

**JK Lakshmipat University**

*Institute of Engineering & Technology*

**LAB1 (ASSIGNMENT SET -1)**

*CS1134: Computer Organisation and Architecture*

**Submitted by**  
Name: Siddhi Nyati  
Roll No: 2022btech101

**Faculty Supervisor:**

Dr. Pranab Roy

Dr. Ajai Jain

Activity no: 1

1.3: 4-bit Full Adder using Full Adder

**Question 3:** Design a VHDL model for a four bit full adder using full adder components

**Components Used:**

* A Half Adder (created using Dataflow Modeling)
* A Full Adder (created using Structural Modeling)

**Theory:**

In this project, we build upon the concepts of Half Adders and Full Adders to create a 4-bit Full Adder. Both components are constructed using fundamental logic gates, namely AND, OR, and XOR gates.

**Logic Gates Overview:**

**AND Gate:**

****

* **Truth Table:**

|  |  |  |
| --- | --- | --- |
| ***A (Input)*** | ***B (Input)*** | ***Q (Output) = A\cdot B*** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* **Boolean Expression:** Q = A . B

***OR Gate:***

**A yellow and black arrow with black text

Description automatically generated**

* **Truth Table:**

|  |  |  |
| --- | --- | --- |
| ***A (Input)*** | ***B (Input)*** | ***Q (Output) = A + B*** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

* **Boolean Expression:** Q = A + B

**Half Adder**

A Half Adder is a basic combinational circuit that adds two single-bit binary digits. It consists of an XOR gate and an AND gate, providing outputs for the sum and carry.

**Boolean Expressions:**

* **Sum:** Sum=A⊕B
* **Carry:** Carry=A⋅B

**Truth Table for Half Adder:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***A (Input)*** | ***B (Input)*** | ***SUM (Output)*** | ***CARRY (Output)*** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**Full Adder**

The Full Adder combines two binary inputs and an additional carry input to produce a sum and a new carry output. It consists of two Half Adders and one OR gate.

**Boolean Expressions:**

* **Sum:** Sum = (A ⊕ B ) ⊕ C
* **Carry:** Carry = (A . B) + ((A⊕B) . C)

**Truth Table for Full Adder:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***A (Input)*** | ***B (Input)*** | ***C (Input)*** | ***Sum (Output)*** | ***Carry (Output)*** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**VHDL Modeling**

VHDL (Very High-Speed Integrated Circuit Hardware Description Language) is essential for hardware design, allowing for various modeling approaches, including Dataflow, Behavioral, and Structural modeling.

In this project, we utilize:

* Dataflow Modeling for the Half Adder, which efficiently represents the flow of data using logical operators.
* Structural Modeling for the Full Adder, enabling the integration of smaller modules to form larger systems.

**Building a 4-Bit Full Adder**

To create a 4-bit Full Adder, we will use four single-bit Full Adders, which in turn are constructed from Half Adders.

**Procedure:**

1. Half Adder Construction: We first implement a Half Adder using XOR and AND gates to produce the sum and carry outputs.

A diagram of a circuit

Description automatically generated

1. Full Adder Construction: Next, we use two Half Adders and an OR gate to build a single-bit Full Adder. The first Half Adder receives inputs A and B, while the second receives the output sum from the first Half Adder and the carry input.



1. 4-Bit Full Adder Implementation: For the 4-bit Full Adder:

* Define input ports (A[3:0], B[3:0], Cin) and output ports (S[3:0], Cout).
* Map the previously defined single-bit Full Adders (FA1, FA2, FA3, FA4) in a cascading manner. Each Full Adder uses its respective bits from A and B, as well as the carry output from the previous Full Adder as input.
* The first Full Adder (FA1) processes A[0], B[0], and Cin, producing S[0] and C1. FA2 processes A[1], B[1], and C1, and so forth, until FA4.

1. Test Bench: Create a test bench to simulate the 4-bit Full Adder, demonstrating changes in the input waveforms over time.

A diagram of a full adder

Description automatically generated

**VHDL Code For The Single-Bit Full Adder Using Dataflow Modeling**

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 20:17:53 08/15/2024

-- Design Name:

-- Module Name: half\_adder - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity four\_bit\_full\_adder is

Port ( A : in STD\_LOGIC\_VECTOR(3 downto 0);

B : in STD\_LOGIC\_VECTOR(3 downto 0);

CIN : in STD\_LOGIC;

SUM : out STD\_LOGIC\_VECTOR(3 downto 0);

COUT : out STD\_LOGIC);

end four\_bit\_full\_adder;

architecture structural of four\_bit\_full\_adder is

signal C: STD\_LOGIC\_VECTOR(3 downto 0);

component full\_adder

Port ( A : in STD\_LOGIC;

B : in STD\_LOGIC;

CIN : in STD\_LOGIC;

SUM : out STD\_LOGIC;

COUT : out STD\_LOGIC);

end component;

begin

FA0: full\_adder port map (A(0), B(0), CIN, SUM(0), C(0));

FA1: full\_adder port map (A(1), B(1), C(0), SUM(1), C(1));

FA2: full\_adder port map (A(2), B(2), C(1), SUM(2), C(2));

FA3: full\_adder port map (A(3), B(3), C(2), SUM(3), COUT);

end structural;

**TEST BENCH CODE FOR THE 4-BIT FULL ADDER**

--------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 09:39:19 09/04/2024

-- Design Name:

-- Module Name: /home/ise/Desktop/f1byh/full4bittb.vhd

-- Project Name: f1byh

-- Target Device:

-- Tool versions:

-- Description:

--

-- VHDL Test Bench Created by ISE for module: full4bit

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

-- Notes:

-- This testbench has been automatically generated using types std\_logic and

-- std\_logic\_vector for the ports of the unit under test. Xilinx recommends

-- that these types always be used for the top-level I/O of a design in order

-- to guarantee that the testbench will bind correctly to the post-implementation

-- simulation model.

--------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity tb\_four\_bit\_full\_adder is

end tb\_four\_bit\_full\_adder;

architecture test of tb\_four\_bit\_full\_adder is

signal A, B: STD\_LOGIC\_VECTOR(3 downto 0);

signal CIN: STD\_LOGIC;

signal SUM: STD\_LOGIC\_VECTOR(3 downto 0);

signal COUT: STD\_LOGIC;

component four\_bit\_full\_adder

Port ( A : in STD\_LOGIC\_VECTOR(3 downto 0);

B : in STD\_LOGIC\_VECTOR(3 downto 0);

CIN : in STD\_LOGIC;

SUM : out STD\_LOGIC\_VECTOR(3 downto 0);

COUT : out STD\_LOGIC);

end component;

begin

UUT: four\_bit\_full\_adder port map (A, B, CIN);

process

begin

-- wait;

A <= NOT A AFTER f 100 NS;

B<=NOT B AFTER 200 NS:

CIN<=NOT CIN AFTER 300 NS:

end process;

end test;

