VHDL Intermediate

Tarek Eldeeb, Valeo Expert

Prerequisites

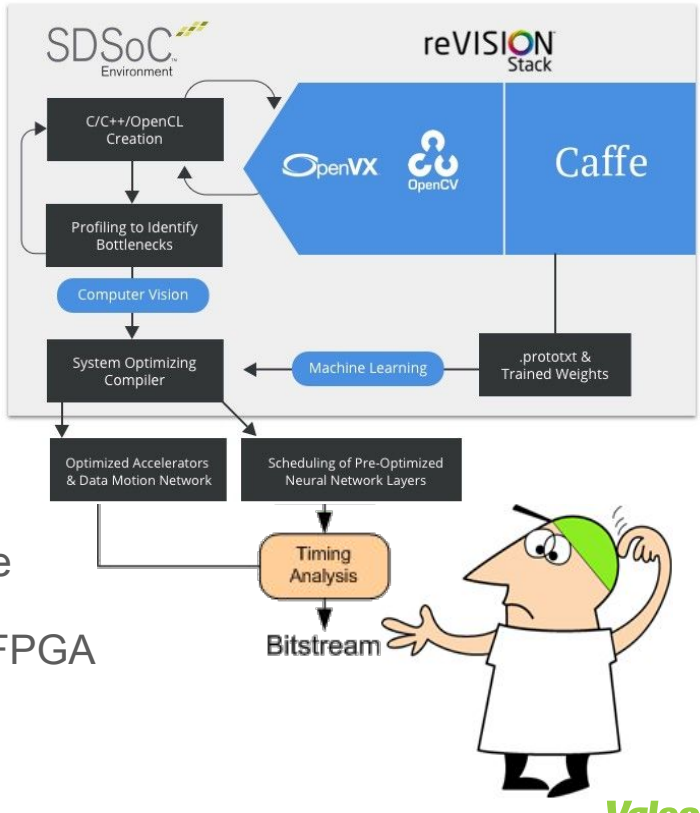
Knowledge

- Basic VHDL Syntax knowledge
- Basic FPGA Technology knowledge
  - Udemy Training

Installed Tools

- Xilinx tools (ISE/XPS 14.7) should be preinstalled with debugger drivers.
- Xilinx development kit with Spartan FPGA
  - ISE/XPS Download

SDSoc: Xilinx Tutorials - Demo Video



Training Outline

Day #1

- Revision: VHDL I
- Pipelining
- Static Timing Analysis
- Exercise

Day #2

- FPGA Advanced:
  - IO banks and Clocks
  - Special blocks
- AXI Bus: Specs and Applications
- Exercise

Day #3: Practical Workshop

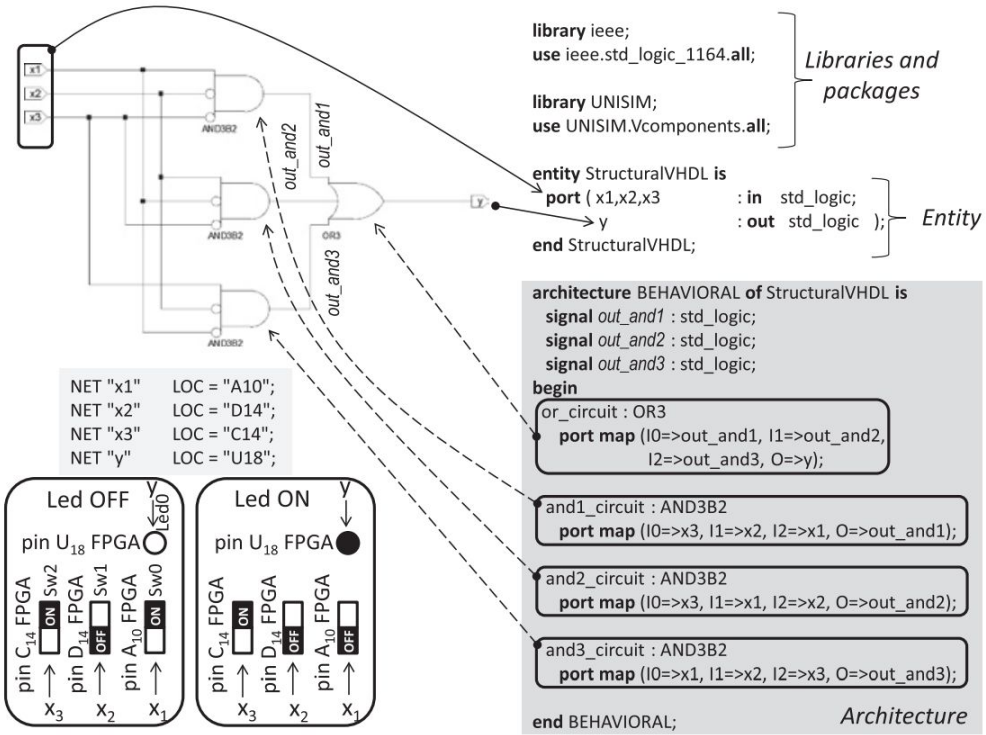
- Analyse Current Design
- Redesign for expandability

Revision

Rule #1: We are describing a HW circuit!

When you are describing hardware in VHDL, you are only describing the behaviour. The actual circuit will be synthesized (by the tools, eg. Xilinx ISE) to gates. The FPGA then implements the gates. The FPGA does NOT execute the VHDL code directly!!

Revision




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

library UNISIM;
use UNISIM.Vcomponents.all;
```

Libraries and packages

```
entity StructuralVHDL is
  port ( x1,x2,x3          : in  std_logic;
         y                 : out std_logic );
end StructuralVHDL;
```

Entity

```
architecture BEHAVIORAL of StructuralVHDL is
  signal out_and1 : std_logic;
  signal out_and2 : std_logic;
  signal out_and3 : std_logic;
begin
  or_circuit : OR3
    port map (I0=>out_and1, I1=>out_and2,
              I2=>out_and3, O=>y);

  and1_circuit : AND3B2
    port map (I0=>x3, I1=>x2, I2=>x1, O=>out_and1);

  and2_circuit : AND3B2
    port map (I0=>x3, I1=>x1, I2=>x2, O=>out_and2);

  and3_circuit : AND3B2
    port map (I0=>x1, I1=>x2, I2=>x3, O=>out_and3);

end BEHAVIORAL;
```

Architecture

Revision

Declaration of libraries:

```
library . . . ;
use . . . ;
. . . . . .
```

Description of entity

```
entity <name of entity> is
Declaration of generic parameters
Declaration of ports
end <name of entity>;
```

Entity declarative part

Description of architecture

```
architecture <name of architecture> of <name of entity> is
Declaration of signals
Declaration of constants
Declaration of types
Declaration of components
Declaration of functions (with bodies)
Declaration of procedures (with bodies)
Declaration of shared variables
begin

Body of architecture

end <name of architecture>;
```

Architecture declarative part

Connected components which are either library primitives or previously designed circuits from:

a) HDL specifications
b) schematic specifications
c) IP cores

- Structural
- Mixed
- Behavioral

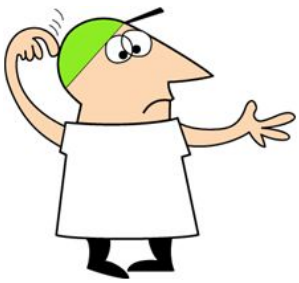
Examine VHDL code and component instantiation for schematic entries

Concurrent constructions:

a) signal assignments
b) VHDL processes

Xilinx ISE *Design* tab

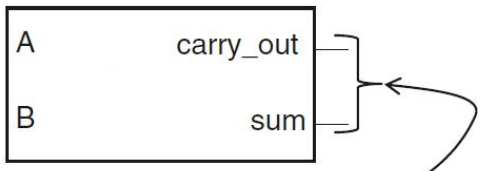
Design Utilities
. . . . . .
View HDL Functional Model
View HDL Instantiation Template



Revision

```
1  library IEEE;
   use IEEE.std_logic_1164.all;
```
use of libraries and packages

HA



Interface

```
entity half_adder is
 port (  A         : in STD_LOGIC;
         B         : in STD_LOGIC;
 2       carry_out : out STD_LOGIC;
         sum       : out STD_LOGIC);
end half_adder;
```

Architecture

```
architecture half_adder_behavior of half_adder is
begin
 3      sum       <= A xor B;
        carry_out <= A and B;
end half_adder_behavior;
```

A	B	carry_out	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Revision

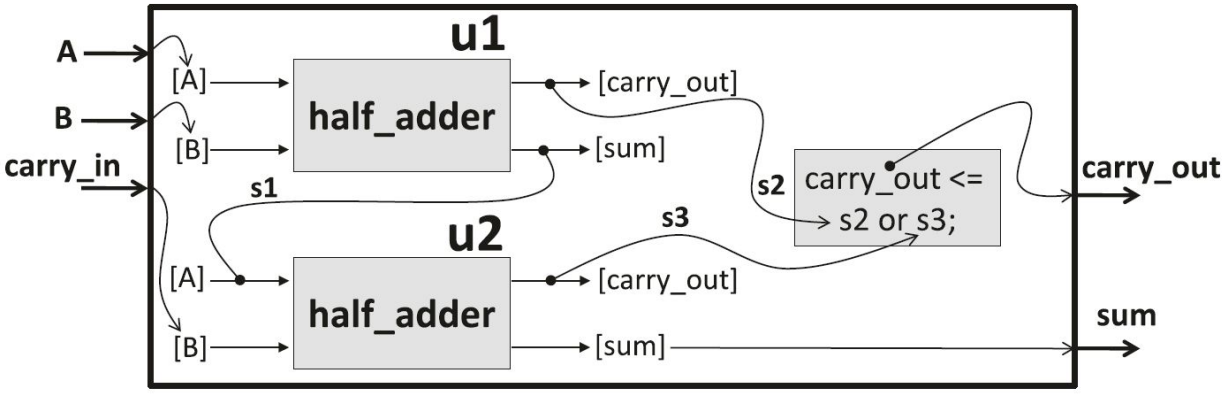
```
library IEEE;
use IEEE.std_logic_1164.all;
entity FULLADD is
port (  A, B, carry_in      : in  std_logic;
        sum, carry_out      : out std_logic );
end FULLADD;
architecture STRUCT of FULLADD is
signal s1, s2, s3 : std_logic;
begin
   u1: entity work.half_adder
               port map(A, B, s2, s1);
   u2: entity work.half_adder
               port map(s1, carry_in, s3, sum);
   carry_out <= s2 or s3;
end STRUCT;
```

Example of positional association

```
library IEEE;
use IEEE.std_logic_1164.all;
entity half_adder is
  port (
    A         : in STD_LOGIC;
    B         : in STD_LOGIC;
    carry_out : out STD_LOGIC;
    sum       : out STD_LOGIC    );
end half_adder;
architecture half_adder_behavior of half_adder is
begin
    sum       <= A xor B;
    carry_out <= A and B;
end half_adder_behavior;
```

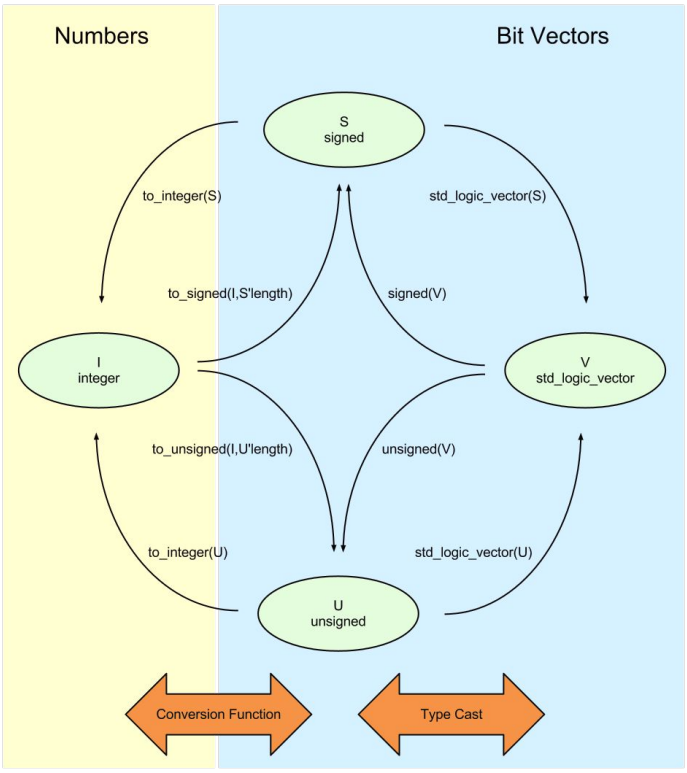
A	B	carry_in	carry_out	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

FULLADD



TIPs: Packages/Types

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_textio.all;
  use IEEE.std_logic_arith.all;
  use IEEE.numeric_bit.all;
  use IEEE.numeric_std.all;
  use IEEE.std_logic_signed.all;
  use IEEE.std_logic_unsigned.all;
  use IEEE.math_real.all;
  use IEEE.math_complex.all;
```



What to use?

```
library ieee;
  use ieee.std_logic_1164.all;   -- UX01ZWLH
  use ieee.numeric_std.all;   -- un/signed
  use ieee.std_logic_arith.all; -- bad, old!
```


TIPs: Attributes

- signal'event
- signal'left/right
- signal'low/high
- signal'length
- signal'range
- signal'reverse_range

This code is NOT accepted: VHDL is hard-typed

```
b(0 to 7) <= a(7 downto 0)
```

```
Sol#1
function reverse_any_vector (a: in std_logic_vector)
return std_logic_vector is
  variable result: std_logic_vector(a'RANGE);
  alias aa: std_logic_vector(a'REVERSE_RANGE) is a;
  begin
    for i in aa'RANGE loop
      result(i) := aa(i);
    end loop;
    return result;
  end; -- function reverse_any_vector
```

```
Sol#2
signal a : std_logic_vector(0 to 7);
signal b : std_logic_vector(7 downto 0);
..
for i in a'range generate
  b(i) <= a(i)
end generate;
```

TIPs: Process

Processes should be broken till each has a **SINGLE** type:

1. Pure Combinational
2. Pure Sequential
3. Sequential with async reset

TIPs: Process

1- Pure Combinational

- Rule 1: Every input (that can affect the output(s)) must be in the sensitivity list.
- Rule 2: Every output must be assigned a value for every possible combination of the inputs

```
process (A, B)
begin
   if (SEL = '0') then
      Y <= A;
   else
      Y <= B;
   end if;
end process;
```

```
process (SEL, A, B)
begin
      if (SEL = '0') then
            Y <= A;
      end if;
end process;
```

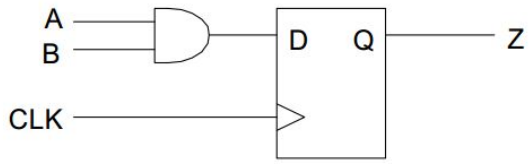
TIPs: Process

2- Pure Sequential

- Rule 1: Only the clock should be in the sensitivity list
- Rule 2: Only signals that change on the same edge of the same clock should be part of the same process

rising_edge(clk)

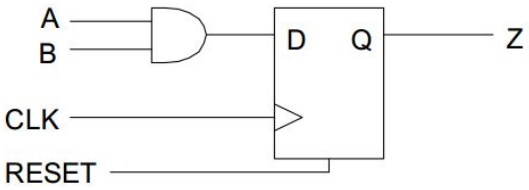
```
process (CLK)
begin
  if (CLK'event and CLK='1') then
     Z <= A and B;
  end if;
end process;
```



TIPs: Process

3- Sequential with async reset

```
process (CLK, RESET)
begin
  if (RESET = '1') then
     Z <= '0';
  elsif (CLK'event and CLK='1') then
     Z <= A and B;
  end if;
end process;
```



Quiz Time :-)

```
entity TestCombProc is
port ( clk : in std_logic;
       BTND : in std_logic;
       led : out std_logic_vector(1 downto 0) := (others=> '0') );
end TestCombProc;
architecture Behavioral of TestCombProc is
       Signal B : std_logic := '1';
       Signal A : std_logic := '0';
begin
test_assign: process(clk)
       begin
       if rising_edge(clk) then
              if BTND = '1' then led <= A & B;
              else
                     if B = '1' then
                            A <= B;
                            B <= A;
                            led(1) <= A;
                            led(0) <= B;
                     end if;
              end if;
       end if;
end process test_assign;
end Behavioral;
```

Quiz : Answered