VHDL Review

Last updated 5/13/20

VHDL Basics

- VHDL
- VHSIC Hardware Description Language
- Very High Speed Integrated Circuit

VHDL Basics

- Concurrent vs Sequential Logic in HDL
  - Concurrent activities are happening all the time (in parallel)
    - Assignment: <=
    - with-select
    - when-else

```
z <= (b or c) when (d = '0') else (e and f);  -- z can change if any value
                                              -- changes, immediately
```

VHDL Basics

- Concurrent vs Sequential Logic in HDL
  - Sequential activities only happen in certain situations
    - Rising edge of clock
  - Sequential activities are identified by placing them in a "process" block
  - Process blocks are only executed when a signal in the blocks "sensitivity list" changes
  - The code in a process block is evaluated sequentially but signals are ONLY updated at the end of the process block
  - Process blocks themselves are concurrent activities

VHDL Basics

- Process block format

```
process (sensitivity list)
  begin
    actions
  end process;
```

```
label process (sensitivity list)
  begin
    actions
  end process;
```

VHDL Basics

- Basic VHDL file

```
----------------------------
--
-- basic_vhdl.vhdl
-- Created: 7/16/18
-- By: johnsontimoj
-- For: EE3921
--

-- File Overview ---
--
-- This file demonstrates basic VHDL file structure

-- File Details ---
-----------------------------
```

Header Information

```
-- Library inclusions
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

Library Inclusions

Generic

Ports

```
-- Entity definition
entity basic_vhdl is
    generic( N:     positive := 8);
    port(    i_a:   in    std_logic;
             i_b:   in    std_logic;
             i_c :  in    std_logic;
             i_p:   in    std_logic_vector(N-1 downto 0);
             i_q:   in    std_logic_vector(N-1 downto 0);
             i_r :  in    std_logic_vector(N-1 downto 0);
             o_x:   out   std_logic;
             o_y:   out   std_logic_vector(N-1 downto 0)
    );
end entity;
```

Entity

```
-- Behavioral Architecture Definition
architecture behavioral of basic_vhdl is
    signal e:   std_logic;
    signal f:   std_logic;
    signal g:   std_logic;
    signal t:   unsigned(N-1 downto 0);
    signal u:   signed(N-1 downto 0);


    Begin

        e <= i_a and i_b;
        f <= i_c nor i_a;
        g <= e xor f;
        o_x <= g and not i_p(3);

        t <= unsigned(i_p xor i_q);
        u <= signed("11" & i_a & t(4) & i_r(3 downto 2) & i_a & g);
        o_y <= (7 => '1' , 4 => u(5), 1 => i_a , 5 => t(4) , others => '1' );    -- array assignment

end architecture;
```

Architecture

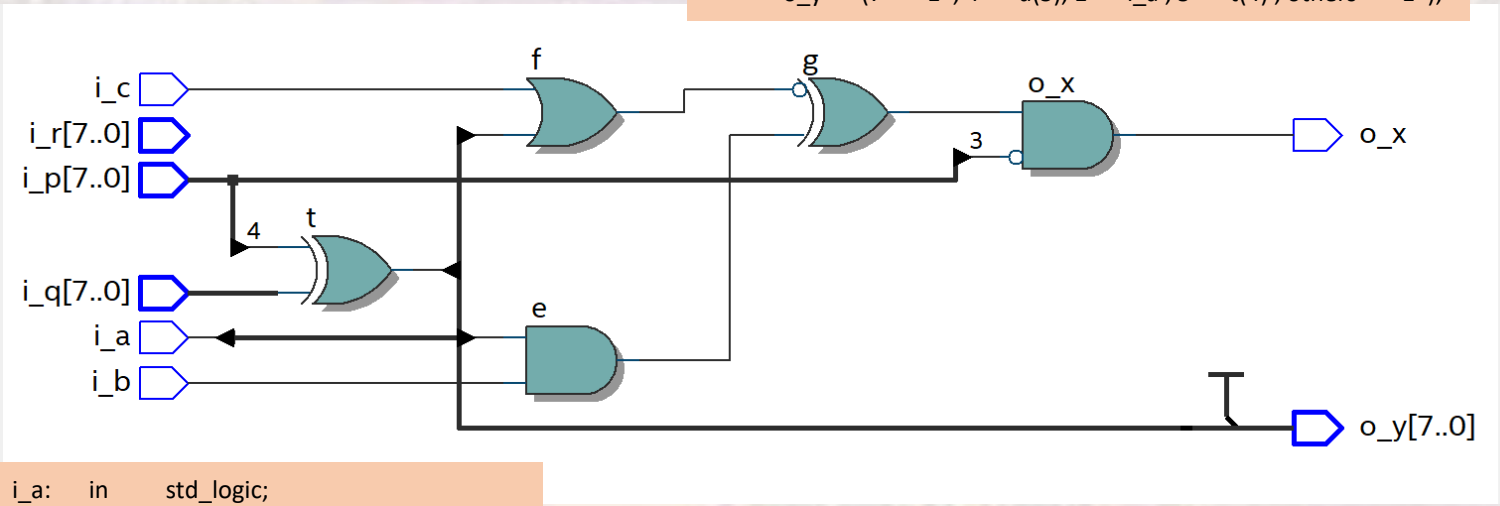
Internal signals

Architectural Definition

VHDL Basics

- Basic VHDL file

```
e <= i_a and i_b;
f <= i_c nor i_a;
g <= e xor f;
o_x <= g and not i_p(3);

t <= unsigned(i_p xor i_q);
u <= signed("11" & i_a & t(4) & i_r(3 downto 2) & i_a & g);
o_y <= (7 => '1' , 4 => u(5), 1 => i_a , 5 => t(4) , others => '1' );
```




```
port(       i_a:    in      std_logic;
            i_b:    in      std_logic;
            i_c:    in      std_logic;
            i_p:    in      std_logic_vector(N-1 downto 0);
            i_q:    in      std_logic_vector(N-1 downto 0);
            i_r :   in      std_logic_vector(N-1 downto 0);
            o_x:    out     std_logic;
            o_y:    out     std_logic_vector(N-1 downto 0)
);
```

VHDL Basics

- Basic VHDL – Structural file

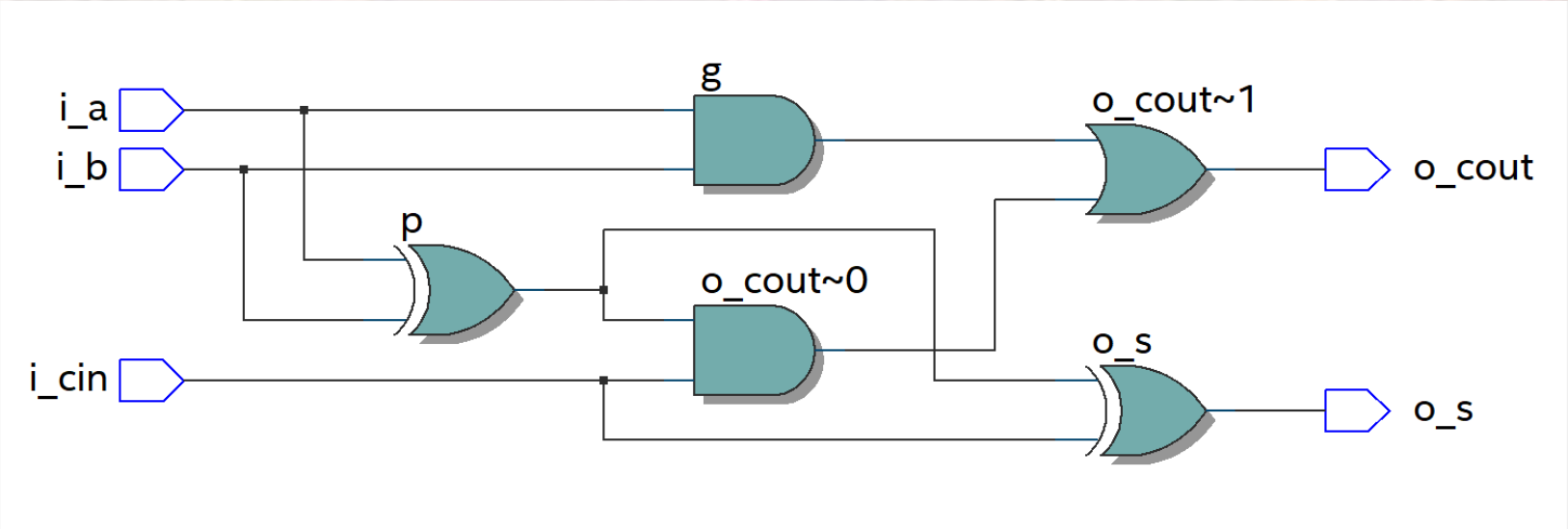
  - Full Adder

```
----------------------------------
--
--  full_adder.vhdl
--
--  by: johnsontimoj
--
--  created: 7/5/2017
--
--  version: 0.0
--
----------------------------------
----------------------------------
--
--  1 bit full adder
--
--  inputs: a, b, cin
--
--  outputs: s, cout
--
----------------------------------

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

```
entity full_adder is
    port( i_a:      in  std_logic;
          i_b:      in  std_logic;
          i_cin:    in std_logic;
          o_s:      out std_logic;
          o_cout:   out std_logic
    );
end entity;

architecture behavioral of full_adder is

    signal p: std_logic;
    signal g: std_logic;

begin

    p <= i_a xor i_b;
    g <= i_a and i_b;

    o_s <= p xor i_cin;
    o_cout <= g or (p and i_cin);

end;
```

VHDL Basics

- Basic VHDL – Structural file
  - Full Adder



VHDL Basics

- Basic VHDL – Structural fil

```
----------------------------------
--
-- adder_4bit.vhdl
--
-- by: tj
--
-- created: 7/5/2017
--
-- version: 0.0
--
----------------------------------
--
-- 4 bit adder to show cell instantiation
--
-- inputs: - a, b, cin
--
-- outputs: - sum, cout
--
----------------------------------
library IEEE;
use ieee.std_logic_1164.all;

entity adder_4bit is
    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
    );
end entity;

architecture structural of adder_4bit is
```

Top Level Entity

```
-----------------------------------
--  1 bit full adder prototype
-----------------------------------
component full_adder is
    port(   i_a:        in   std_logic;
            i_b:        in   std_logic;
            i_cin:      in std_logic;
            o_s:        out std_logic;
            o_cout:     out std_logic
    );
end component;
-----------------------------------
-- intermediate carrys mapped to co
-- with 1st stage Cout mapped to co(0) and 4th stage cout mapped to co(3)
-----------------------------------
signal co: STD_LOGIC_VECTOR(3 downto 0); -- intermediate carrys
                                         -- no vector interpretation
```

Component Prototype

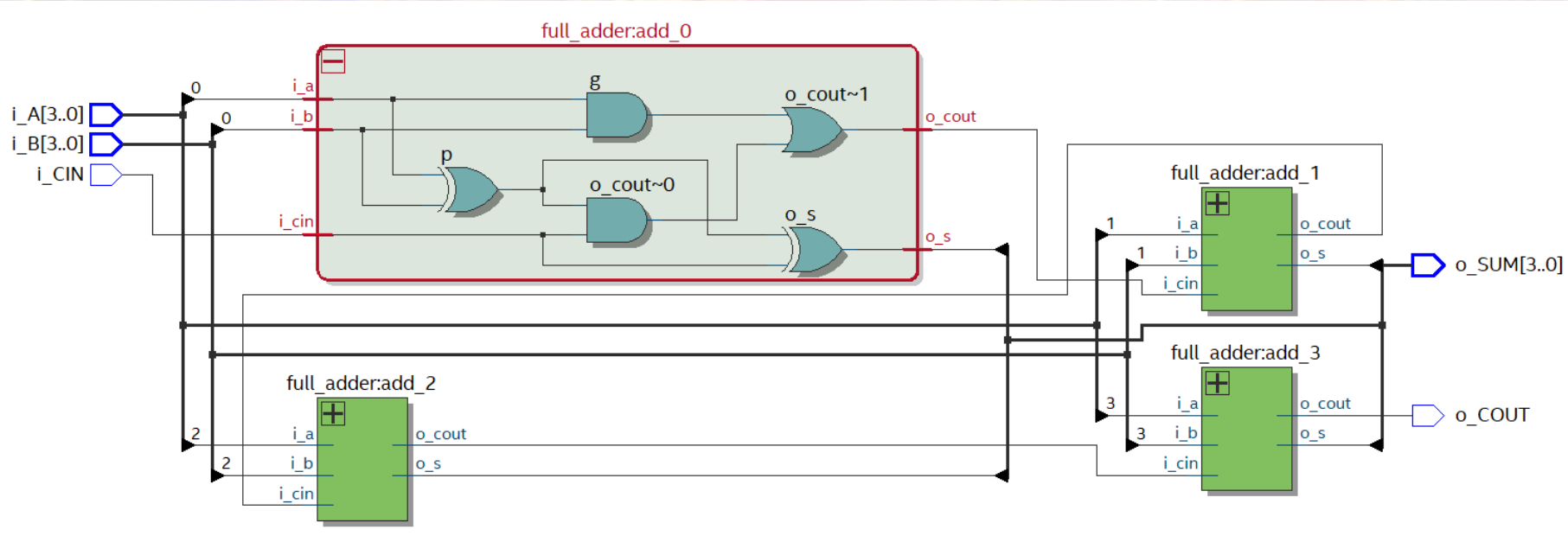
```
begin

    add_0:  full_adder port map(  i_a => i_A(0),
                                  i_b => i_B(0),
                                  i_cin => i_CIN,
                                  o_s => o_SUM(0),
                                  o_cout => co(0)
                                  );
    add_1:  full_adder port map(  i_a => i_A(1),
                                  i_b => i_B(1),
                                  i_cin => co(0),
                                  o_s => o_SUM(1),
                                  o_cout => co(1)
                                  );
    add_2:  full_adder port map(  i_a => i_A(2),
                                  i_b => i_B(2),
                                  i_cin => co(1),
                                  o_s => o_SUM(2),
                                  o_cout => co(2)
                                  );
    add_3:  full_adder port map(  i_a => i_A(3),
                                  i_b => i_B(3),
                                  i_cin => co(2),
                                  o_s => o_SUM(3),
                                  o_cout => co(3)
                                  );


    o_COUT <= co(3);

end architecture;
```

Instantiations

Explicit Port Mapping

VHDL Basics

- Basic VHDL – Structural file



VHDL Basics

- VHDL – Selection
  - with-select
    - Choose a value when a certain situation exists

```
with decision_signal select result_signal <=        -- exhaustive list
      result_value when decision_value,
      result_value when decision_value,
      result_value when decision_value,
      result_value when decision_value;
```

Limitation: Only one result signal

VHDL Basics

- VHDL – Selection
  - with-select

Exhaustive List

```
with inA select outA <=   "0100" when "01",
                          "0010" when "10",
                          "0001" when "11",
                          "1010" when "00",
                          "0000" when others;
```

Partial List

```
with inA select outA <=   "0100" when "01",
                          "0010" when "10",
                          "0000" when others;
```

Partially Common Result

```
with inA select outA <=   "0100" when "01",
                          "0010" when "10",
                          "0001" when ("11" or "00"),
                          "0000" when others;
```

Complex Selection

```
with (inA and inB) select outA <= "0100" when "01",
                                  "0010" when "10",
                                  "0001" when "00",
                                  "0000" when others;
```

VHDL Basics

- VHDL – Selection
  - with-select

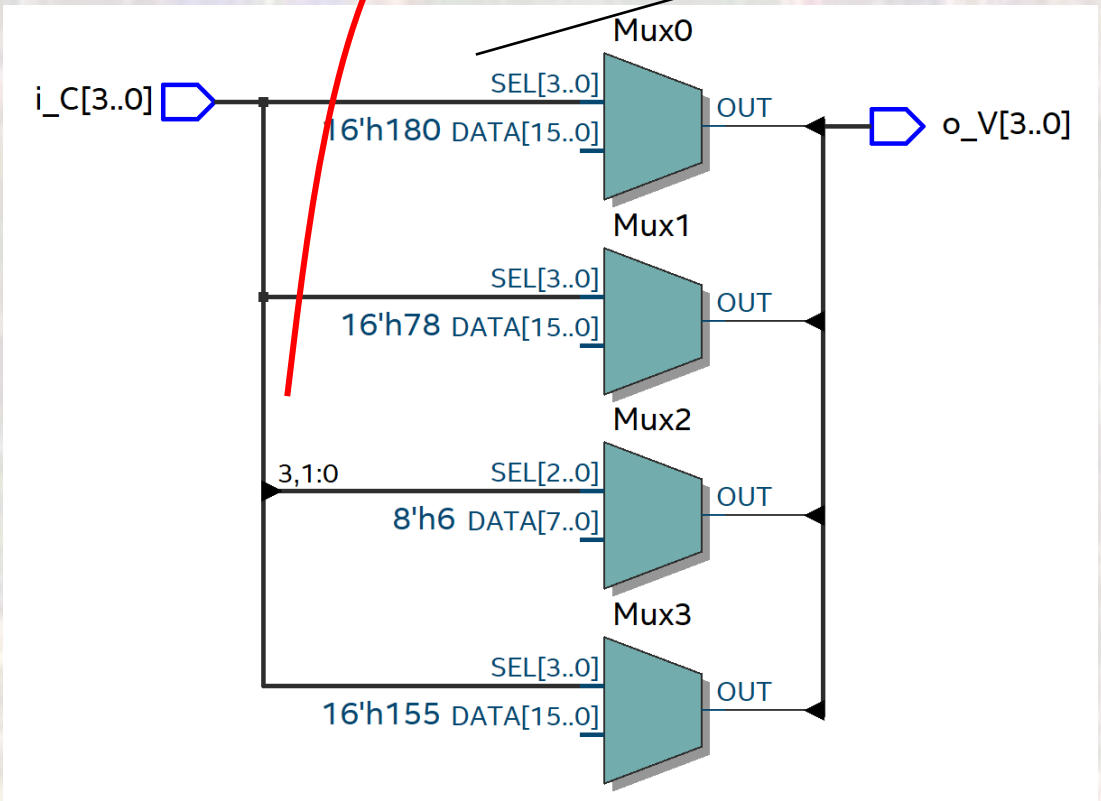
```
----------------------------------------
--
-- with_select.vhdl
--
-- created 7/5/2018
-- tj
--
-- rev 0
----------------------------------------
--
-- with-select example
--
----------------------------------------
--
-- Inputs: C
-- Outputs: V
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity with_select is
    port (
        i_C:    in std_logic_vector(3 downto 0);
        o_V:    out std_logic_vector(3 downto 0)
    );
end entity;
```

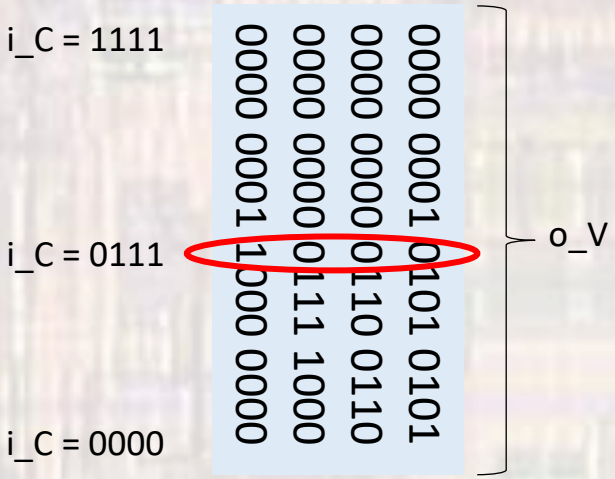
```
architecture behavioral of with_select is

begin
-------------------------------------------
-- Code to show mux
-------------------------------------------
    with i_C select o_V <=
        "0001" when "0000",
        "0010" when "0001",
        "0011" when "0010",
        "0100" when "0011",
        "0101" when "0100",
        "0110" when "0101",
        "0111" when "0110",
        "1000" when "0111",
        "1001" when "1000",
        "0000" when others;

end behavioral;
```

VHDL Basics

- VHDL – Selection
  - with-select

Mux3 0000 0001 0101 0101
0000 0000 0110 0110
0000 0000 0111 1000
Mux0 0000 0001 1000 0000






```
"0111" when "0110",
"1000" when "0111",
```

VHDL Basics

- VHDL – Selection
  - when-else
    - Choose a value when a certain situation exists

```
result_signal <=    result_value when decision_signal = decision_value else
                    result_value when decision_signal = decision_value else
                    result_value when decision_signal = decision_value else
                    result_value;
```

Limitation: Only one result signal

VHDL Basics

- VHDL – Selection
  - when-else

Exhaustive List

```
outA <= "1000" when inA = "00" else
        "0100" when inA = "01" else
        "0010" when inA = "10" else
        "0010" when inA = "11" else
        "0001";
```

Partial List

```
outA <= "1000" when inA = "00" else
        "0100" when inA = "01" else
        "0001";
```

Partially Common Result

```
outA <= "1000" when inA = "00" else
        "0100" when inA = "01" else
        "0010" when inA = ("10"  or "11") else
        "0001";
```

Complex Selection

```
outA <= "1000" when (inA or inB) = "00" else
        "0100" when (inA or inB) = "01" else
        "0010" when (inA or inB) = "10" else
        "0001";
```

VHDL Basics

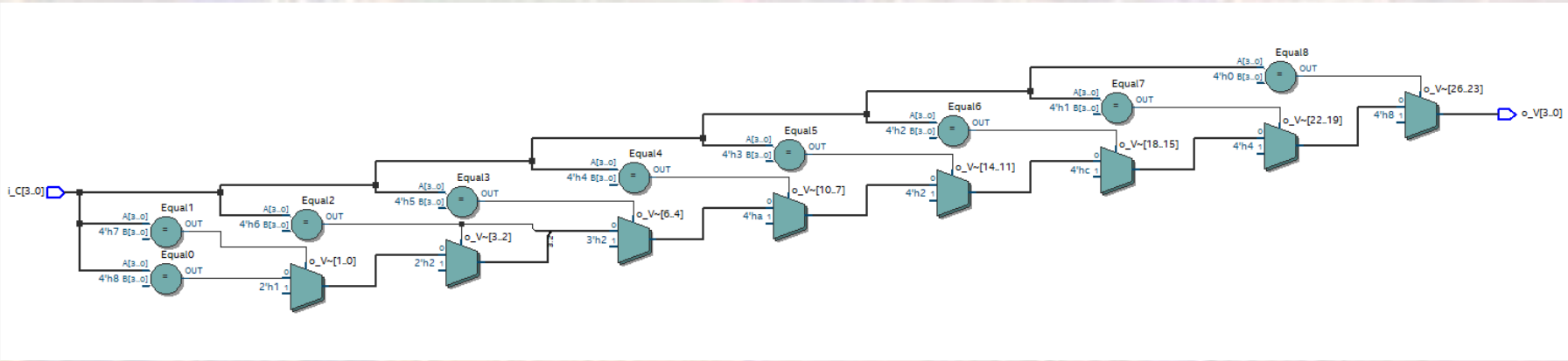
- VHDL – Selection
  - when-else

```
----------------------------------------
--
-- when_else.vhdl
--
-- created 7/5/2018
-- tj
--
-- rev 0
----------------------------------------
--
-- when-else example
--
----------------------------------------
--
-- Inputs: C
-- Outputs: V
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity when_else is
    port (
        i_C:    in std_logic_vector(3 downto 0);
        o_V:    out std_logic_vector(3 downto 0)
    );
end entity;
```

```
architecture behavioral of when_else is

begin

    o_V <=      "0001" when i_C = "0000" else
                "0010" when i_C = "0001" else
                "0011" when i_C = "0010" else
                "0100" when i_C = "0011" else
                "0101" when i_C = "0100" else
                "0110" when i_C = "0101" else
                "0111" when i_C = "0110" else
                "1000" when i_C = "0111" else
                "1001" when i_C = "1000" else
                "0000";
end behavioral;
```

VHDL Basics

- VHDL – Selection
  - when-else



VHDL Basics

- VHDL – Selection
  - if-else
    - Choose a value when a certain situation exists

```
if (decision_signal = decision_value_X) then
  result_signal_1 <= result_value_1;
  result_signal_2 <= result_value_2;
  result_signal_3 <= result_value_3;
elsif (decision_signal = decision_value_Y) then
  result_signal_1 <= result_value_1a;
  result_signal_2 <= result_value_2b;
  result_signal_3 <= result_value_3c;
else
  result_signal_1 <= result_value_a;
  result_signal_2 <= result_value_b;
  result_signal_3 <= result_value_c;
end if;
```

Typically use the same decision signal
Exception – creating registers (Flip-Flops)

Limitation: Must be used in a process

VHDL Basics

- VHDL – Selection
  - if-else

Exhaustive List

```
if(inA = "00") then
   outW <= "1000";
elsif(inA = "01") then
   outW <= "0100";
elsif(inA = "10") then
   outW <= "0010";
elsif(inA = "11") then
   outW <= "0011";
else
   outW <= "0000";
end if;
```

Partial List

```
if(inA = "00") then
   outX <= "1000";
elsif(inA = "01") then
   outX <= "0100";
else
   outX <= "0000";
end if;
```

Partially Common Result

```
if(inA = "00") then
   outY <= "1000";
elsif(inA = "01") then
   outY <= "0100";
elsif((inA = "10") or (inA = "11")) then
   outY <= "0110";
else
   outX <= "0000";
end if;
```

Complex Selection

```
if((inA or inB) = "00") then
   outZ <= "1000";
elsif((inA or inB)  = "01") then
   outZ <= "0100";
else
   outZ <= "0000";
end if;
```

VHDL Basics

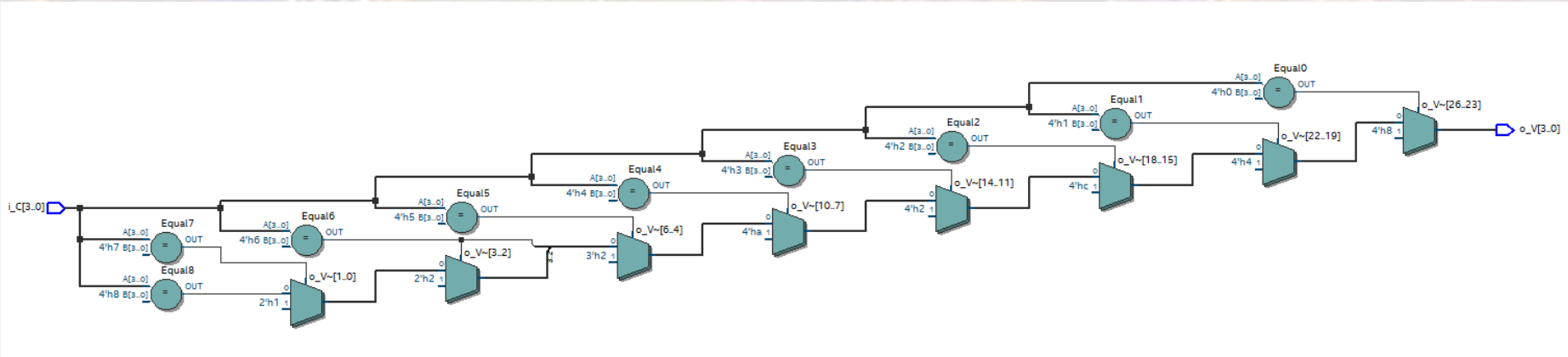
- VHDL – Selection
  - if-else

```
----------------------------------------
--
-- if_else.vhdl
--
-- created 7/5/2018
-- tj
--
-- rev 0
----------------------------------------
--
-- if-else example
--
----------------------------------------
--
-- Inputs: C
-- Outputs: V
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity if_else is
     port (
               i_C:     in std_logic_vector(3 downto 0);
               o_V:     out std_logic_vector(3 downto 0)
      );
end entity;
```

```
architecture behavioral of if_else is

begin
     process(i_C)
     begin
          if(i_C = "0000") then
               o_V <= "0001";
          elsif(i_C = "0001") then
               o_V <= "0010";
          elsif(i_C = "0010") then
               o_V <= "0011";
          elsif(i_C = "0011") then
               o_V <= "0100";
          elsif(i_C = "0100") then
               o_V <= "0101";
          elsif(i_C = "0101") then
               o_V <= "0110";
          elsif(i_C = "0110") then
               o_V <= "0111";
          elsif(i_C = "0111") then
               o_V <= "1000";
          elsif(i_C = "1000") then
               o_V <= "1001";
          else
               o_V <= "0000";
          end if;
     end process;

end behavioral;
```

VHDL Basics

- VHDL – Selection
  - if-else



VHDL Basics

- VHDL – Selection
  - case
    - Choose a value when a certain situation exists

```
case decision_signal is
  when decision_value_X =>  result_signal_1 <= result_value_1;
                            result_signal_2 <= result_value_2;
                            result_signal_3 <= result_value_3;
  when decision_value_Y =>  result_signal_1 <= result_value_1a;
                            result_signal_2 <= result_value_2b;
                            result_signal_3 <= result_value_3c;
  when others =>            result_signal_1 <= result_value_a;
                            result_signal_2 <= result_value_b;
                            result_signal_3 <= result_value_c;
end case;
```

Limitations: Must be used in a process
Only one decision signal

VHDL Basics

- VHDL – Selection
  - case

Exhaustive List

```
case inA is
   when "00" => outW <= "1000";
   when "01" => outW <= "0100";
   when "10" => outW <= "0010";
   when "11" => outW <= "0011";
   when others => outW <= "0000";
end case;
```

Partial List

```
case inA is
   when "00" => outX <= "1000";
   when "01" => outX <= "0100";
   when others => outX <= "0000";
end case;
```

Partially Common Result

```
case inA is
   when "00" => outY <= "1000";
   when "01" => outY <= "0100";
   when ("10“ or “11”) => outY <= "0010";
   when others => outY <= "0000";
end case;
```

Complex Selection

```
case (inA or inB) is
   when "00" => outW <= "1000";
   when "01" => outW <= "0100";
   when "10" => outW <= "0010";
   when "11" => outW <= "0011";
   when others => outW <= "0000";
end case;
```

VHDL Basics

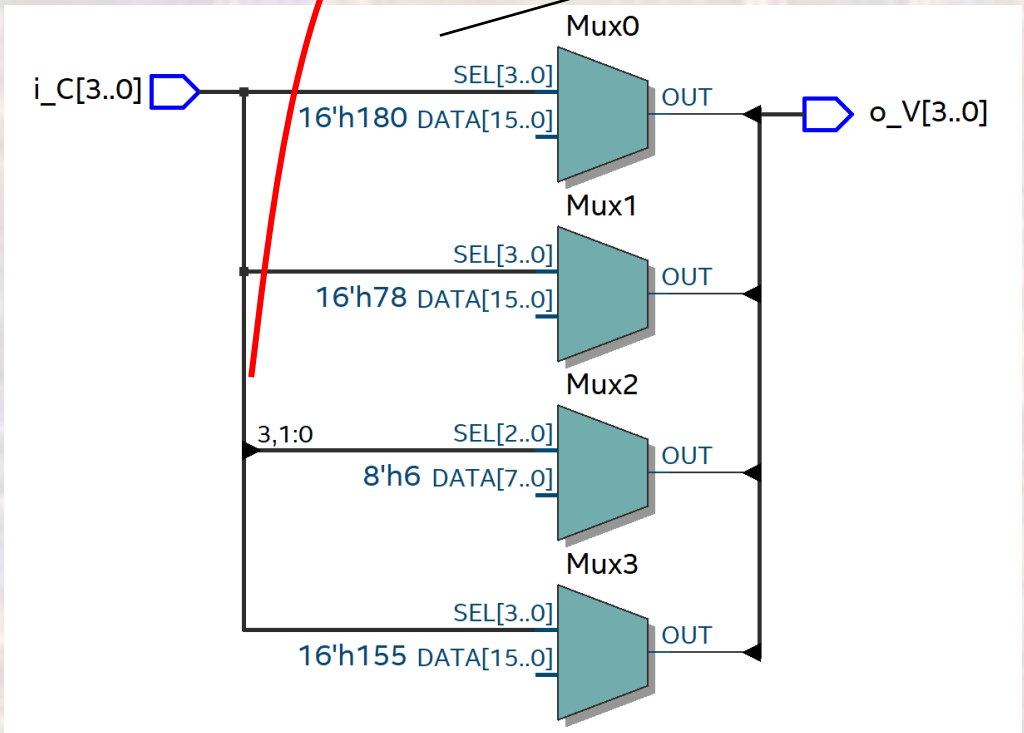
- VHDL – Selection
  - case

```
----------------------------------------
--
-- case_ex.vhdl
--
-- created 7/5/2018
-- tj
--
-- rev 0
----------------------------------------
--
-- case example
--
----------------------------------------
--
-- Inputs: C
-- Outputs: V
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity case_ex is
    port (
        i_C:    in std_logic_vector(3 downto 0);
        o_V:    out std_logic_vector(3 downto 0)
);
end entity;
```

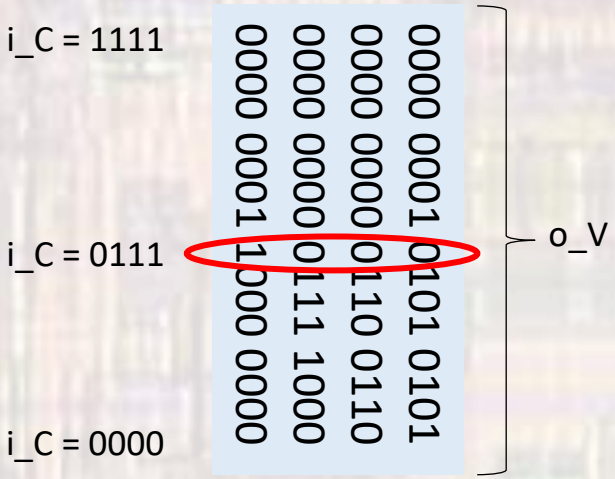
```
architecture behavioral of case_ex is

begin
    process(all)
    begin
        case i_C is
            when "0000" => o_V <= "0001";
            when "0001" => o_V <= "0010";
            when "0010" => o_V <= "0011";
            when "0011" => o_V <= "0100";
            when "0100" => o_V <= "0101";
            when "0101" => o_V <= "0110";
            when "0110" => o_V <= "0111";
            when "0111" => o_V <= "1000";
            when "1000" => o_V <= "1001";
            when others => o_V <= "0000";
        end case;
    end process;

end behavioral;
```

VHDL Basics

- VHDL – Selection
  - case



Mux3 0000 0001 0101 0101
0000 0000 0110 0110
0000 0000 0111 1000
Mux0 0000 0001 1000 0000




```
when "0110" => o_V <= "0111";
when "0111" => o_V <= "1000";
```

VHDL Basics

- Registers (Flip-Flops) are recognized by a pre-defined VHDL template

```
process (clock signal)
  begin
    if(clock edge detection) then
      actions
    end if;
  end process;
```

note:
- here the else is not required because the synthesizer recognizes the edge detection

- you can include an else for clarity

```
process (clk)
  begin
    if(clk'event and clk = 1) then
      q <= d;
    end if;
  end process;
```

```
process (clk)
  begin
    if(rising_edge(clk)) then
      q <= d;
    end if;
  end process;
```

2008 release

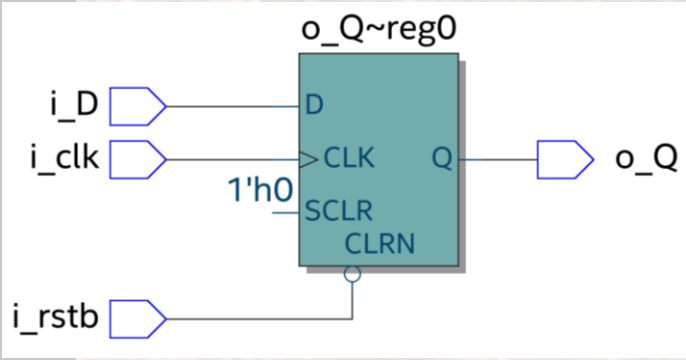
VHDL Basics

- D-FF w/ asynchronous rstb

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

entity dff_a 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 dff_a 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 architecture;
```

Note addition of rstb to the sensitivity list



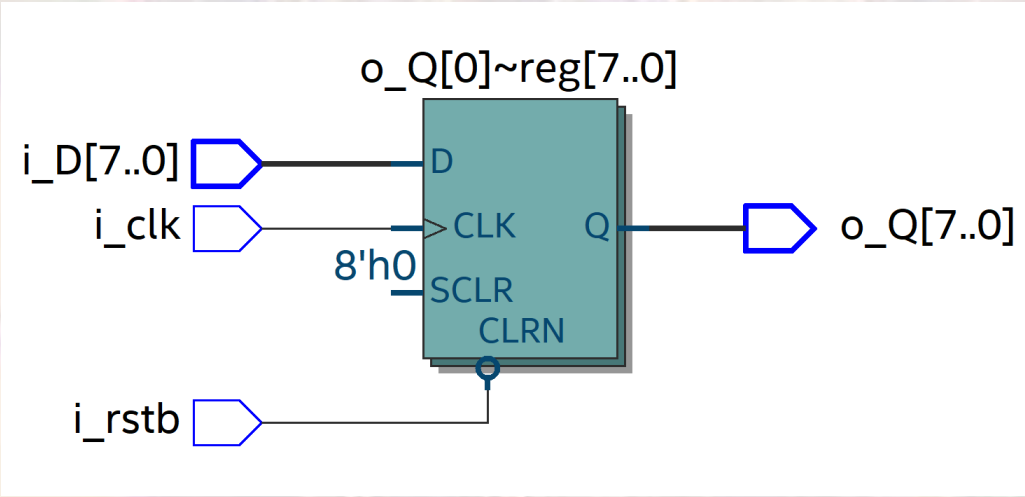
Most of our designs will use DFFs with an asynchronous reset

Data Path designs will use DFFs with no reset

VHDL Basics

- 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;
------------------------------
```



VHDL Basics

- Warning – Warning – Warning
  - Most (maybe all) of the times you do not complete an if-else with an else a latch will be created
  - All cases must be covered
  - We do not want latches - EVER