```vhdl
------------------------------------------------------------------------------
-- SIN(X) ENTITY --
------------------------------------------------------------------------------

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

-- Fixed point 32 bit signals
-- 0000000000000000000.0000000000000
-- |______19bit______|.|___13bit___|
--         integer          fractional

-- Input 16 bit signal is a fixed point number of the form:
-- 000.0000000000000
-- 3 bits before decimal: Integral part
-- 13 bits after decimal: Fractional part

ENTITY sine IS
    PORT (
        A : IN std_logic_vector(15 DOWNTO 0);
        S : OUT std_logic_vector(31 DOWNTO 0)
    );
END sine;

ARCHITECTURE arch OF sine 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);
            carry : OUT std_logic
        );
    END COMPONENT;

    --Inputs
    SIGNAL input_A : std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');
    --Constants
    CONSTANT two : std_logic_vector(31 DOWNTO 0) := "10000000000000000010000000000000";
    CONSTANT one : std_logic_vector(31 DOWNTO 0) := "00000000000000000000000000000001"; --for two's complement

    CONSTANT fac_3 : std_logic_vector(31 DOWNTO 0) := "00000000000000000000010101010101"; --0.16666666666
    CONSTANT fac_5 : std_logic_vector(31 DOWNTO 0) := "00000000000000000000000001000100"; --0.00833333333
    CONSTANT fac_7 : std_logic_vector(31 DOWNTO 0) := "00000000000000000000000000000001"; --0.00019841269

    --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_5 : std_logic_vector(31 DOWNTO 0);
    SIGNAL pow_7 : 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);

    --Calculation signals
    SIGNAL ones_complement_1 : std_logic_vector(31 DOWNTO 0);
    SIGNAL ones_complement_2 : std_logic_vector(31 DOWNTO 0);
    SIGNAL twos_complement : std_logic_vector(31 DOWNTO 0);

    --signals used to store the sign bit/carry bit from the adder
    SIGNAL dummy_1 : std_logic;
    SIGNAL dummy_2 : std_logic;
    SIGNAL dummy_3 : std_logic;
    SIGNAL dummy_4 : std_logic;
    SIGNAL sign_MSB : std_logic;

    --Result signals
    SIGNAL original : std_logic_vector(31 DOWNTO 0);
    SIGNAL complement : std_logic_vector(31 DOWNTO 0);
    SIGNAL result : std_logic_vector(32 DOWNTO 0);
    SIGNAL fp_result : std_logic_vector(31 DOWNTO 0);


    --Function to convert a signed binary 32-bit fixed pt number to 32-
bit floating pt sign-magnitude format
    FUNCTION SIGNED_TO_FP(input_number : unsigned; carry : std_logic) RETURN s
td_logic_vector IS

        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

        IF input_number = (31 DOWNTO 0 => '0') THEN
            floating_point := (OTHERS => '0');
        ELSE
            mantissa := '0' & unsigned(input_number); --
we use the mag of input_number to create a mantissa

            --start with biased exp equiv to 2**(LENGTH-
1), so m becomes the mantissa, m.mmmmm...
            --i.e. mag(input_number) = 2**(exp-127) * m.m m m m m....
            exp := 127 + mantissa'length - 1;
            exp := exp - 13; -- -
13 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_ve
ctor(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(manti
ssa'length - 2 DOWNTO mantissa'length - 24));
            END IF;

            --assigning the sign bit in accordance with two's coplement
            IF carry = '1' THEN
                floating_point(31) := '1';
            ELSE
                floating_point(31) := '0';
            END IF;
        END IF;
        RETURN floating_point;
    END FUNCTION SIGNED_TO_FP;

    --Function to finalise the output (+ve or -
ve) depending on the two's complement subtraction
    FUNCTION FINALISE_RESULT(original : std_logic_vector; complement : std_log
ic_vector; carry : std_logic) RETURN std_logic_vector IS
        VARIABLE result : std_logic_vector(31 DOWNTO 0); --returned FP
    BEGIN
        IF carry = '1' THEN
            --RETURN original;
            result := original;
        ELSE
            --RETURN complement;
            result := complement;
        END IF;
        RETURN std_logic_vector(result);
        --RETURN result;
    END FUNCTION FINALISE_RESULT;

BEGIN
    --Converting input to 32-
bit fixed point: 19 bits for the integer part, 13 parts for the fractional par
t
    input_A(15 DOWNTO 0) <= A(15 DOWNTO 0);

    --x^n
    SQUARE : multiplier PORT MAP(input_A, input_A, pow_2);
    CUBE : multiplier PORT MAP(pow_2, input_A, pow_3);
    FIFTH : multiplier PORT MAP(pow_3, pow_2, pow_5);
    SEVENTH : multiplier PORT MAP(pow_5, pow_2, pow_7);

    --individual terms
    T_2 : multiplier PORT MAP(pow_3, fac_3, term_2);
    T_3 : multiplier PORT MAP(pow_5, fac_5, term_3);
    T_4 : multiplier PORT MAP(pow_7, fac_7, term_4);

    --adding the terms together
    ADD_1 : adder PORT MAP(input_A, term_3, add_result_1, dummy_1); -
- x + (x^5/5!)
    ADD_2 : adder PORT MAP(term_2, term_4, add_result_2, dummy_2); -
- (x^3/3!) + (x^7/7!)

    --converting the negative terms to two's complement
    ones_complement_1 <= NOT(add_result_2);
    ADD_3 : adder PORT MAP(ones_complement_1, one, twos_complement, dummy_3);
-- - ((x^3/3!) + (x^7/7!))

    ADD_4 : adder PORT MAP(add_result_1, twos_complement, add_result_3, sign_M
SB); -- x + (x^5/5!) - ((x^3/3!) + (x^7/7!))

    --RESULT--

    --this is for the case where the two's complement subtraction has a carry
    original <= add_result_3;

    --
this is for the case where the two's complement subtraction doesn't have a car
ry
    ones_complement_2 <= NOT(add_result_3);
    ADD_5 : adder PORT MAP(ones_complement_2, one, complement, dummy_4);

    result(32) <= NOT(sign_MSB); --stores the sign of the result

    --picks result depending on whether the output is +ve or -ve
    result(31 DOWNTO 0) <= FINALISE_RESULT(original, complement, sign_MSB);

    --converting the result to floating point for making it easily readable
    fp_result <= SIGNED_TO_FP(unsigned(result(31 DOWNTO 0)), result(32));
    S <= fp_result;

END arch;

------------------------------------------------------------------------------
-- MULTIPLIER ENTITY --
```

```vhdl
------------------------------------------------------------------------------

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

-- Fixed point 32 bit representation
-- 0000000000000000000.0000000000000
-- |______19bit______|.|___13bit___|
--       integer         fractional

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(12 DOWNTO 0) <= temp(25 DOWNTO 13); --decimal part
    S(31 DOWNTO 13) <= temp(44 DOWNTO 26); --fractional part

END Behavioral;

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

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

-- Fixed point 32 bit representation
-- 0000000000000000000.0000000000000
-- |______19bit______|.|___13bit___|
--       integer         fractional

ENTITY adder IS
    PORT (
        A : IN std_logic_vector(31 DOWNTO 0); -- 32 bit Addend
        B : IN std_logic_vector(31 DOWNTO 0); -- 32 bit Augend
        S : OUT std_logic_vector(31 DOWNTO 0); -- real result
        carry : OUT std_logic -- 1 bit carry: required for two's complement
    );
END adder;

ARCHITECTURE Behavioral OF adder IS

    SIGNAL number_1 : std_logic_vector(32 DOWNTO 0);
    SIGNAL number_2 : std_logic_vector(32 DOWNTO 0);
    SIGNAL result : std_logic_vector(32 DOWNTO 0);

BEGIN

    number_1 <= '0' & A;
    number_2 <= '0' & B;

    result <= std_logic_vector(UNSIGNED(number_1) + UNSIGNED(number_2));

    S <= result(31 DOWNTO 0);
    carry <= result(32);

END Behavioral;
```