**DIGITAL CURRENCY CONVERTER**

**INTRODUCTION:**

In the modern era, digital currency has gained significant attention. Digital currency converters play an important role in facilitating the exchange of one type of digital currency into another, making it easier for users to transfer, trade, and invest in various cryptocurrencies. This mini-project will focus on creating a simple and efficient Verilog-based design that can take an input value in one cryptocurrency and convert it into the equivalent value in another cryptocurrency based on predefined conversion rates.

**ABSTRACT:**

The Digital Currency Converter project aims to create a digital system that can convert between two digital currencies. Using Verilog, this project will model the hardware logic required to perform currency conversion. The converter will use basic arithmetic operations (multiplication, division) and control structures to compute the exchange rate, with the conversion rate being predefined or configurable by the user. The purpose of this project is to showcase how hardware design (using Verilog) can be applied to practical financial applications like cryptocurrency conversion, highlighting the utility of digital logic design in modern financial technology.

**CODE:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity currency\_converter is

Port (

clk : in STD\_LOGIC;

input\_currency : in STD\_LOGIC\_VECTOR(1 downto 0); -- *Input currency* *selection*

output\_currency : in STD\_LOGIC\_VECTOR(1 downto 0); -- *Output currency* *selection*

amount\_in : in INTEGER; -- *Amount to convert*

converted\_amount : out INTEGER -- *Converted amount*

);

end currency\_converter;

architecture Behavioral of currency\_converter is

signal conversion\_rate : INTEGER; -- *Keep conversion\_rate as INTEGER*

begin

process(clk)

begin

if rising\_edge(clk) then

-- *Default conversion rate*

conversion\_rate <= 1000; -- *Default to 100% if input or output is invalid*

*-- Determine conversion rate based on input and output currency*

case input\_currency is

when "00" => -- *USD*

case output\_currency is

when "00" => conversion\_rate <= 1000; -- *USD to USD*

when "01" => conversion\_rate <= 85000; *-- USD to INR*

when "10" => conversion\_rate <= 930; *-- USD to EUR*

when others => conversion\_rate <= 1000; -- *Default case*

end case;

when "01" => -- *INR*

case output\_currency is

when "00" => conversion\_rate <= 12; -- *INR to USD*

when "01" => conversion\_rate <= 1000; -- *INR to INR*

when "10" => conversion\_rate <= 11; -- *INR to EUR*

when others => conversion\_rate <= 1000; -- *Default case*

end case;

when "10" => *-- EUR*

case output\_currency is

when "00" => conversion\_rate <= 1080; -- *EUR to USD*

when "01" => conversion\_rate <= 90790; -- *EUR to INR*

when "10" => conversion\_rate <= 1000; -- *EUR to EUR*

when others => conversion\_rate <= 1000; -- *Default case*

end case;

when others =>

conversion\_rate <= 1000; -- *Default for invalid input\_currency*

end case;

converted\_amount <= (amount\_in \* conversion\_rate)/1000; -- *Adjust by the scaling factor*

end if;

end process;

end Behavioral;

TEST BENCH CODE:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity tb\_CurrencyConverter is

end tb\_CurrencyConverter;

architecture Behavioral of tb\_CurrencyConverter is

signal clk : STD\_LOGIC := '0';

signal input\_currency : STD\_LOGIC\_VECTOR(1 downto 0);

signal output\_currency : STD\_LOGIC\_VECTOR(1 downto 0);

signal amount\_in : INTEGER;

signal converted\_amount : INTEGER;

constant CLK\_PERIOD : time := 10 ns;

-- *Instantiate the CurrencyConverter*

component currency\_converter

Port (

clk : in STD\_LOGIC;

input\_currency : in STD\_LOGIC\_VECTOR(1 downto 0);

output\_currency : in STD\_LOGIC\_VECTOR(1 downto 0);

amount\_in : in INTEGER;

converted\_amount : out INTEGER

);

end component;

begin

uut: currency\_converter

port map (

clk => clk,

input\_currency => input\_currency,

output\_currency => output\_currency,

amount\_in => amount\_in,

converted\_amount => converted\_amount

);

-- *Clock Generation*

clk\_process : process

begin

clk <= '0';

wait for CLK\_PERIOD / 2;

clk <= '1';

wait for CLK\_PERIOD / 2;

end process;

-- *Stimulus Process*

stimulus: process

begin

*-- Test Case*

input\_currency <= "10"; -- *Input currency: EUR*

output\_currency <= "01"; -- *Output currency: INR*

amount\_in <= 11; -- *Amount to convert*

wait for 20 ns;

wait;

end process;

end Behavioral;

**CODE EXPLAINATION**:

1. Library Imports:

* The code includes three important libraries:
  + IEEE.STD\_LOGIC\_1164: Provides definitions for standard logic types (e.g., STD\_LOGIC, STD\_LOGIC\_VECTOR).
  + IEEE.STD\_LOGIC\_ARITH: Defines arithmetic operations for data types like INTEGER and STD\_LOGIC\_VECTOR.
  + IEEE.STD\_LOGIC\_UNSIGNED: Allows operations on unsigned numbers (typically used for handling STD\_LOGIC\_VECTOR types as unsigned integers).

2.Entity Declaration:

* currency\_converter is the name of the entity, which represents the currency converter module.
* Ports (Inputs and Outputs):
  + clk: The clock signal that triggers updates in the converter (synchronous design).
  + input\_currency: A 2-bit input vector representing the selected input currency.
  + output\_currency: A 2-bit input vector representing the selected output currency.
  + amount\_in: The integer value representing the amount of currency to be converted.
  + converted\_amount: The output integer that represents the result of the conversion.

3. Architecture (Behavioral):

* The architecture Behavioral describes the functional behavior of the currency converter.
* Signal Declaration:
  + conversion\_rate: An integer signal used to store the exchange rate based on the input and output currency pair.

4. Process Block:

* The main logic of the converter is described inside a process block that is sensitive to the clk signal, i.e., it updates on the rising edge of the clock signal (rising\_edge(clk)).

5. Determining Conversion Rate:

* The process determines the conversion\_rate based on the combination of input\_currency and output\_currency. The currency pairs are represented as 2-bit vectors:
  + "00": USD
  + "01": INR (Indian Rupee)
  + "10": EUR (Euro)
* The conversion rates are hardcoded for each combination of input and output currencies, for example:
  + USD to USD: conversion\_rate <= 1000
  + USD to INR: conversion\_rate <= 85000
  + INR to USD: conversion\_rate <= 12
  + EUR to USD: conversion\_rate <= 1080

6. Default Conversion Rate:

* If the currency pair doesn't match any valid combination, the conversion\_rate is set to 1000 by default, indicating no conversion (or a 1:1 ratio).

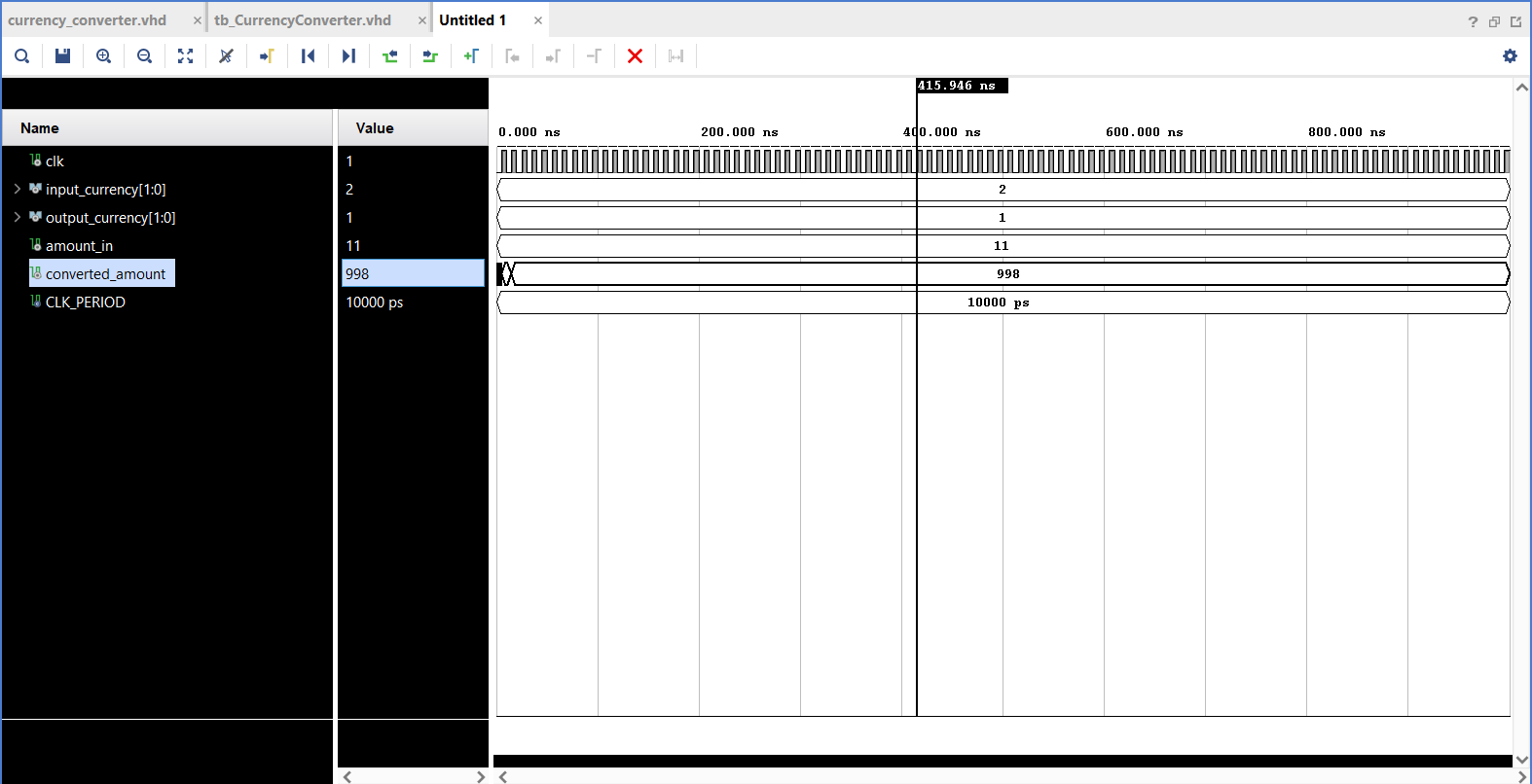
7. Conversion Logic:

* After the conversion rate is determined, the converted\_amount is calculated using the formula:
  + converted\_amount <= (amount\_in \* conversion\_rate) / 1000
  + The factor of 1000 is used to scale the result appropriately, ensuring the output is accurate based on the rates provided (i.e., to handle precision or fractional conversions).

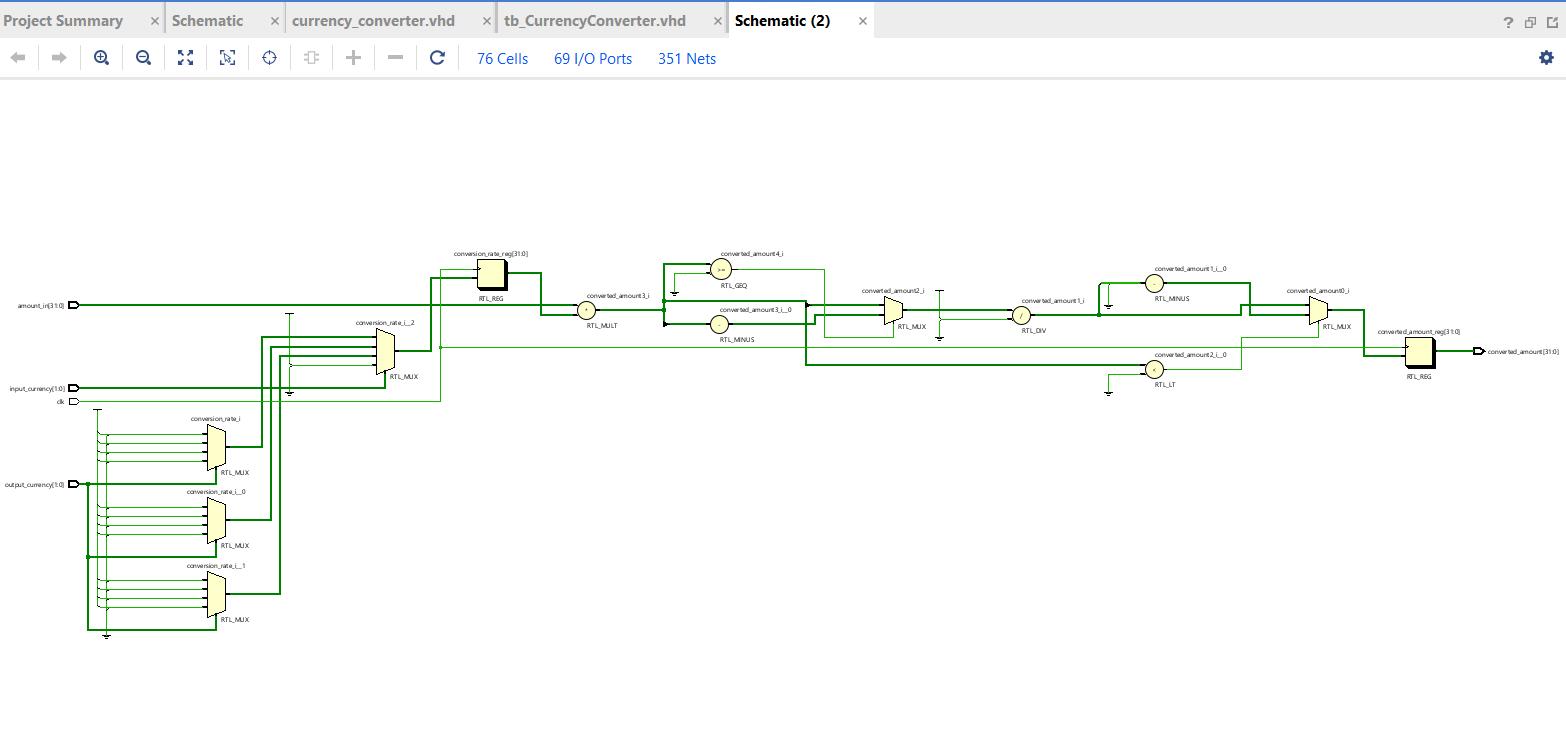
8. End of Process:

* The process block ends with the calculated converted\_amount being assigned to the output port, which will reflect the result of the currency conversion based on the given inputs.

**STIMULATION:**



**SCHEMATIC:**



**BLOCK DIAGRAM:**

CLOCK(CLK)

CURRENCY CONVERTER LOGIC(RATE LOOKUP)

INPUT CURRENCY(2-BIT)

AMOUNT\_IN

(INTEGER)

OUTPUT CURRENCY(2-BIT)

CONVERSION RATE

(SCALED BY 1000)

CONVERTED AMOUNT(INTEGER INPUT)

**CONCLUSION:**

The currency converter entity provides a simple hardware-based currency conversion system that takes an input amount and performs a conversion based on the selected currencies, using a predefined exchange rate for each pair.

The use of Verilog ensures that the design is synthesizable and can be implemented in hardware, making it suitable for digital systems where fast, reliable, and efficient currency conversion is required.

The project can be expanded in the future to include more sophisticated features such as handling dynamic exchange rates, supporting more cryptocurrencies, and integrating real-time data from cryptocurrency.