VHDL
Best Practices – CPE1500

Last updated 1/21/25

VHDL Best Practices – CPE1500

- Best Practices ???
  - Best practices are often defined by company, toolset or device
  - In our case – Dr. Johnson is setting the “best practices”
  - These rules are for Class/Lab purposes. Industry best practices would include a much more extensive list
    - I/O synchronization
    - Clock domains
    - Revision control
    - Test coverage
    - …
  - These rules have been reduced from the ELE3510 rules

VHDL Best Practices – page 1/2

- Use meaningful names for blocks, signals and programs
- Use i_xyz for block input signal names and o_xyz for block output signal names
- Use _tb and _de10 name extensions for testbenches and hardware implementations
- 1 design file, instantiate it in the testbench and hardware implementation files
- No latches
- Use explicit port mapping when instantiating components
- No signal initialization in declarations

VHDL Best Practices – page 2/2

- No variables as signals
- I/O signals are SLV, internal signals are signed/unsigned as appropriate
- Use rising_edge()
- Reset_bar for general (control) synchronous logic
- Clock divider OK for slowing to human speeds
- Break FSM designs into separate Next State, Register, and Output Logic(Mealy) sections

Use meaningful names …

- Use meaningful names for blocks, signals and programs

Stoplight with emergency detection for lab 22

~~lab22.vhdl~~

~~testbench.vhdl~~

~~board.vhdl~~

stoplight_w_emergency.vhdl

stoplight_w_emergency_tb.vhdl

stoplight_w_emergency_de10.vhdl

Note: primary function followed by secondary functions

Use i_xyz …

- Use i_xyz for block input names and o_xyz for block output names

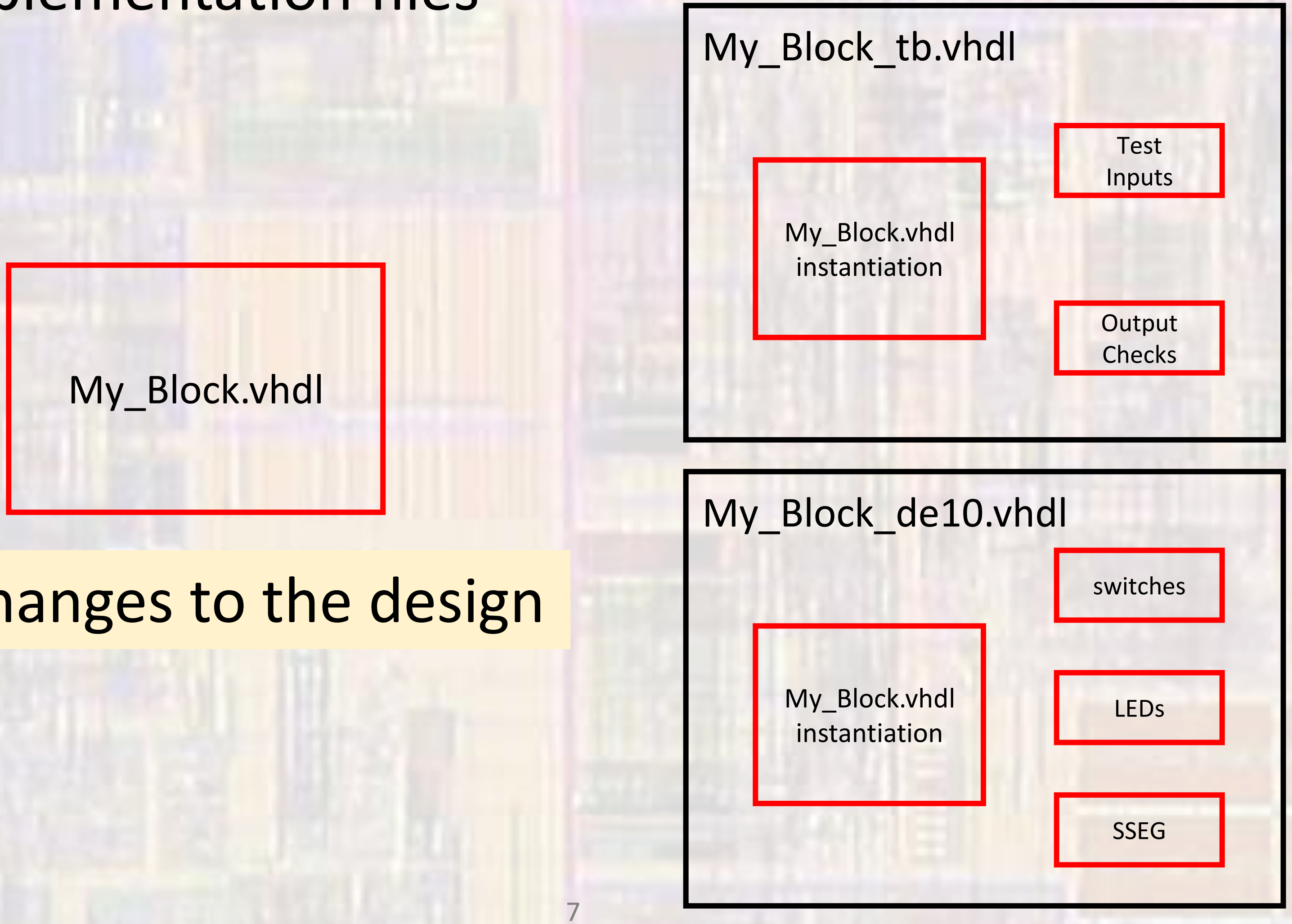
```
port(i_A:       in    std_logic_vector(3 downto 0);
     i_B:       in    std_logic_vector(3 downto 0);
     i_CIN:     in    std_logic;
     o_SUM:     out   std_logic_vector(3 downto 0);
     o_COUT:    out   std_logic
     );
```

Exception: When using the pin-names from the QSF file for DE10 implementations, the names must match exactly

```
entity lab_4_de10 is
  port(
    CLOCK_50 :  in    std_logic;
    SW:         in    std_logic_vector(9 downto 0);
    HEX0:       out   std_logic_vector(7 downto 0);
    HEX1:       out   std_logic_vector(7 downto 0);
    HEX2:       out   std_logic_vector(7 downto 0);
    HEX3:       out   std_logic_vector(7 downto 0)
  );
end entity;
```

1 design file, instantiate …

- 1 design file, instantiate it in the testbench and HW implementation files

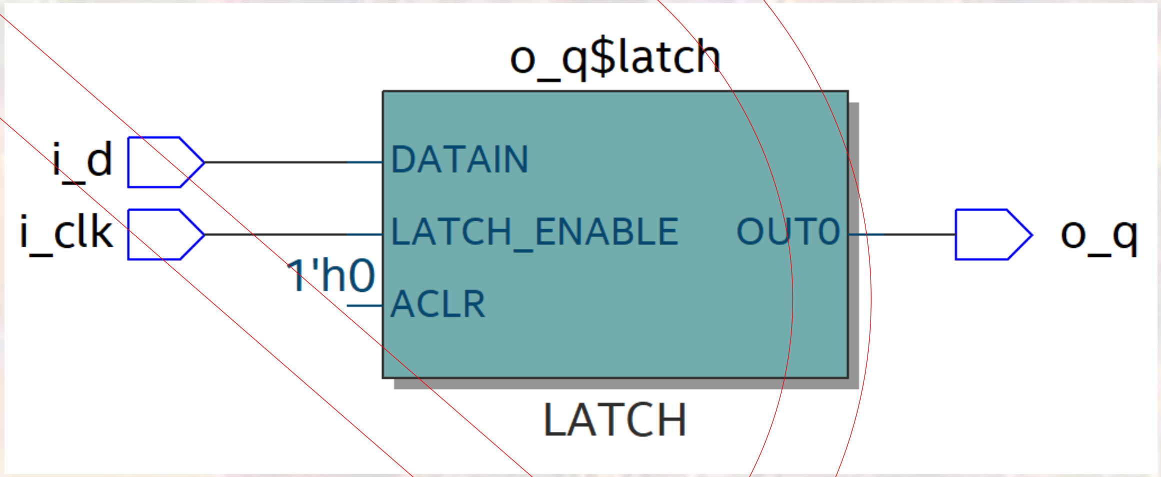


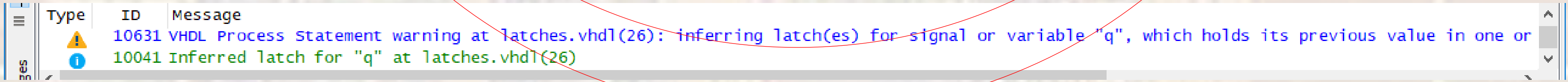
No Changes to the design

No Latches

- No Latches

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

entity latches is
  port(
        i_clk:    in std_logic;
        i_d :     in std_logic;

        o_q :   out std_logic
  );
end entity latches;

architecture behavioral of latches is
begin
  process(i_clk, i_d)
  begin
    if(i_clk = '1') then
      o_q <= i_d;
    end if;
  end process;
end architecture;
```





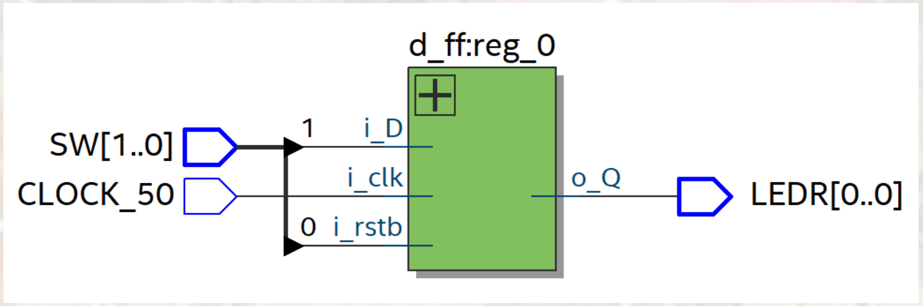
```
Type  ID     Message
      10631  VHDL Process Statement warning at latches.vhdl(26): inferring latch(es) for signal or variable "q", which holds its previous value in one or
      10041  Inferred latch for "q" at latches.vhdl(26)
```

Use Explicit Port Mapping

- Always use explicit port mapping on component instantiation

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

entity dff_instantiation is
    port (
            CLOCK_50 :  in std_logic;
            SW      :   in std_logic_vector(1 downto 0);
            LEDR :      out std_logic_vector(0 downto 0)
     );
end entity;
```

port map



```
architecture structural of dff_instantiation is

   component d_ff
      port(
            i_D :     in std_logic;
            i_clk :   in std_logic;
            i_rstb:   in std_logic;

            o_Q:      out std_logic
      );
   end component;

begin
   reg_0: d_ff
      port map(i_D   => SW(1),
               i_clk    => CLOCK_50,
               i_rstb   => SW(0),
               o_Q      => LEDR(0)
      );
end architecture;
```

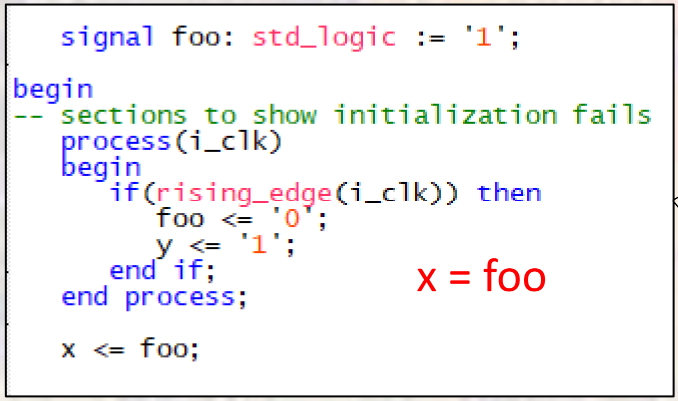
component prototype

explicit port mapping
component pin => my signal

No Signal Initialization

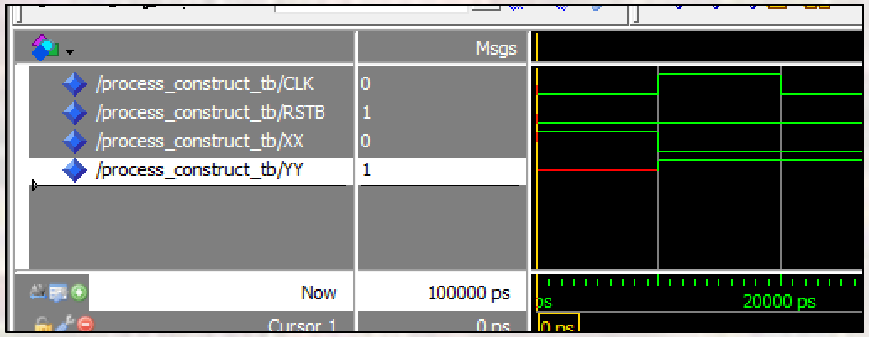
- No signal initialization in declarations
  - It is not possible to implement signal initialization in hardware
  - Rely on reset for any required initialization in hardware

signal foo: std_logic := ‘1’;

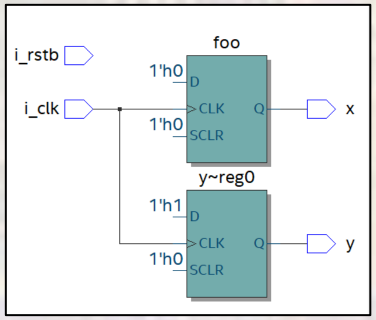


```
  signal foo: std_logic := '1';

begin
-- sections to show initialization fails
  process(i_clk)
  begin
     if(rising_edge(i_clk)) then
        foo <= '0';
        y <= '1';
     end if;
  end process;

  x <= foo;
```

x = foo



```
                              Msgs
/process_construct_tb/CLK     0
/process_construct_tb/RSTB    1
/process_construct_tb/XX      0
/process_construct_tb/YY      1

Now                           100000 ps
Cursor 1                      0 ps
```



Sim shows x (foo) starts at ‘1’

Hardware has nothing to make x (foo) start at 1

Sim says it works
HW fails !!!

No Variables as Signals

- No variables as signals
  - We are using HDL code to represent HARDWARE
  - Variables do not have a HARDWARE analog
  - Variables are treated differently than signals
    - Variables are updated immediately in a process
    - Signals are only updated at the end of a process
  - Variable are appropriate for compile time calculations
    - Generate
    - Test Benches

I/O signals are …

- I/O signals are SLV, internal signals are signed/unsigned as appropriate
  - We are using HDL code to represent HARDWARE

  - I/O ports are represented by std_logic or std_logic_vectors
    - They are interpreted as connections

  - Internal signals
    - Use std_logic to represent single wires
    - Use unsigned to represent unsigned bus signals and structural buses (memory addresses, …)
    - Use signed to represent signed bus signals

Use Rising_Edge()

- Use Rising_Edge()
  - (rising_edge(clk)) instead of (clk’event and clk = ‘1’) in register (FF designs)
  - Also use (falling_edge(clk))
  - These do better multi-state checking in simulation

clk’event includes things like

Z → 1

U → 1

rising_edge only includes 0 → 1

```
process(i_clk, i_rstb)
  begin
    if (i_rstb = '0') then
      o_Q <= '0';
    elsif (rising_edge(i_clk)) then
      o_Q <= i_D;
    end if;
end process;
```

Reset_bar for general …

- Reset_bar for general (control) synchronous logic
  - All non-data path registers will have a rstb signal

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

entity d_ff is
    port (
            i_clk : in std_logic;
            i_rstb:in std_logic;
            i_D :       in std_logic;

            o_Q:        out std_logic
     );
end entity;

architecture behavioral of d_ff is
begin
    process(i_clk, i_rstb)
    begin
        if (i_rstb = '0') then
            o_Q <= '0';
        elsif (rising_edge(i_clk)) then
            o_Q <= i_D;
        end if;
    end process;
end behavioral;
```

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

entity registers is
    generic(
        N:  integer := 8
    );
    port (
            i_clk :  in std_logic;
            i_rstb: in std_logic;
            i_D :       in std_logic_vector((N - 1) downto 0);

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

architecture behavioral of registers 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;
```