Variables, VHDL Rules and Synthesis

Final Exemption

- Students with an **A or an A-** in lecture AND an **A** in lab may be exempt from the DSD final.
  - Decisions will not be made until all graded items are submitted.
  - **This includes lab 11 and quiz 7**

93.00 – 100.00	A
90.00 – 92.99	A-
87.00 – 89.99	B+
83.00 – 86.99	B
80.00 – 82.99	B-
77.00 – 79.99	C+
73.00 – 76.99	C
70.00 – 72.99	C-
60.00 – 69.99	D
00.00 – 59.99	F

*If you take the final and your grade is higher than one of your previous exams, your final grade will replace your lowest exam grade.

Variables

- Used to store intermediate values in a process or subprogram
  - can only be accessed from a single process or subprogram in which it is declared
- Declared like a signal
  - VARIABLE count : std_logic_vector(3 downto 0);
- Assignments made with :=
  - Count := “0000”;

Variables

- Declared in the process before the BEGIN statement
- Take the value assigned to them immediately
  - Similar to a traditional programming language
  - Remember signals don't get assigned until the end of the process
- Order matters because they are updated immediately

Variable vs. Signal

- Signals represent wires or registers (when in a synchronous process)
- Variables store intermediate values and do not synthesize to hardware components
- They cannot be used interchangeably

Variable Example

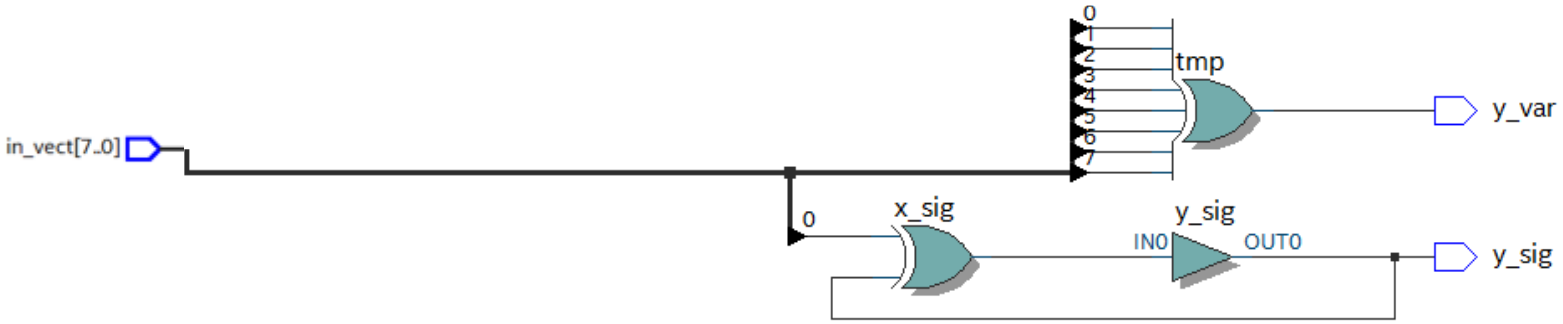
Here variable is used to store an intermediate value

```
entity generic_xor is
      generic(wide          : integer := 8);
      port(in_vect          : in std_logic_vector(wide-1 downto 0);
           y                : out std_logic);
end generic_xor;

architecture model of generic_xor is

   begin

    xorit: process (in_vect)
      variable tmp: std_logic;
      BEGIN
        tmp := '0';
          for i in wide-1 downto 0 LOOP
            tmp  := tmp XOR in_vect(i);
          end LOOP;
        y <= tmp ;
      END PROCESS;

end model;
```

Variable vs. Signal Example

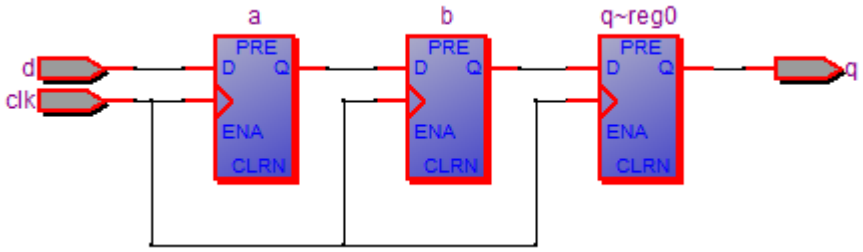
```
xor_var: process (in_vect)
   variable tmp: std_logic;
BEGIN
  tmp := '0';
  for i in 7 downto 0 LOOP
    tmp  := tmp XOR in_vect(i);
  end LOOP;
  y_var <= tmp ;
END PROCESS;
```

```
xor_sig: process (in_vect)
BEGIN
  for i in 7 downto 0 loop
  x_sig <= x_sig xor in_vect(i);
  end loop;
  y_sig <= x_sig ;
end process;
```



Variable vs. Signal Example

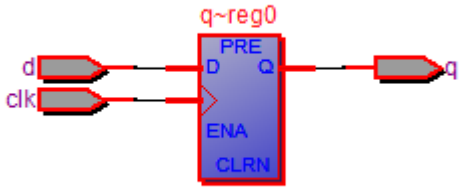
```
entity regs is
    port(d, clk       : in std_logic;
         q            : out std_logic);
    end regs;

architecture behave of regs is
    signal a,b : std_logic;
Begin
    process(clk)
        begin
            if (clk'event and clk = '1') then
                a <= d;
                b <= a;
                q <= b;
            end if;
    end process;
end behave;
```



How many registers?

Variable vs. Signal Example

```
entity regv is
    Port (clk,d       : in std_logic;
          q           : out std_logic);
    end regv;

architecture behave of regv is
   begin
   process(clk)
      variable a,b : std_logic;
    begin
       if (clk'event and clk = '1') then
          a := d;
          b := a;
          q <= b;
        end if;
    end process;
end behave;
```



How many registers?

VHDL RULES

One Output Per process

- This is a standard adopted by the ECTET department.
- What is an output of a process?
  - Any signal on the left hand side of an assignment statement
- The number of processes does not affect the final implementation. However ….
  - Less chance of unintentionally creating a latch
  - Higher probability that that process can be re-used in a future design

Avoid Latches

- If a signal is assigned in one branch of a case, it must be assigned in all branches

```
bad: PROCESS (inputs) IS
    BEGIN
        CASE inputs IS
            WHEN "001"| "010" | "100" | "111" =>
                S <= '1';
            WHEN "011" | "101" | "110" | "111" =>
                Cout <= '1';
            WHEN OTHERS =>
                S <= '0';
                Cout <= '0';
        END CASE;
    END PROCESS;
END behavioral;
```

What is the problem here?

Avoid Latches

- If a signal is assigned in one branch of if/elsif/else, it must be assigned in all branches

```
Bad: Process(inputs) is
    Begin
        if inputs = “001” then
            S <= ‘1’;
        Elsif inputs = “011” then
              Cout <= ‘1’;
        Else
              S <= ‘0’;
              Cout <= ‘0’;
        End if;
    End Process;
```

What is the problem here?

Avoid Latches

- An if requires an else unless default values are assigned prior to the if/elsif/else statement

```
process(ins)
  begin
    if (ins = "00") then
      y <= "11";
    elsif (ins = "10") then
      y <= "00";
    elsif (ins = "01") then
      y <= "10";
  end if;
end process;
```

What is the problem here?

Avoid Latches

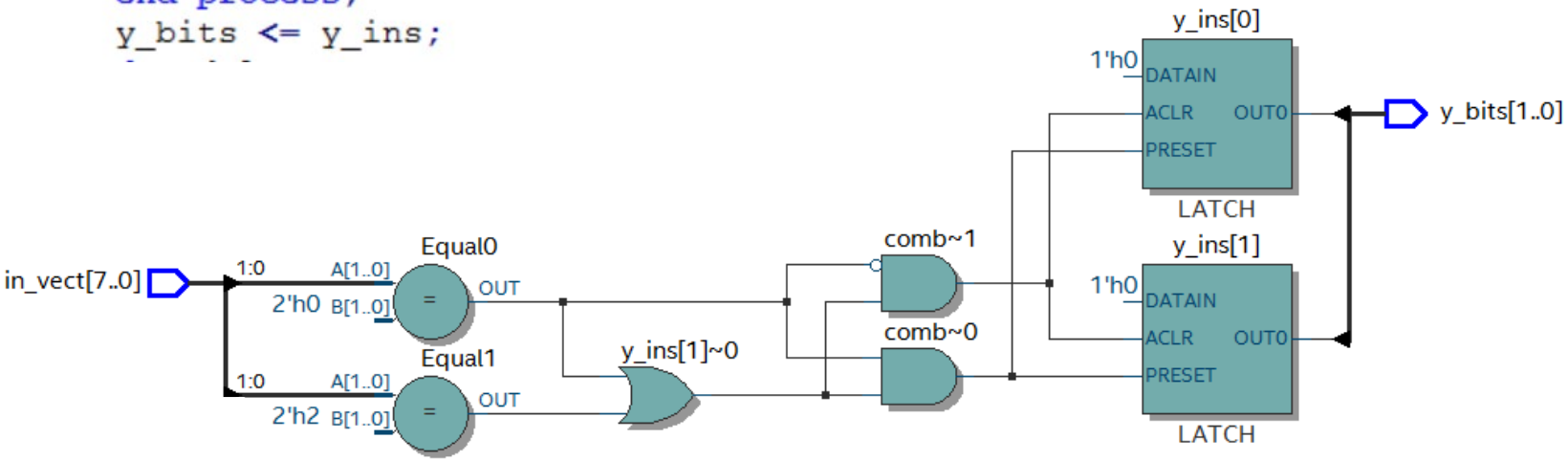
- If the same signal is on the right and the left hand sides of an assignment it must be in a synchronous (clk) process

```
process(ins)
  begin
    if (ins = "00") then
      y <= "11";
    elsif (ins = "10") then
      y <= "00";
    else
      y <= y;
  end if;
end process;
```

What is the problem here?

Avoid Latches

```
ins_sig: process (in_vect)
BEGIN
  if in_vect(1 downto 0) = "00" then
    y_ins <= "11";
  elsif in_vect(1 downto 0) = "10" then
    y_ins <= "00";
  else
    y_ins <= y_ins;
  end if;
end process;
y_bits <= y_ins;
```



Coding Guidelines

- From the textbook
  - Use meaningful signal names
  - Use generics when possible
  - Use STD_LOGIC_VECTOR for ports and cast to other types internally
  - Use descending indices (X downto Y)
  - Use the same name for project, file and entity
  - Do not use BUFFER type, use an internal signal instead
  - Comment your code
  - Use 3 + processes for state machines

What is Synthesis

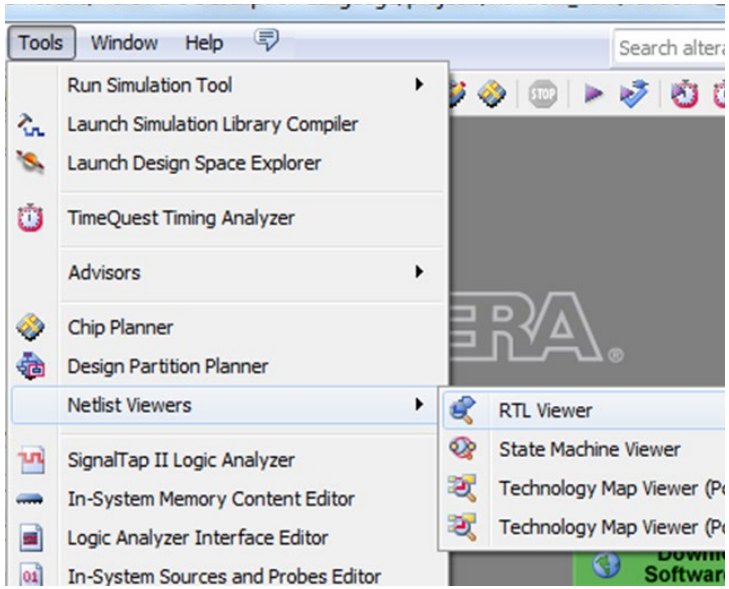
- Synthesis means taking a higher-level description and creating an equivalent lower-level description
  - Moving from English to VHDL, to schematic, to circuit.
  - We will focus on the creation of RTL and gate level design from VHDL

What is RTL?

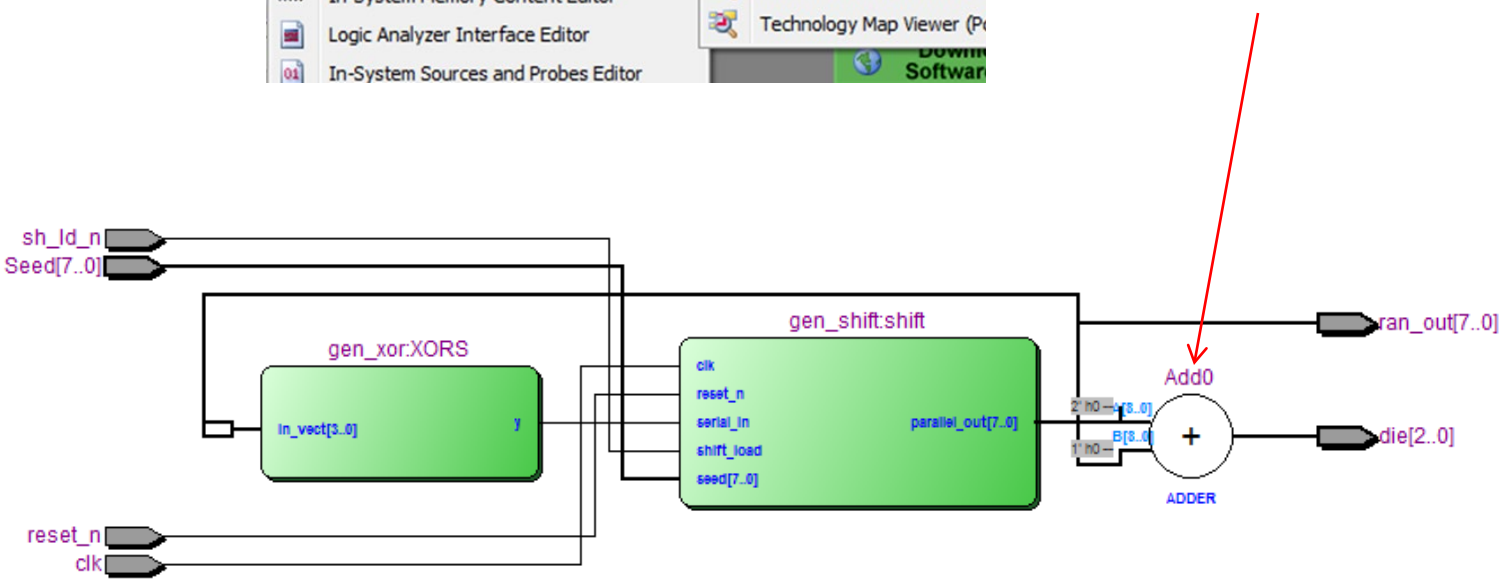
- Register-Transfer Level
- The level of abstraction in which operation is described in terms of data transfers from one register to another register and data manipulations performed combinational logic that exists between the registers

What is RTL?

- Concerned with the flow of data



Indicates addition, but
Does not include circuitry



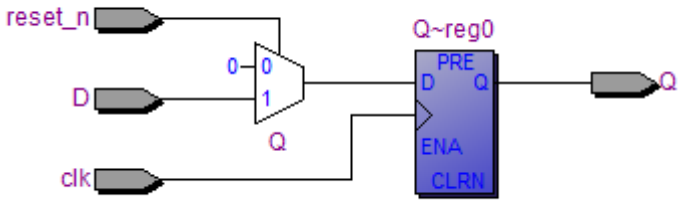
Synthesis Tools

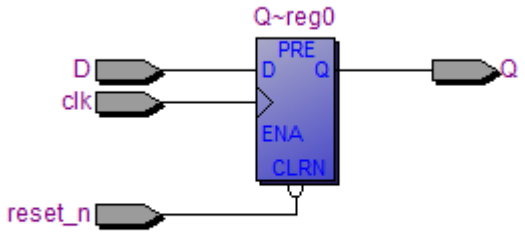
- Synthesis tools convert VHDL to RTL and gate level representation
  - The tool infers basic logic structures from the code
  - The way the code is written makes a BIG difference

Synthesis Example

```
entity deeff is
   port (D, clk, reset_n    : in std_logic;
         Q                  : out std_logic);
   end deeff;
architecture behave of deeff is
   begin
      synch: process(clk, reset_n)is
         begin
            if (clk'event and clk = '1') then
               if (reset_n = '0') then
                  Q <= '0';
               else
                  Q <= D;
               end if;
            end if;
         end process;
   end behave;
```

```
entity deeff is
   port (D, clk, reset_n    : in std_logic;
         Q                  : out std_logic);
   end deeff;
architecture behave of deeff is
   begin
      asynch: process(clk, reset_n)is
         begin
            if (reset_n = '0') then
               Q <= '0';
            elsif (clk'event and clk = '1') then
               Q <= D;
            end if;
         end process;
   end behave;
```



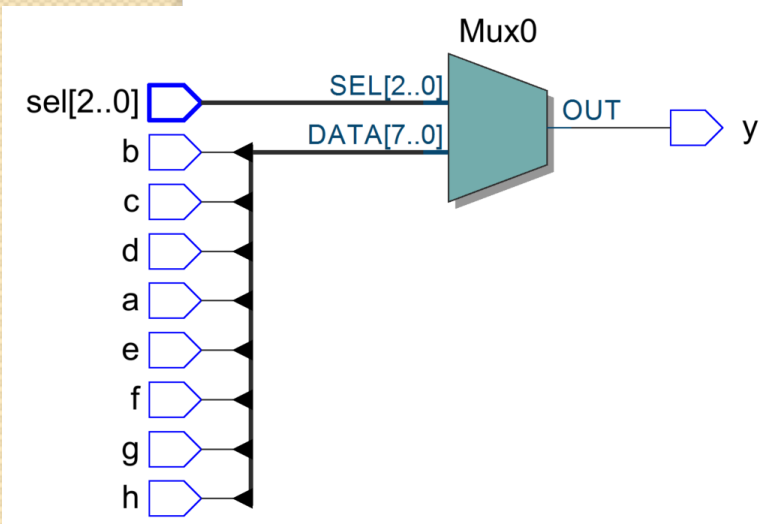


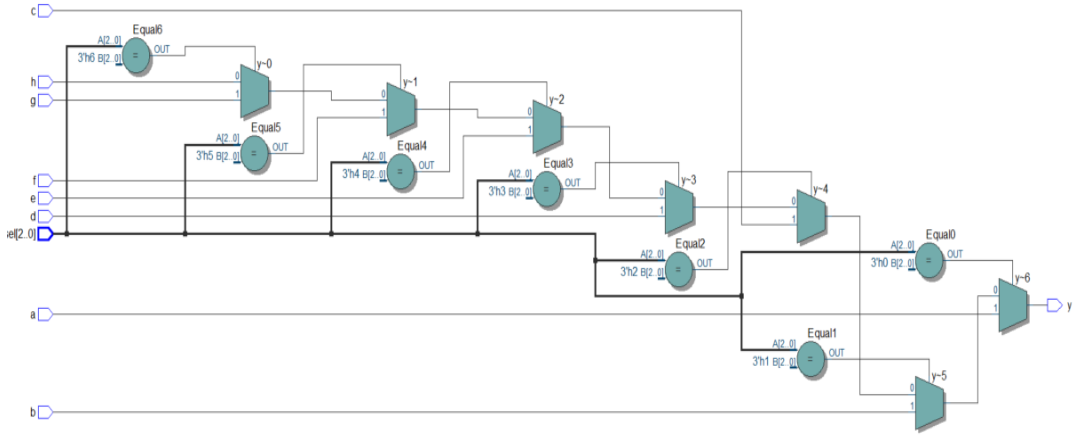
WHY?

Synthesis Example

```
architecture behave of mux8to1 is
begin

   process (a, b, c, d, e, f, g, h, sel) is
      begin
         case sel is
            when "000" => y <= a;
            when "001" => y <= b;
            when "010" => y <= c;
            when "011" => y <= d;
            when "100" => y <= e;
            when "101" => y <= f;
            when "110" => y <= g;
            when others => y <= h;
         end case;
      end process;
end behave;
```

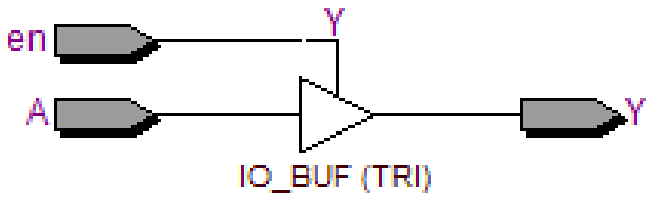
```
process (a, b, c, d, e, f, g, h, sel) is
   begin
         if sel = "000" then y <= a;
         elsif sel =  "001" then y <= b;
         elsif sel =  "010" then y <= c;
         elsif sel =  "011" then y <= d;
         elsif sel =  "100" then y <= e;
         elsif sel =  "101" then y <= f;
         elsif sel =  "110" then y <= g;
         else                    y <= h;
      end if;
   end process;
end behave;
```





Primitive Inference

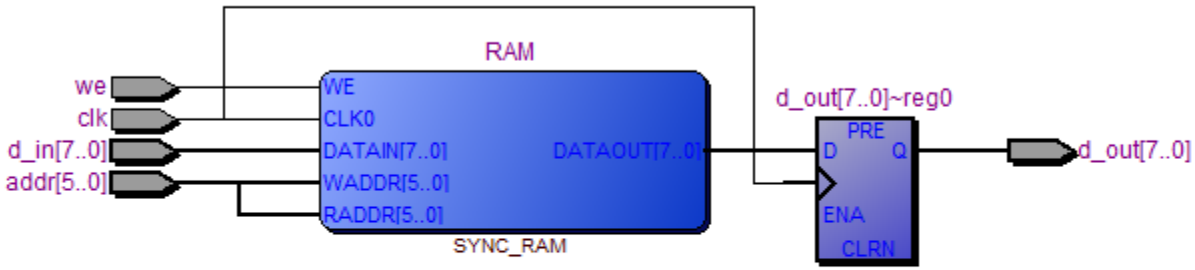
```
entity tri_state is
   port(en, A  : in std_logic;
        Y : out std_logic);
   end tri_state;

architecture behave of tri_state is
   begin
      process (en, A) is
        begin
         if en = '1' then
            Y <= A;
         else
            Y <= 'Z';
         end if;
      end process;
end behave;
```



Primitive inference

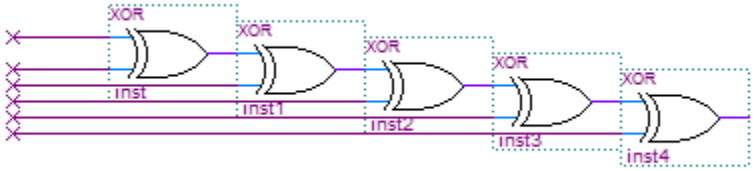
- RAM

```
type ram_type is array ((2**addr_width - 1) downto 0) of std_logic_vector (data_width - 1 downto 0);
signal RAM : ram_type;

begin
process(clk)
   begin
   if (clk'event and clk = '1') then
      if (we = '1') then
         RAM(to_integer(unsigned(addr))) <= d_in;
      end if;
   end if;
end process;
d_out <= RAM(to_integer(unsigned(addr)));
```



Tools Optimize for Timing

- If the tool recognizes an implementation that is better than another, it gets inferred
  - 32 bit parity detector
    - Written:



    - inferred