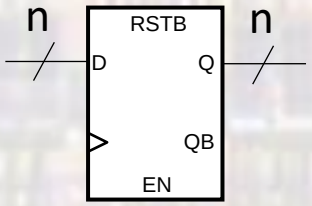
Registers - HDL

Last updated 1/18/21

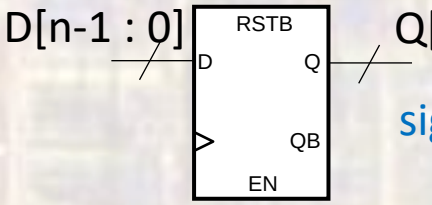
Registers - HDL

- Register
  - Collection of n Flip-Flops
  - Configured in parallel
  - Common clock, set/reset, enable
  - Stores an n-bit state variable

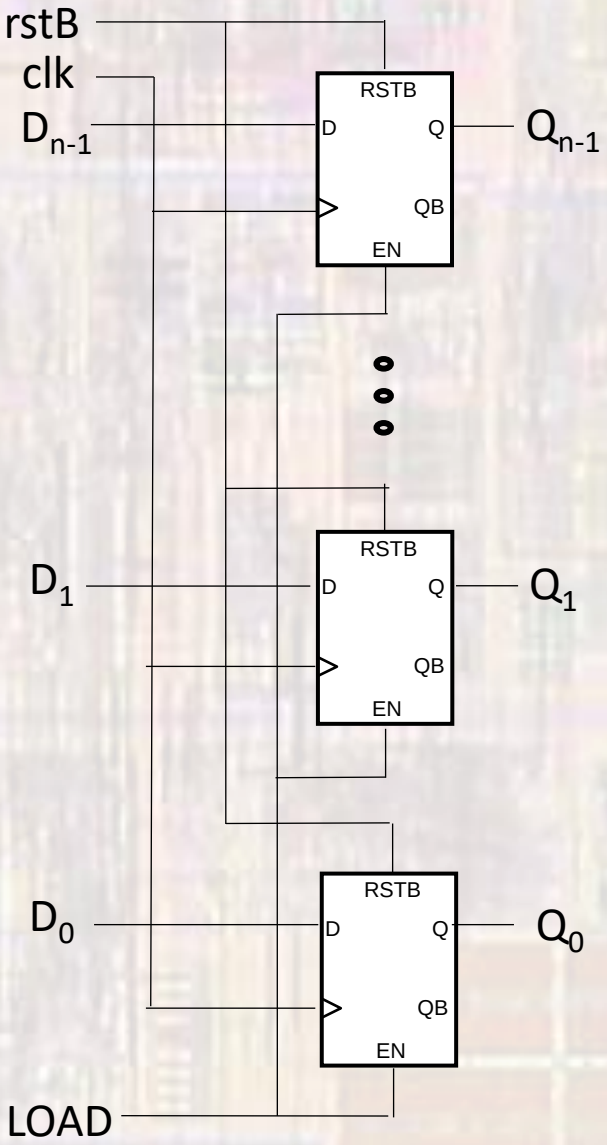
n-bit register



n bit wide bus



signal indices n-1 down to 0



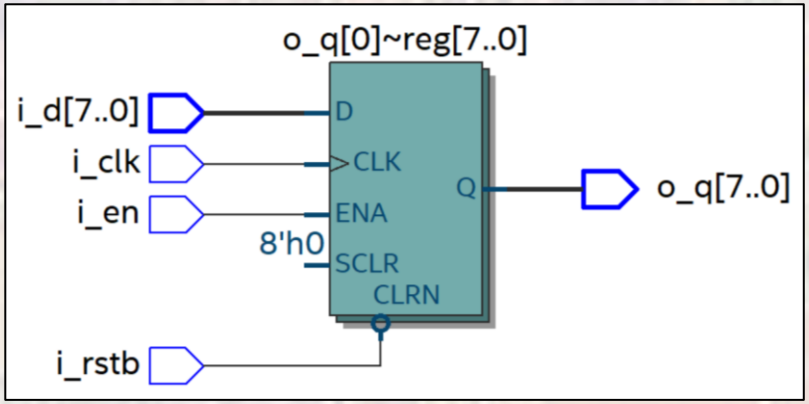
Registers - HDL

- 8 bit register w/enable

```
--------------------------------
--
-- reg_8_bit_en.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- 8 bit register w/enable example
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

---- fixed 8 bit register -----------------
entity reg_8_bit_en is
   port (
      i_d   :  in std_logic_vector(7 downto 0);
      i_en  :  in std_logic;
      i_clk :  in std_logic;
      i_rstb:  in std_logic;

      o_q   :  out std_logic_vector(7 downto 0)
   );
end entity;
```

```
architecture behavioral of reg_8_bit_en is
begin
   process(i_clk, i_rstb)
      begin
         if (i_rstb = '0') then
            o_q <= "00000000";
         elsif (rising_edge(i_clk)) then
            if (i_en = '1') then
               o_q <= i_d;
            else
               null;
            end if;
         else
            null;
         end if;
   end process;
end behavioral;
--------------------------------
```

not necessary



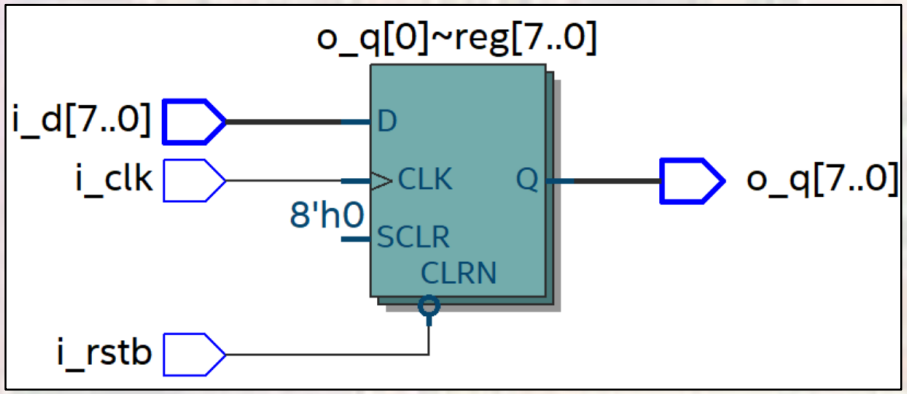
Registers - HDL

- 8 bit register – no enable

Note: no elses - matches template

```
--------------------------------
--
-- reg_8_bit.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- 8 bit register example
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

---- fixed 8 bit register -----------------
entity reg_8_bit is
   port (
      i_d   :  in std_logic_vector(7 downto 0);
      i_clk :  in std_logic;
      i_rstb:  in std_logic;

      o_q   :  out std_logic_vector(7 downto 0)
   );
end entity;
```

```
architecture behavioral of reg_8_bit is
begin
   process(i_clk, i_rstb)
      begin
         if (i_rstb = '0') then
            o_q <= "00000000";
         elsif (rising_edge(i_clk)) then
            o_q <= i_d;
         end if;
   end process;
end behavioral;
--------------------------------
```



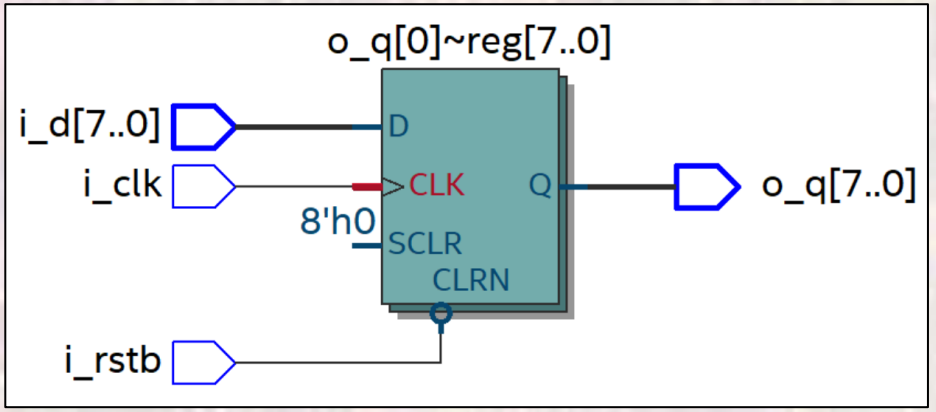
Registers - HDL

- N bit register

```
--
-- reg_n_bit.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- n bit register example
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;



---- generic N bit register -----------------
entity reg_n_bit is
   generic(
      N: integer := 8
   );
   port (
      i_d   :  in std_logic_vector((N - 1) downto 0);
      i_clk :  in std_logic;
      i_rstb:  in std_logic;

      o_q   :  out std_logic_vector((N - 1) downto 0)
    );
end entity;

architecture behavioral of reg_n_bit is
begin
   process(i_clk, i_rstb)
      begin
         if (i_rstb = '0') then
            o_q <= (others => '0');
         elsif (rising_edge(i_clk)) then
            o_q <= i_d;
         end if;
   end process;
end behavioral;
```

generic section added
- defines N
- defaults N to 8
- can be overwritten when instantiated

Vector sizes now defined with N



(others => '0') used since N can change

Registers - HDL

- 64 bit register – using instantiation

Component Prototype

```
--
-- COMPONENT PROTOTYPE ------------
--
component reg_n_bit is
    generic(
        N: integer := 8
    );
    port (
        i_d    :  in std_logic_vector((N - 1) downto 0);
        i_clk  :  in std_logic;
        i_rstb:  in std_logic;

        o_q    :  out std_logic_vector((N - 1) downto 0)
     );
end component;
-----------------------------------
```

Instantiation

```
--
-- COMPONENT INSTANTIATION -------
--
myBigRegister: reg_n_bit
generic map(
            N => 64
)
port map(
          i_rstb    => i_RSTB,
          i_clk     => i_CLK,
          i_d       => i_D,
          o_q       => o_Q
);
-----------------------------------
```

Generic - overwrite

Explicit port mapping

Registers - HDL

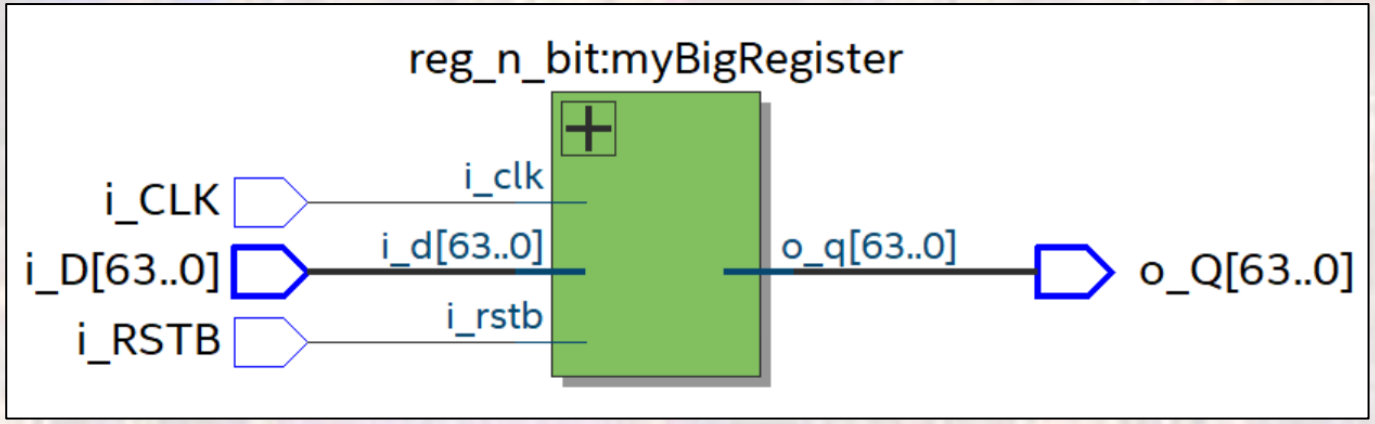
- 64 bit register – using instantiation

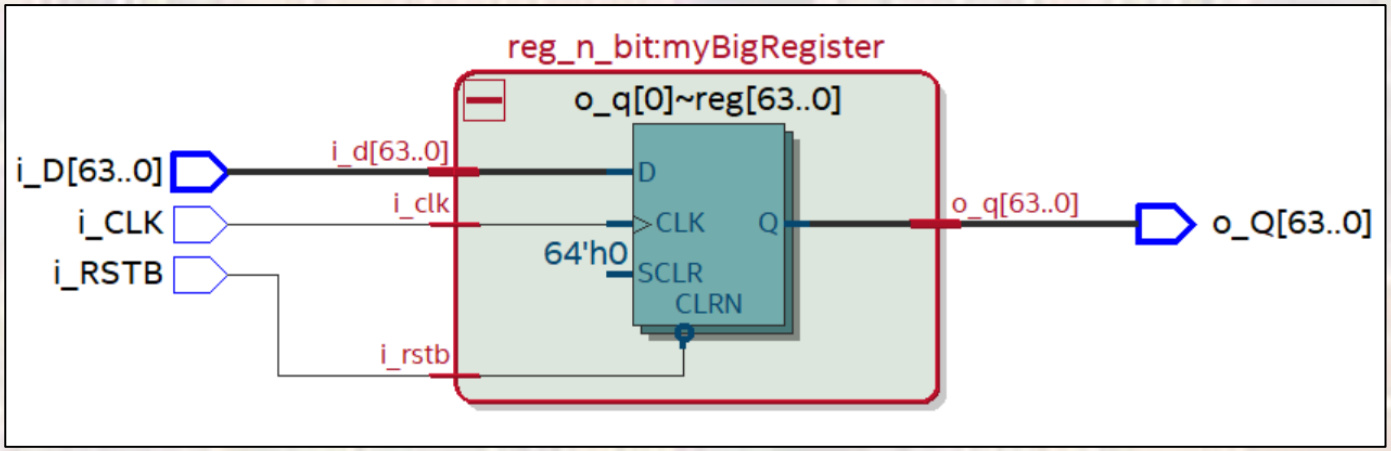
```
--------------------------------
--
-- reg_64_bit.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- 64 bit register using instantiation
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity reg_64_bit is
   port (
      i_D   :  in std_logic_vector(63 downto 0);
      i_CLK :  in std_logic;
      i_RSTB:  in std_logic;

      o_Q   :  out std_logic_vector(63 downto 0)
   );
end entity;
```

```
architecture behavioral of reg_64_bit is

   --
   -- COMPONENT PROTOTYPE ------------
   --
   component reg_n_bit is
      generic(
         N: integer := 8
      );
      port (
         i_d   :  in std_logic_vector((N - 1) downto 0);
         i_clk :  in std_logic;
         i_rstb:  in std_logic;

         o_q   :  out std_logic_vector((N - 1) downto 0)
       );
   end component;
   -----------------------------------

begin

   --
   -- COMPONENT INSTANTIATION -------
   --
   myBigRegister: reg_n_bit
   generic map(
               N => 64
   )
   port map(
            i_rstb   => i_RSTB,
            i_clk    => i_CLK,
            i_d      => i_D,
            o_q      => o_Q
   );
   -----------------------------------

end behavioral;
------------------------------
```

Registers - HDL

- 64 bit register – using instantiation



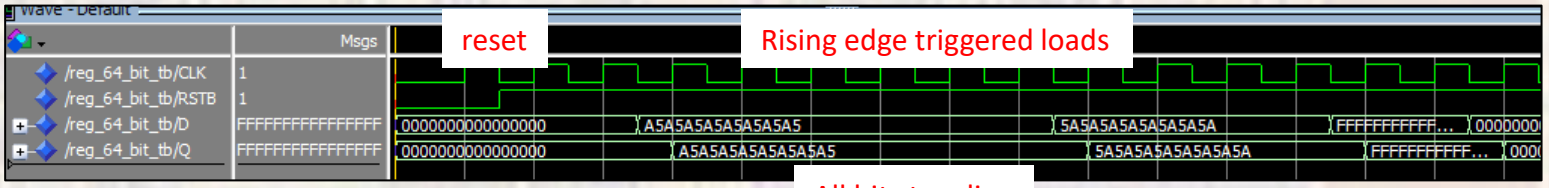


Registers - HDL

- 64 bit register – testbench

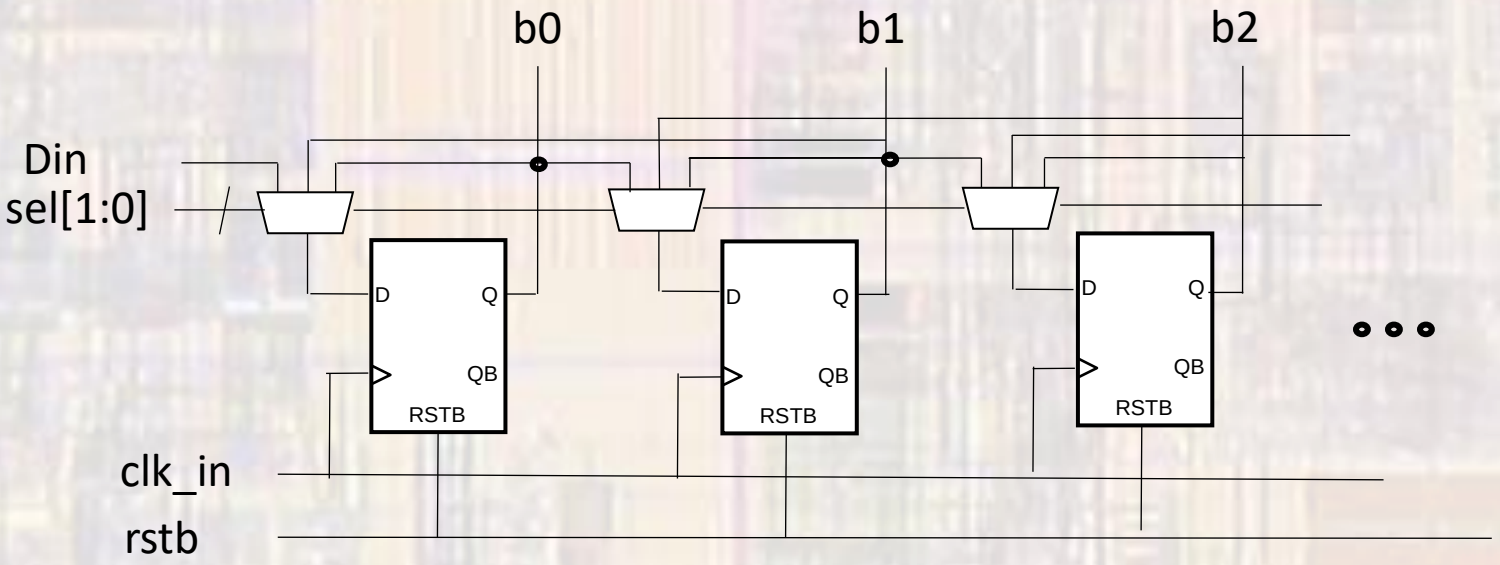
```
--------------------------------
--
-- reg_64_bit_tb.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- testbench for 64 bit  register
-- of reg_64_bit.vhdl
--
-- brute force implementation
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity reg_64_bit_tb is
   -- no entry - testbench
end entity;

architecture testbench of reg_64_bit_tb is
   signal   CLK:     std_logic;
   signal   RSTB:    std_logic;
   signal   D:       std_logic_vector(63 downto 0);

   signal   Q:       std_logic_vector(63 downto 0);

   constant PER:  time  := 20 ns;
   ----------------------------------
   -- Component prototype
   ----------------------------------
   COMPONENT reg_64_bit
      port (
      i_D   :  in std_logic_vector(63 downto 0);
      i_CLK :  in std_logic;
      i_RSTB:  in std_logic;

      o_Q   :  out std_logic_vector(63 downto 0)
      );
   END COMPONENT;
   ----------------------------------

   begin
```

```
   ----------------------------------
   -- Device under test (DUT)
   ----------------------------------
   DUT: reg_64_bit
      port map(
               i_clk       => CLK,
               i_rstb      => RSTB,
               i_d         => D,

               o_q         => Q
               );


   ----------------------------------
   -- Test processes
   ----------------------------------

   -- Clock process - 50MHz
   clock: process    -- no sensitivity list
      begin
         CLK <= '0';
         wait for PER;
         infinite: loop
            CLK <= not CLK; wait for PER/2;
         end loop;
   end process clock;

   -- Reset process
   reset: process    -- no sensitivity list
   begin
      RSTB <= '0'; wait for 1.5*PER;
      RSTB <= '1'; wait;
   end process reset;

   -- Run process
   run: process      -- no sensitivity list
      begin
         -------------------------
         -- Initialize  input
         D <= (others => '0');

         wait for PER*3.5; -- wait for reset

         -- setup varius D values
         D <= x"A5A5A5A5A5A5A5A5";
         wait for 6*PER;
         D <= x"5A5A5A5A5A5A5A5A";
         wait for 4*PER;
         D <= (others => '1');
         wait for 2*PER;
         D <= (others => '0');
         wait for 2*PER;

   end process run;

   ----------------------------------
   -- End test processes
   ----------------------------------
end architecture;
```



Registers - HDL

- Shift Register
  - Shift a series of bits within a register
  - sel[0]: shift right / left
  - sel[1]: shift / no-shift



Registers - HDL

- Shift Register

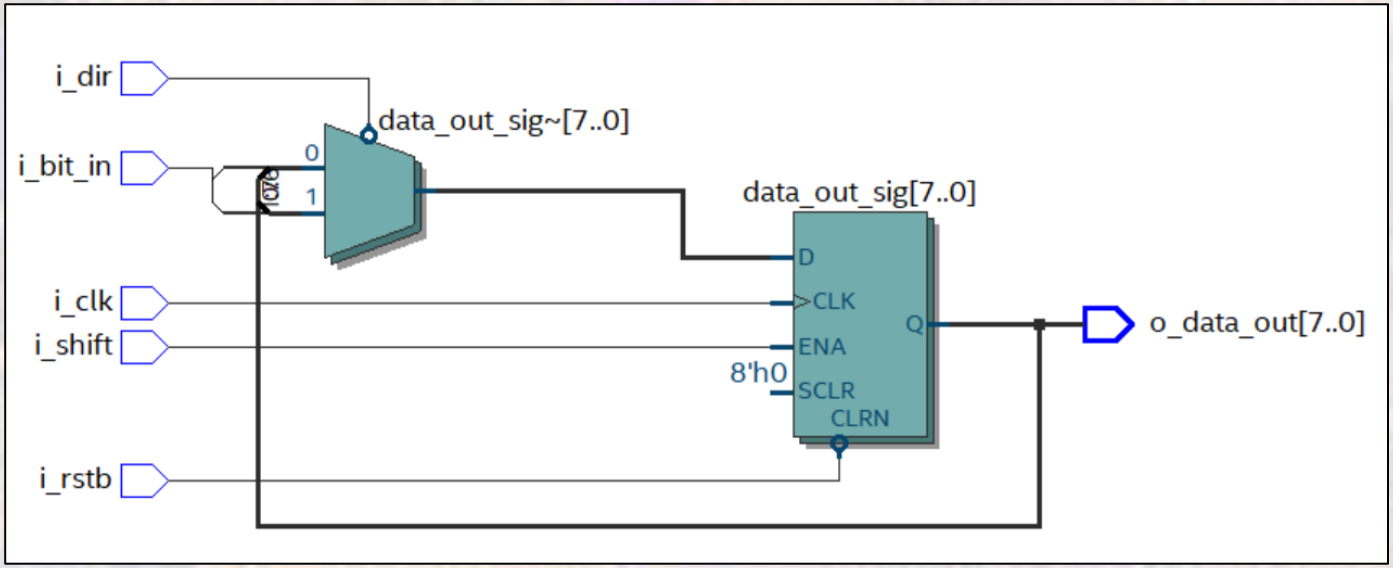
```
--------------------------------------
--
-- shift_reg_8_bit.vhdl
--
-- created 2/29/17
-- tj
--
-- rev 0
--------------------------------------
--
-- 8 bit L/R shift register example
--
--------------------------------------
--
-- Inputs: rstb, clk, bit_in, shift, dir
-- Outputs: data_out[7:0]
--
--------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity shift_reg_8_bit is
    port (
      i_clk :     in std_logic;
      i_rstb :    in std_logic;
      i_bit_in :  in std_logic;
      i_shift:    in std_logic;
      i_dir:      in std_logic;  -- 0 for left, 1 for right

      o_data_out :out std_logic_vector(7 downto 0)
    );
end entity;
```

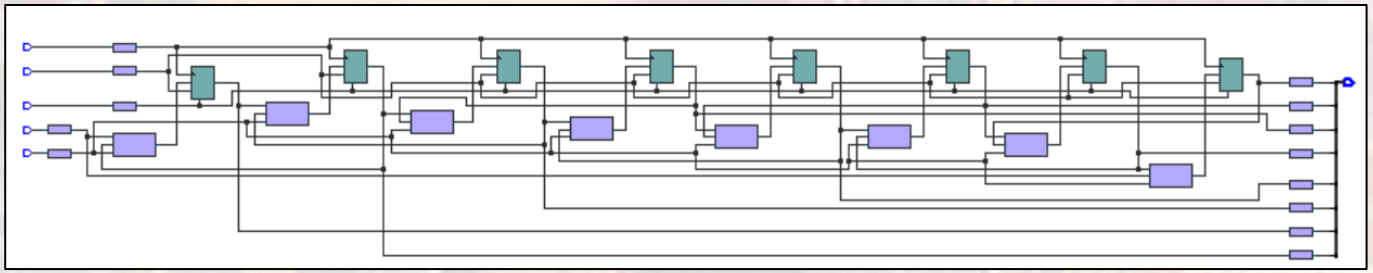
```
architecture behavioral of shift_reg_8_bit is

   --
   -- internal signals
   --
   signal data_out_sig: std_logic_vector(7 downto 0);

begin

   process(i_clk, i_rstb)
      begin
         --
         -- reset
         --
         if (i_rstb = '0') then
            data_out_sig <= (others => '0');
         --
         -- rising clk edge
         --
         elsif (rising_edge(i_clk)) then
            --
            -- shifting
            --
            if (i_shift = '0') then
               data_out_sig <= data_out_sig;
            elsif(i_dir = '0') then
               data_out_sig <= data_out_sig(6 downto 0) & i_bit_in;
            else
               data_out_sig <= i_bit_in & data_out_sig(7 downto 1);
            end if;
         end if;
   end process;

   --
   -- Output logic
   --
   o_data_out <= data_out_sig;

end behavioral;
```

why?

Registers - HDL

- Shift Register





Registers - HDL

- Shift Register - testbench

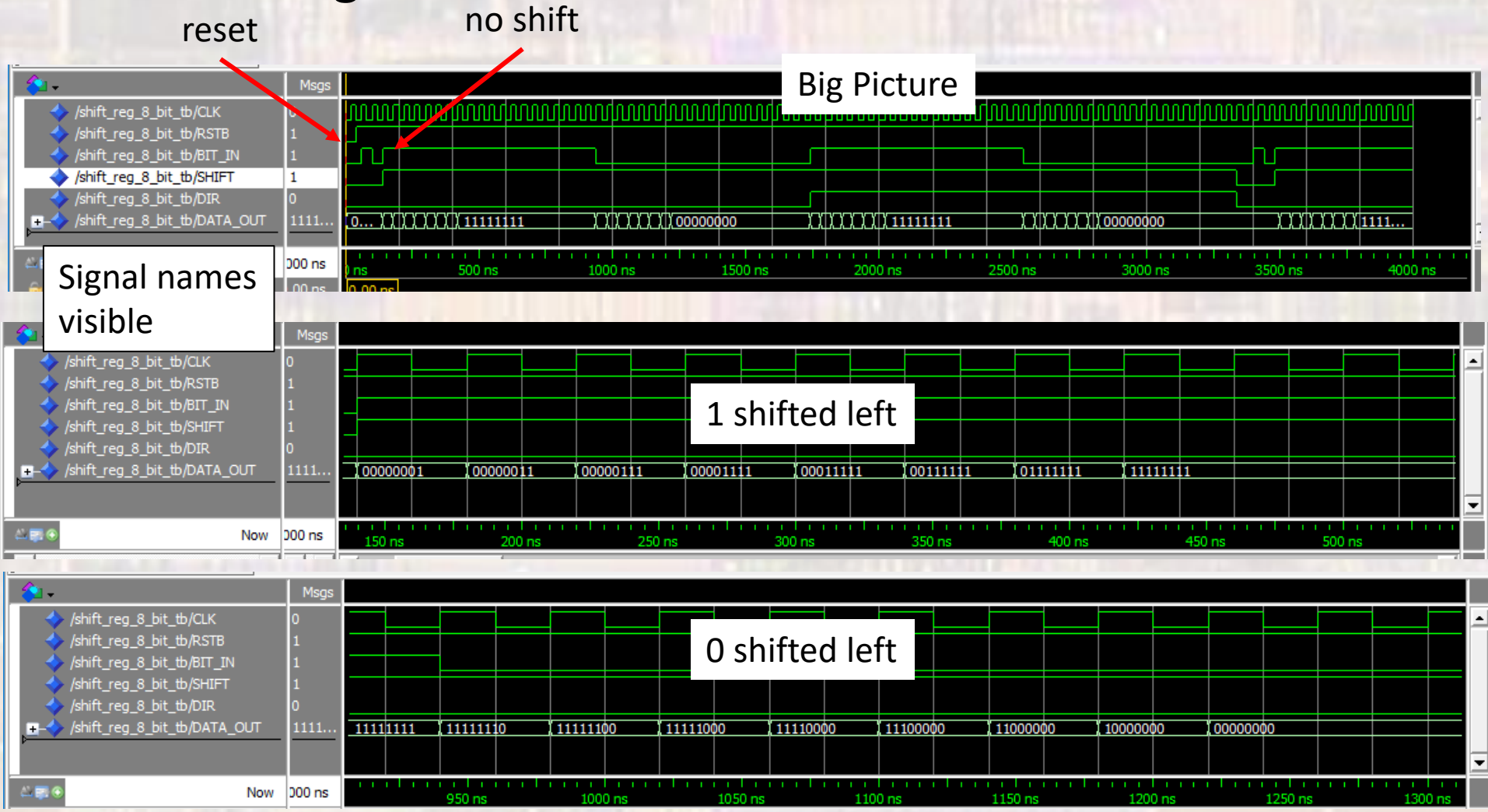
```
--------------------------------
--
-- shift_reg_8_bit_tb.vhdl
--
-- created: 1/26/18
-- by: johnsontimoj
-- rev: 0
--
-- testbench for 8 bit shift register
-- of shift_reg_8_bit.vhdl
--
-- brute force implementation
--
--------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity shift_reg_8_bit_tb is
   -- no entry - testbench
end entity;

architecture testbench of shift_reg_8_bit_tb is
   signal   CLK:      std_logic;
   signal   RSTB:     std_logic;
   signal   BIT_IN:   std_logic;
   signal   SHIFT:    std_logic;
   signal   DIR:      std_logic;

   signal   DATA_OUT:std_logic_vector(7 downto 0);

   constant PER:  time  := 20 ns;
   ---------------------------------
```

```
   ---------------------------------
   -- Component prototype
   ---------------------------------
   COMPONENT shift_reg_8_bit
      port (
         i_clk :        in std_logic;
         i_rstb :       in std_logic;
         i_bit_in :     in std_logic;
         i_shift:       in std_logic;
         i_dir:         in std_logic;  -- 0 for left, 1 for right

         o_data_out :   out std_logic_vector(7 downto 0)
         );
   END COMPONENT;
   ---------------------------------

   begin

   ---------------------------------
   -- Device under test (DUT)
   ---------------------------------
   DUT: shift_reg_8_bit
      port map(
               i_clk       => CLK,
               i_rstb      => RSTB,
               i_bit_in    => BIT_IN,
               i_shift     => SHIFT,
               i_dir       => DIR,

               o_data_out  => DATA_OUT
               );
```

```
   ---------------------------------
   -- Test processes
   ---------------------------------

   -- Clock process - 50MHz
   clock: process    -- note - no sensitivity list allowed
      begin
         CLK <= '0';
         wait for PER;
         infinite: loop
            CLK <= not CLK; wait for PER/2;
         end loop;
   end process clock;

   -- Reset process
   reset: process    -- note - no sensitivity list allowed
   begin
      RSTB <= '0'; wait for 1.5*PER;
      RSTB <= '1'; wait;
   end process reset;

   -- Run process
   run: process      -- note - no sensitivity list allowed
      begin
         ------------------------
         -- Initialize  inputs
         BIT_IN <= '0';
         SHIFT <= '0';
         DIR <= '0';

         wait for PER*3;    -- wait for reset

         -- verify no shift
         BIT_IN <= '1'; wait for 2*PER;
         BIT_IN <= '0'; wait for 2*PER;

         -- verify shift lt
         SHIFT <= '1';
         DIR <= '0';
         BIT_IN <= '1'; wait for 40*PER;
         BIT_IN <= '0'; wait for 40*PER;

         -- verify shift rt
         SHIFT <= '1';
         DIR <= '1';
         BIT_IN <= '1'; wait for 40*PER;
         BIT_IN <= '0'; wait for 40*PER;
   end process run;

   ---------------------------------
   -- End test processes
   ---------------------------------
end architecture;
```

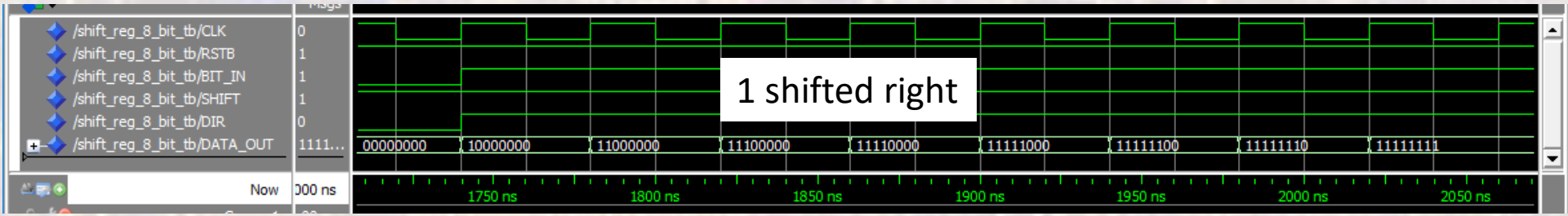
Registers - HDL

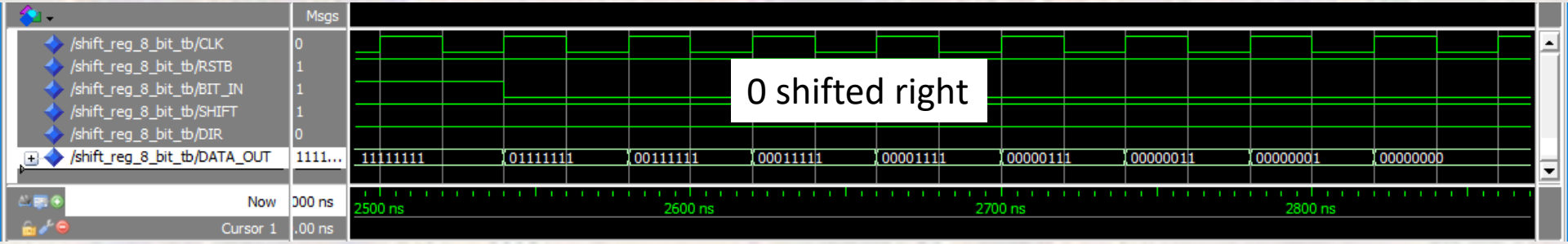
- Shift Register - verification



Registers - HDL

- Shift Register - verification





Registers - HDL

- Shift Register - n

```
----------------------------------------
--
-- shift_reg_n_bit.vhdl
--
-- created 2/29/17
-- tj
--
-- rev 0
----------------------------------------
--
-- n bit L/R shift register example
--
----------------------------------------
--
-- Inputs: rstb, clk, bit_in, shift, dir
-- Outputs: data_out[(N-1):0]
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity shift_reg_n_bit is
   generic(
         N: natural := 8
         );
   port(
      i_clk :     in std_logic;
      i_rstb :    in std_logic;
      i_bit_in :  in std_logic;
      i_shift:    in std_logic;
      i_dir:      in std_logic;  -- 0 for left, 1 for right

      o_data_out :out std_logic_vector((N-1) downto 0)
    );
end entity;
```

```
architecture behavioral of shift_reg_n_bit is

   --
   -- internal signals
   --
   signal data_out_sig: std_logic_vector((N-1) downto 0);

begin

   process(i_clk, i_rstb)
      begin
         --
         -- reset
         --
         if (i_rstb = '0') then
            data_out_sig <= (others => '0');
         --
         -- rising clk edge
         --
         elsif (rising_edge(i_clk)) then
            --
            -- shifting
            --
            if (i_shift = '0') then
               data_out_sig <= data_out_sig;
            elsif(i_dir = '0') then
               data_out_sig <= data_out_sig(((N-1)-1) downto 0) & i_bit_in;
            else
               data_out_sig <= i_bit_in & data_out_sig((N-1) downto 1);
            end if;
         end if;
   end process;

   --
   -- Output logic
   --
   o_data_out <= data_out_sig;

end behavioral;
```