen_1bit is
  signal sreg : std_logic_vector(6 downto 0);

begin  -- prbs7_gen_1bit
-----------------------------------------------------------------------
-- Description:
--  Generate a PRBS7 data stream
  process (clk, rst)
  begin
    if (rst = '1') then
      o_prbs7 <= '0';
      sreg    <= (others => '1');

    elsif (clk'event and clk = '1') then
      if (en = '1') then

        if (sreg = "0000000") then
          -- Prevent an all '0's pattern from locking up the generator
          sreg <= (others => '1');
        else
          -- Generate the next bit in the stream
          sreg <= sreg(5 downto 0) & (sreg(6) xor sreg(5));
        end if;
        o_prbs7 <= not sreg(6);

      end if;  -- Enabled
    end if;
  end process;

end prbs7_gen_1bit_rtl;
```

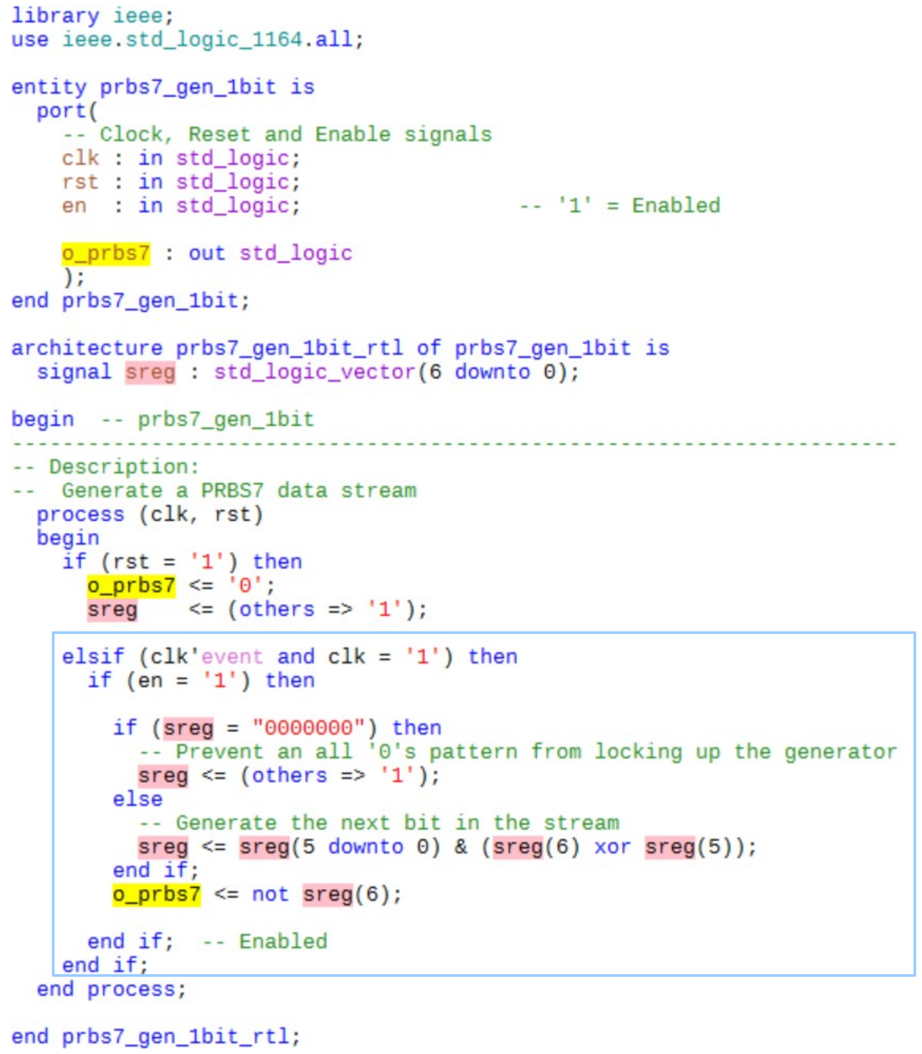
Synthesise It In Your Head



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

entity prbs7_gen_1bit is
  port(
    -- Clock, Reset and Enable signals
    clk : in std_logic;
    rst : in std_logic;
    en  : in std_logic;               -- '1' = Enabled

    o_prbs7 : out std_logic
    );
end prbs7_gen_1bit;

architecture prbs7_gen_1bit_rtl of prbs7_gen_1bit is
  signal sreg : std_logic_vector(6 downto 0);

begin  -- prbs7_gen_1bit
----------------------------------------------------------------------
-- Description:
--   Generate a PRBS7 data stream
  process (clk, rst)
  begin
    if (rst = '1') then
      o_prbs7 <= '0';
      sreg    <= (others => '1');

    elsif (clk'event and clk = '1') then
      if (en = '1') then

        if (sreg = "0000000") then
          -- Prevent an all '0's pattern from locking up the generator
          sreg <= (others => '1');
        else
          -- Generate the next bit in the stream
          sreg <= sreg(5 downto 0) & (sreg(6) xor sreg(5));
        end if;
        o_prbs7 <= not sreg(6);

      end if;  -- Enabled
    end if;
  end process;

end prbs7_gen_1bit_rtl;
```

- Being able to do this makes you a better HDL coder
- Think in terms of the physical hardware blocks, not software functions like “if”, “class”, “procedure”, etc.

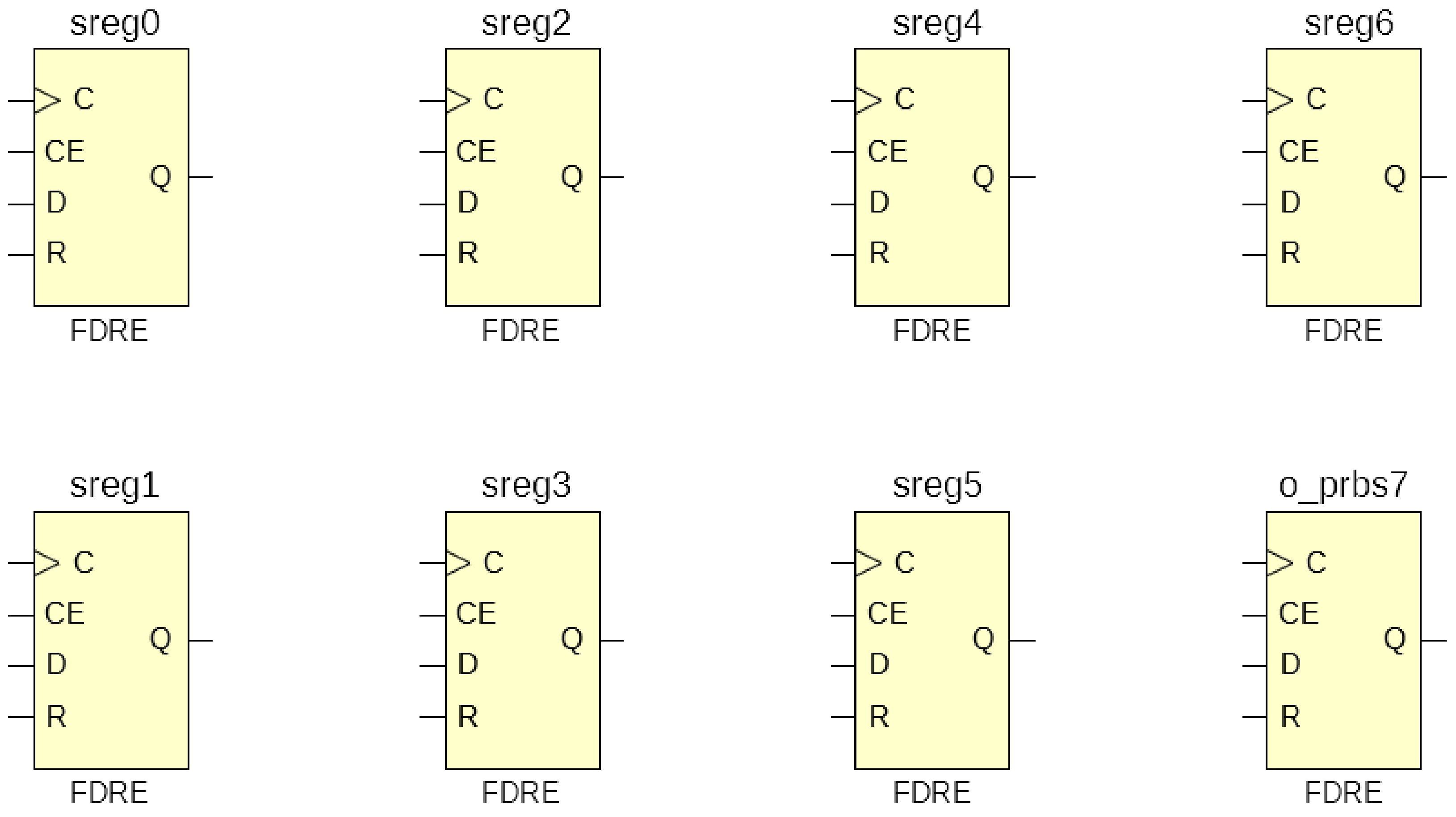
Synthesise It In Your Head



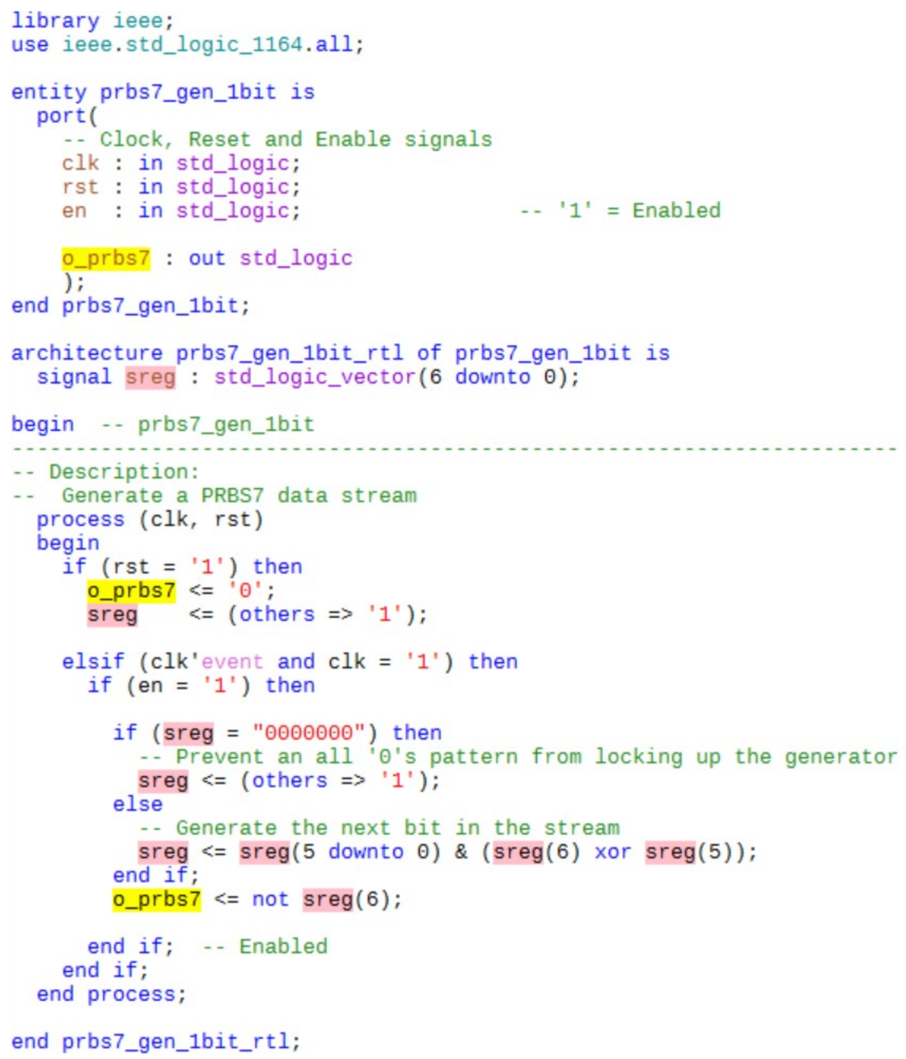
```
library ieee;
use ieee.std_logic_1164.all;

entity prbs7_gen_1bit is
  port(
    -- Clock, Reset and Enable signals
    clk : in std_logic;
    rst : in std_logic;
    en  : in std_logic;                    -- '1' = Enabled

    o_prbs7 : out std_logic
    );
end prbs7_gen_1bit;

architecture prbs7_gen_1bit_rtl of prbs7_gen_1bit is
  signal sreg : std_logic_vector(6 downto 0);

begin  -- prbs7_gen_1bit
----------------------------------------------------------------------
-- Description:
--   Generate a PRBS7 data stream
  process (clk, rst)
  begin
    if (rst = '1') then
      o_prbs7 <= '0';
      sreg    <= (others => '1');

    elsif (clk'event and clk = '1') then
      if (en = '1') then

        if (sreg = "0000000") then
          -- Prevent an all '0's pattern from locking up the generator
          sreg <= (others => '1');
        else
          -- Generate the next bit in the stream
          sreg <= sreg(5 downto 0) & (sreg(6) xor sreg(5));
        end if;
        o_prbs7 <= not sreg(6);

      end if;  -- Enabled
    end if;
  end process;

end prbs7_gen_1bit_rtl;
```

- Each bit of each clocked thing gets a flip-flop (FF)
- Clocked things are to the left of the “<=” operator (in a clocked process)
- Everything that is clocked is in the blue box
- Count them

You Should Expect These FF’s



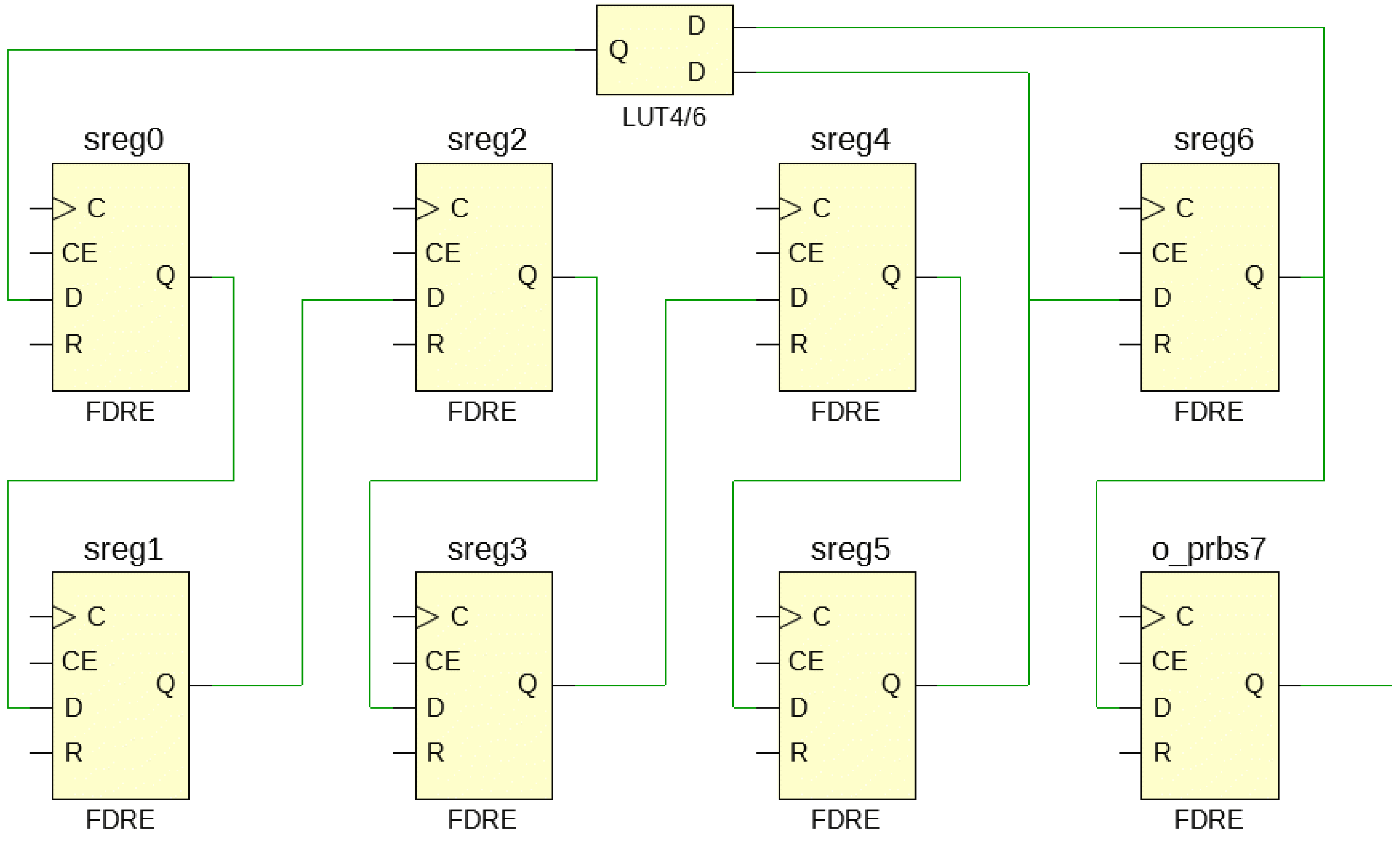
Synthesise It In Your Head



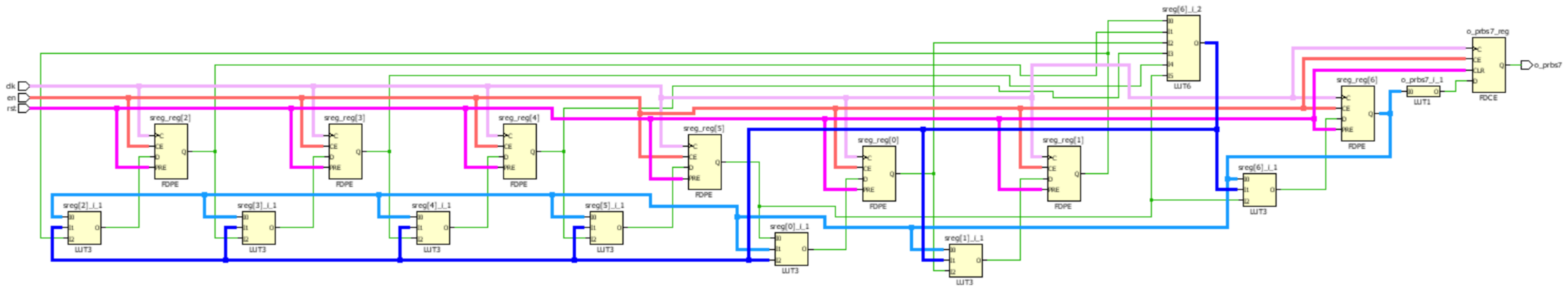
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity prbs7_gen_1bit is
  port(
    -- Clock, Reset and Enable signals
    clk : in std_logic;
    rst : in std_logic;
    en  : in std_logic;              -- '1' = Enabled

    o_prbs7 : out std_logic
    );
end prbs7_gen_1bit;

architecture prbs7_gen_1bit_rtl of prbs7_gen_1bit is
  signal sreg : std_logic_vector(6 downto 0);

begin  -- prbs7_gen_1bit
-----------------------------------------------------------------------
-- Description:
--   Generate a PRBS7 data stream
  process (clk, rst)
  begin
    if (rst = '1') then
      o_prbs7 <= '0';
      sreg    <= (others => '1');

    elsif (clk'event and clk = '1') then
      if (en = '1') then

        if (sreg = "0000000") then
          -- Prevent an all '0's pattern from locking up the generator
          sreg <= (others => '1');
        else
          -- Generate the next bit in the stream
          sreg <= sreg(5 downto 0) & (sreg(6) xor sreg(5));
        end if;
        o_prbs7 <= not sreg(6);

      end if;  -- Enabled
    end if;
  end process;

end prbs7_gen_1bit_rtl;
```

- Next, look at what affects each clocked thing.
- What affects ‘o_prbs7’?
  - = ‘sreg(6)’, ‘en’, ‘clk’, ‘rst’
- What affects ‘sreg(6)’?
  - = ‘sreg(5)’, ‘en’, ‘clk’, ‘rst’
- etc.
- What affects ‘sreg(0)’?
  - = ‘sreg(6)’, ‘sreg(5)’, ‘en’, ‘clk’, ‘rst’

Expected Result (Just Datapath)



Actual Result (Xilinx)



- Oops, forgot the “all zero’s” check (dark blue)
  `if (sreg = ”0000000”) then`
- Got the correct number of FF’s, though