Combinational Circuits

Introduction to VHDL for FPGAs

I. INTRODUCTION :

VHDL stands for VHSIC (very high speed integrated circuits) Hardware Description Language. Sported by US Department of Defense and later became IEEE standard. There are many revisions of VHDL [1987 and 1993]. VHDL can describe and model really complex systems. To get some understanding of this language, an example design will be used. This is a Half Adder that has following block diagram and truth table. This is a simple combinational circuit where output values are solely depends on input values. There is no memory component where output value stored.

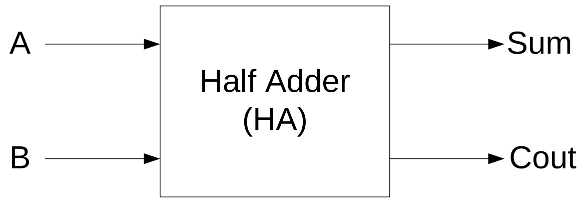


Figure 1. Half Adder Block Diagram

Table 1. Half Adder Truth Table

A	B	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Using Boolean algebra Sum and Cout can be written in terms of input :

$$\mathbf{Sum} = \overline{\mathbf{A}}\,.\,\mathbf{B} + \mathbf{A}\,.\,\overline{\mathbf{B}} = \mathbf{A} \oplus \mathbf{B}$$

$$\mathbf{Cout} = \mathbf{A}\,.\,\mathbf{B}$$

Using the Boolean equations Sum [1] and Cout [2] the schematic would be,

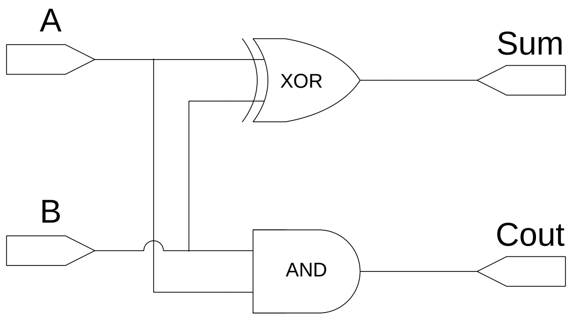


Figure 2. Half Adder Schematic

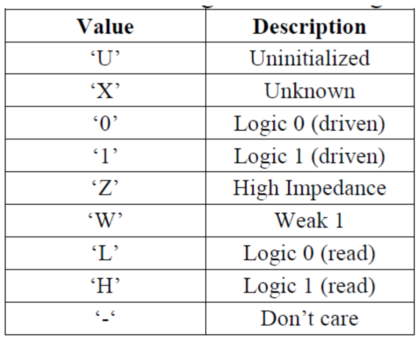
1-) Gate level description of Half Adder is

```
library ieee;                          // Due to special std_logic is used the std_logic_1164 needs to be used.
use ieee.std_logic_1164.all;           // You can use many different data type, functions and operators.
entity HA is                           // This is the entity declaration. You must have an entity name.
port      (                            //  Input and output must be declared in entity. This is description of block
          A, B           : in std_logic;     // diagram of HA that is given in figure 1. The std_logic data type is used
          Sum, Cout      : out std_logic     // for input and output.
          );                           // I/O port declaration* : signal_name1, signal name2,…. : mode data_type
end HA;                                // Entity must be ended.

architecture gate_HA of HA is          // Operation of the circuit (what schematic) is described in here. Architecture
                                       // has a unique name (gate_HA). All of the internal signals, constants, and
begin                                  // components must be declared before the "begin" statement.
                                       // Description of operation is between  "begin" and "end" .
          Sum <= A xor B;              // These statements are concurrent statements.
          Cout <= A and B;             // The left side of the statement is output and when one of the inputs (A or B) is
                                       // changed after a delay, the new value of Cout is assigned.
end gate_HA;                           // end the begin statement and name of the architecture.
```

*   Data Types : VHDL is a strongly typed language where data type must be declared correctly. One of the most commonly used type is std_logic.

std_logic : This type has following values :



Value	Description
'U'	Uninitialized
'X'	Unknown
'0'	Logic 0 (driven)
'1'	Logic 1 (driven)
'Z'	High Impedance
'W'	Weak 1
'L'	Logic 0 (read)
'H'	Logic 1 (read)
'-'	Don't care

Figure 3.std_logic values

Some of the common *std_logic* deceleration ;

```
Sum     : out std_logic_vector (7 downto 0)        //Sum is  8-bit output (MSB = A(7), LSB =A(0)) .

Sum     : out std_logic_vector (0 downto 7)        //Sum is  8-bit output (MSB = A(0), LSB =A(7)) .
```

2-) Structural description of Half Adder is

Most of the systems are made of smaller subsystems. You can build a large system by *instantiation* of smaller systems. Designing large system using *component instantiation* is known as *structural description*. The HA as structural description,

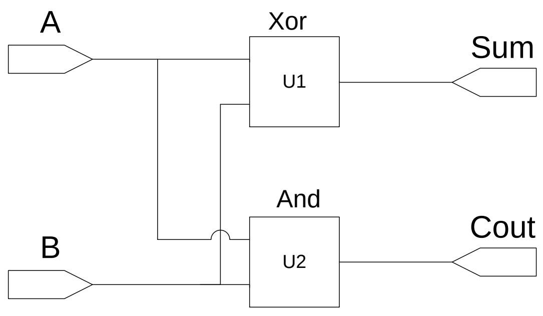


Figure 4. Structural Description of Half Adder

```
library ieee;
use ieee.std_logic_1164.all;
entity and2 is
port  (
          A, B        : in std_logic;
          Z           : out std_logic
     );
          end and2;
architecture dataflow of and2 is
begin
Z <= A and B;
end dataflow;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
port  (
          A, B        : in std_logic;
          Z           : out std_logic
     );
          end xor2;
architecture dataflow of and2 is
begin
Z <= A xor B;
end dataflow;
```

Figure 5. Description of AND and XOR gates.

```
-- Structural Description of HA using "component instantiation".
architecture struct_HA of HA is                    // name of the architecture.

component xor2                                     // Component declaration of XOR2 gate.
port(a,b: in std_logic; z: out std_logic);         // Port declaration is similar to entity declaration above.
end component;                                     // end component

component and2                                     // Component declaration of AND2 gate
port(a,b: in std_logic; z: out std_logic);         // Port declaration is similar to entity declaration above.
end component;                                     // end component

begin
U1 : xor2 port map(a => A, b => B, z => Sum);      // instantiate component XOR2
U2 : and2 port map(a => A, b => B, z => Cout);     // instantiate component AND2
End struct_HA;
```

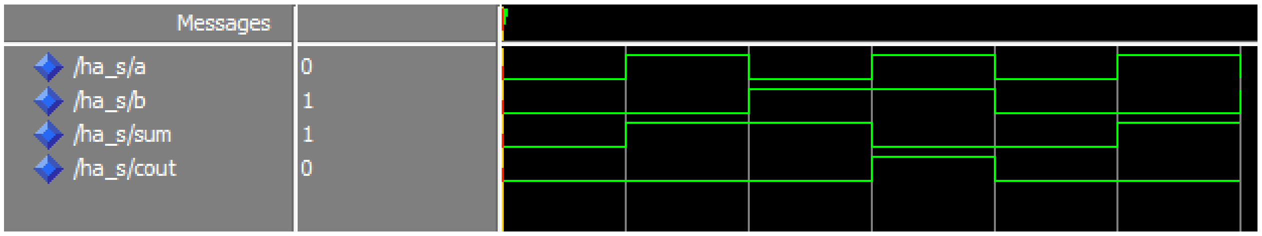


Figure 6. Simulation results.

```
-- Structural Description of HA using "entity instantiation".
architecture struct_HA of HA is
begin
U1 : entity xor2 port map(a => A, b => B, z => Sum);
U2 : entity and2 port map(a => A, b => B, z => Cout);
End struct_HA;
```

II. RTL (Register Transfer Level) Combinational Circuits:

RTL is useful for computer design where machine language instructions are defined as sequence of more primitive steps involving loading, storing, combining and testing registers.

2.1. RTL Components

2.1.1. Operators and Data Types

Table 2.1 Operators and Data Types VHDL-93 and IEEE std_logic_1164

Operator	Description	Data type of operands	Data type of result
x ** y	Exponent	integer	integer
x – y	Subtraction	Integer type for constant and array boundaries, not synthesis	
x * y	Multiplication		
x / y	Division		
x + y	Addition		
x & b	concatenation	1-D array element	1-D Array
x = y	Equal to	any	Boolean
x /= y	Not equal to		
x < y	Less than	Scalar or 1-D Array	Boolean
x <= y	Less than or equal to		
x > y	Greater than		
x >= y	Greater than or equal to		
not x	negation	Boolean, std_logic, std_logic_vector	
x and y	And		
x or b	Or		
x xor b	Xor		

Table 2.2 Operators and Data Types IEEE numeric_std

Operator	Description	Data type of operands	Data type of result
x + y	Addition	Unsigned, natural signed, integer	unsigned signed
x - y	Subtraction		
x * y	Multiplication		
x = y	Equal to	Unsigned, natural signed, integer	boolean
x /= y	Not equal to		
x < y	Less than		
x <= y	Less than or equal to		
x > y	Greater than		
x >= y	Greater than or equal to		

Table 2.3 Type conversion between std_logic_vector and numeric data types

Data Type of x	To data type	Conversion function/type casting
Unsigned, signed	std_logic_vector	std_logic_vector(x)
signed, std_logic_vector	unsigned	unsigned(x)
unsigned, std_logic_vector	signed	signed(x)
Unsigned, signed	integer	to_integer(x)
Natural	unsigned	to_unsigned(x, size)
integer	signed	to_signed(x, size)

Concatenation [1]:

```
signal  a1          :       std_logic;
signal  a4          :       std_logic_vector (3 downto 0);
signal  b8, c8, d8  :       std_logic_vector (7 downto 0);
…..
b8      <=      a4 & a4;
c8      <=      a1 & a1 & a4 & “00”;
d8      <=      b8 (3 downto 0) & c8 (3 downto 0);
```

III. Conditional Signal assignments:

3.1. When-else

Signal_name <= expression_1 when Boolean_expression1 else
Expression_2 when Boolean_expression2 else
.
.
Expression_i

Example : (3-1 Mux)

Table 3.1 Truth table for 3-1 MUX

A	B	C	Sel	Z
A	X	X	00	A
X	B	X	01	B
X	X	C	10	C
X	X	X	11	Z

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

Entity MUX_When is
port (
 A , B, C : in std_logic;
 Sel : in std_logic_vector (1 downto 0);
 Z : out std_logic );
end MUX_When;

Architecture behavioral of MUX_When is
begin
Z <= A when Sel="00" else
    B when Sel="01" else
    C when Sel="10" else
    'Z' ;
end behavioral;
```

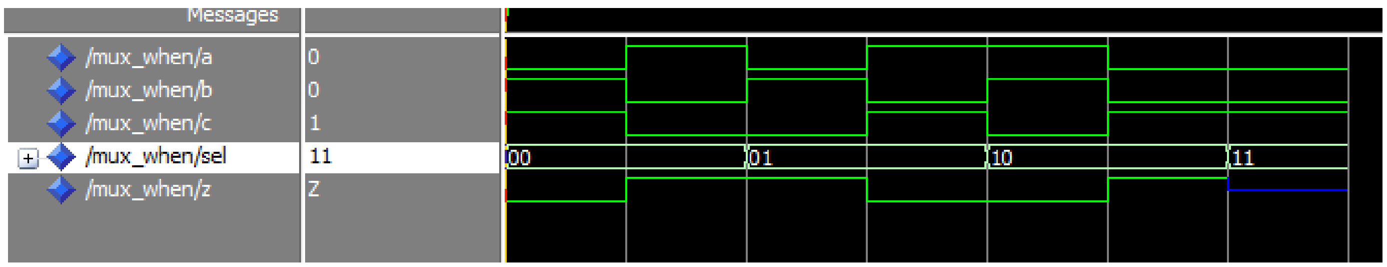


Figure 3.1 Simulation Results:

3.2. Select

```
With sel select
Signal_name    < =       expression_1 when choice_1
                         Expression_2 when choice_2
                         .
                         .
                         Expression_i when others
```

Example : (3-1 Mux)

Table 3.1 Truth table for 3-1 MUX

A	B	C	Sel	Z
A	X	X	00	A
X	B	X	01	B
X	X	C	10	C
X	X	X	11	Z

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

Entity MUX_Select is
port (
 A , B, C : in std_logic;
 Sel : in std_logic_vector (1 downto 0);
 Z : out std_logic );
end MUX_Select;

Architecture behavioral of MUX_Select is
begin
With Sel select
Z <= A   when "00",
    B   when "01" ,
    C   when "10" ,
    'Z' when others;
end behavioral;
```

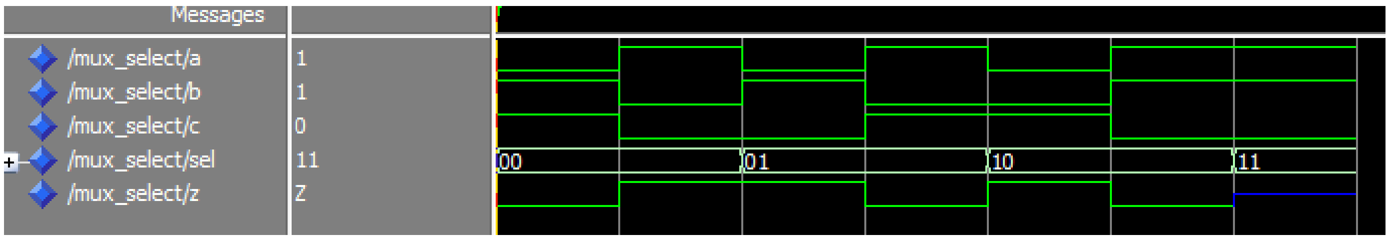


Figure 3.2 Simulation Results:

3.3. Using PROCESS

```
Process (sensitivity list)
Begin
        Sequential statements;
End process;
```

3.3.1 Signal Assignment

Signals are assigned have different order in process and outside of process:

```
process (a,b)
begin
z <= a or b;
z <= a and b;
end process;
```

only the last statement is affected. The code that is given below is same as above

```
process (a,b)
begin
z <= a and b;
end process;
```

3.3.2. If else Statement

```
If Boolean_exp_1 then
  Sequential statements
elsIf Boolean_exp_1 then
  Sequential statements
.
.
else
  Sequential statements
End if;
```

Example : (3-1 Mux)

Table 3.1 Truth table for 3-1 MUX

A	B	C	Sel	Z
A	X	X	00	A
X	B	X	01	B
X	X	C	10	C
X	X	X	11	Z

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

Entity MUX_IF is
port (
 A , B, C : in std_logic;
 Sel : in std_logic_vector (1 downto 0);
 Z : out std_logic );
end MUX_IF;

Architecture behavioral of MUX_IF is
begin
process (A,B,C,Sel)
  begin
if Sel = "00" then
  Z <= A;
elsif Sel = "01" then
  Z <= B;
elsif Sel = "10" then
  Z <= C;
else
  Z <= 'Z';
end if;

end process;
end behavioral;
```

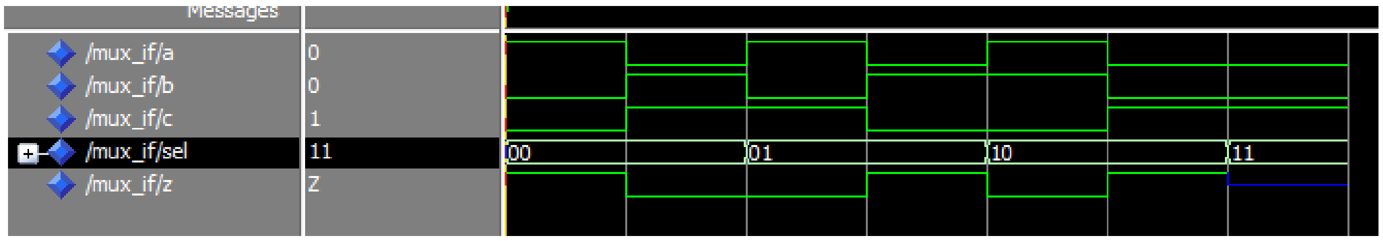


Figure 3.3 Simulation Results:

3.3.2. Case Statement

```
Case sel is

        When choice_1 =>
                Sequential statements;
        When choice_2 =>
                Sequential statements;
.
.
        When others =>
                Sequential statements;
End case
```

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

Entity MUX_CASE is
port (
 A , B, C : in std_logic;
 Sel : in std_logic_vector (1 downto 0);
 Z : out std_logic );
end MUX_CASE;

Architecture behavioral of MUX_CASE is
begin
process (A,B,C,Sel)
  begin

case Sel is
when "00" =>
  Z <= A;
when "01" =>
  Z <= B;
  when "10" =>
  Z <= C;
  when others =>
  Z <= 'Z';
end case;
end process;
end behavioral;
```

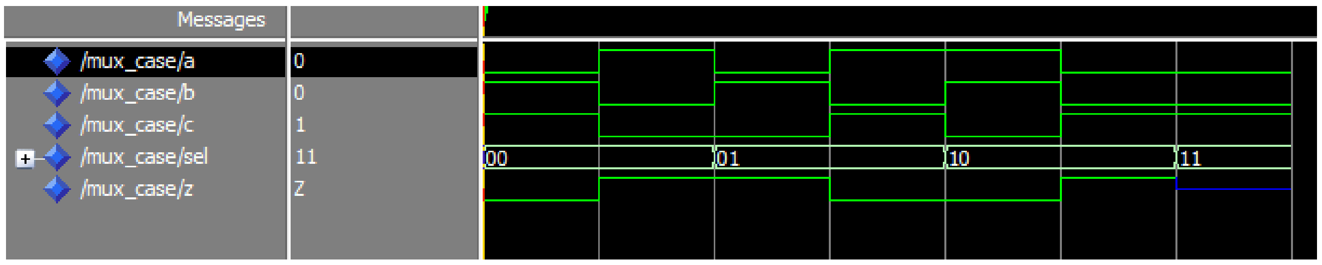


Figure 3.4 Simulation Results:

Tutorial assignments :

1-) Using 2 HA and component instantiation please design 1 full adder.

2-) Using if-else, case design 32-bit 5-1 Multiplexers.

References :

[1] FPGA PROTOTYPING BY VHDL EXAMPLES , Pong P. Chu, Wiley Publishing, 2008
[2] DIGITAL DESIGN , John F. Wakerly, Prenticehall, 2006