```
------------------------------------------------------------------------------
-- e^x ENTITY --
------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

-- Fixed point 32 bit signals
-- 0000000000000000000000.0000000000
-- |________22bit_______|.|_10bit_|
--          integer        fractional

ENTITY exponent IS
    PORT (
        A : IN std_logic_vector(3 DOWNTO 0);
        S : OUT std_logic_vector(31 DOWNTO 0)
    );
END exponent;

ARCHITECTURE arch OF exponent IS

    COMPONENT multiplier IS
        PORT (
            A : IN std_logic_vector(31 DOWNTO 0);
            B : IN std_logic_vector(31 DOWNTO 0);
            S : OUT std_logic_vector(31 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT adder IS
        PORT (
            A : IN std_logic_vector(31 DOWNTO 0);
            B : IN std_logic_vector(31 DOWNTO 0);
            S : OUT std_logic_vector(31 DOWNTO 0)
        );
    END COMPONENT;

    --Inputs
    SIGNAL input_A : std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');

    --Constants
    CONSTANT one : std_logic_vector(31 DOWNTO 0) := "0000000000000000000001000
0000000"; --1
    CONSTANT fac_2 : std_logic_vector(31 DOWNTO 0) := "00000000000000000000001
000000000"; --0.5000
    CONSTANT fac_3 : std_logic_vector(31 DOWNTO 0) := "00000000000000000000000
010101010"; --0.1667
```

```
    CONSTANT fac_4 : std_logic_vector(31 DOWNTO 0) := "00000000000000000000000
000101010"; --0.0417
    CONSTANT fac_5 : std_logic_vector(31 DOWNTO 0) := "00000000000000000000000
000001000"; --0.0083
    CONSTANT fac_6 : std_logic_vector(31 DOWNTO 0) := "00000000000000000000000
000000001"; --0.0014

    --Addition signals
    SIGNAL add_result_1 : std_logic_vector(31 DOWNTO 0);
    SIGNAL add_result_2 : std_logic_vector(31 DOWNTO 0);
    SIGNAL add_result_3 : std_logic_vector(31 DOWNTO 0);
    SIGNAL add_result_4 : std_logic_vector(31 DOWNTO 0);
    SIGNAL add_result_5 : std_logic_vector(31 DOWNTO 0);
    SIGNAL add_result_6 : std_logic_vector(31 DOWNTO 0);

    --Power signals
    SIGNAL pow_2 : std_logic_vector(31 DOWNTO 0);
    SIGNAL pow_3 : std_logic_vector(31 DOWNTO 0);
    SIGNAL pow_4 : std_logic_vector(31 DOWNTO 0);
    SIGNAL pow_5 : std_logic_vector(31 DOWNTO 0);
    SIGNAL pow_6 : std_logic_vector(31 DOWNTO 0);

    --Term signals
    SIGNAL term_2 : std_logic_vector(31 DOWNTO 0);
    SIGNAL term_3 : std_logic_vector(31 DOWNTO 0);
    SIGNAL term_4 : std_logic_vector(31 DOWNTO 0);
    SIGNAL term_5 : std_logic_vector(31 DOWNTO 0);
    SIGNAL term_6 : std_logic_vector(31 DOWNTO 0);

    --Result signals
    SIGNAL fp_result : std_logic_vector(31 DOWNTO 0);

    --Function to convert an unsigned binary 32-bit fixed pt number to 32-
bit floating pt sign-magnitude format
    FUNCTION UNSIGNED_TO_FP(input_number : unsigned) RETURN std_logic_vector I
S

        VARIABLE i : INTEGER RANGE 0 TO input_number'left + 1;
        VARIABLE j : INTEGER RANGE 0 TO 255;
        VARIABLE floating_point : std_logic_vector(31 DOWNTO 0); --returned FP
        VARIABLE exp : INTEGER RANGE -1024 TO 1024; --exponent
        VARIABLE mantissa : unsigned(input_number'length DOWNTO 0); --
mantissa + leading bit

    BEGIN
        mantissa := '0' & unsigned(input_number); --
we use the mag of v to create a mantissa
```

```vhdl
        --start with biased exp equiv to 2**(LENGTH-
1), so m becomes the mantissa, m.mmmmm...
        --i.e. mag(v) = 2**(exp-127) * m.m m m m m....
        exp := 127 + mantissa'length - 1;
        exp := exp - 10; -- -
10 is to account for the fixed point decimal parts

        --
normalize m as the mantissa with one bit in front of the decimal point
        FOR i IN 0 TO mantissa'left LOOP
            IF mantissa(mantissa'left) = '1' THEN
                j := i;
                EXIT;
            ELSE
                mantissa := mantissa(mantissa'left - 1 DOWNTO 0) & '0';
            END IF;
        END LOOP;

        exp := exp - j;

        --assigning the exponent bits
        floating_point(30 DOWNTO 23) := std_logic_vector(TO_UNSIGNED(exp, 8));

        --Make sure we have enough bits for a normalized full mantissa (23)
        --Remove the mantissa leading 1
        IF mantissa'length < 24 THEN -
- <24 bits, bottom bits set to 0, and drop the leading 1
            floating_point(23 - mantissa'length DOWNTO 0) := (OTHERS => '0');
            floating_point(22 DOWNTO 24 - mantissa'length) := std_logic_vector
(mantissa(mantissa'length - 2 DOWNTO 0));
        ELSE --
if >= 24, drop leading 1 and take next 23 bits for fp (kind of like rounding o
ff)
            floating_point(22 DOWNTO 0) := std_logic_vector(mantissa(mantissa'
length - 2 DOWNTO mantissa'length - 24));
        END IF;

        --assigning the sign bit
        floating_point(31) := '0';
        RETURN floating_point;
    END FUNCTION UNSIGNED_TO_FP;
BEGIN
    --Converting input to 32-
bit fixed point: 22 bits for the integer part, 10 parts for the fractional par
t
    input_A(10) <= A(0);
    input_A(11) <= A(1);
    input_A(12) <= A(2);
    input_A(13) <= A(3);

    --x^n
    SQUARE : multiplier PORT MAP(input_A, input_A, pow_2);
    CUBE : multiplier PORT MAP(pow_2, input_A, pow_3);
    FOURTH : multiplier PORT MAP(pow_3, input_A, pow_4);
    FIFTH : multiplier PORT MAP(pow_4, input_A, pow_5);
    SIXTH : multiplier PORT MAP(pow_5, input_A, pow_6);

    --individual terms
    T_2 : multiplier PORT MAP(pow_2, fac_2, term_2);
    T_3 : multiplier PORT MAP(pow_3, fac_3, term_3);
    T_4 : multiplier PORT MAP(pow_4, fac_4, term_4);
    T_5 : multiplier PORT MAP(pow_5, fac_5, term_5);
    T_6 : multiplier PORT MAP(pow_6, fac_6, term_6);

    --adding the terms together
    ADD_1 : adder PORT MAP(one, input_A, add_result_1); -- 1 + x
    ADD_2 : adder PORT MAP(add_result_1, term_2, add_result_2); -
- 1 + x + (x^2/2!)
    ADD_3 : adder PORT MAP(add_result_2, term_3, add_result_3); -
- 1 + x + (x^2/2!) + (x^3/3!)
    ADD_4 : adder PORT MAP(add_result_3, term_4, add_result_4); -
- 1 + x + (x^2/2!) + (x^3/3!) + (x^4/4!)
    ADD_5 : adder PORT MAP(add_result_4, term_5, add_result_5); -
- 1 + x + (x^2/2!) + (x^3/3!) + (x^4/4!) + (x^5/5!)
    ADD_6 : adder PORT MAP(add_result_5, term_6, add_result_6); -
- 1 + x + (x^2/2!) + (x^3/3!) + (x^4/4!) + (x^5/5!) + (x^6/6!)

    --result
    fp_result <= UNSIGNED_TO_FP(unsigned(add_result_6));
    S <= fp_result;

END arch;

------------------------------------------------------------------------------
-- MULTIPLIER ENTITY --
------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

-- Fixed point 32 bit signals
-- 0000000000000000000000.0000000000
-- |________22bit_______|.|_10bit_|
--         integer        fractional
```

```vhdl
ENTITY multiplier IS
    PORT (
        A : IN std_logic_vector(31 DOWNTO 0); -- N bit multiplier
        B : IN std_logic_vector(31 DOWNTO 0); -- N bit multiplicant
        S : OUT std_logic_vector(31 DOWNTO 0) -- real result
    );
END multiplier;

ARCHITECTURE Behavioral OF multiplier IS
    SIGNAL temp : std_logic_vector(63 DOWNTO 0);

BEGIN

    temp <= std_logic_vector(UNSIGNED(A) * UNSIGNED(B));
    S(9 DOWNTO 0) <= temp(19 DOWNTO 10);
    S(31 DOWNTO 10) <= temp(41 DOWNTO 20);

END Behavioral;

------------------------------------------------------------------------------
-- ADDER ENTITY --
------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

-- Fixed point 32 bit signals
-- 0000000000000000000000.0000000000
-- |________22bit_______|.|_10bit_|
--         integer        fractional

ENTITY adder IS
    PORT (
        A : IN std_logic_vector(31 DOWNTO 0); -- N bit Addend
        B : IN std_logic_vector(31 DOWNTO 0); -- N bit Augend
        S : OUT std_logic_vector(31 DOWNTO 0) -- real result
    );
END adder;

ARCHITECTURE Behavioral OF adder IS

BEGIN

    S <= std_logic_vector(UNSIGNED(A) + UNSIGNED(B));

END Behavioral;
```