```
-------------------------------------------EDGE DET
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY edge_det IS
        PORT
        (
                sig       : std_logic; ---a
                reset_bar : IN std_logic; --clr_bar
                clk       : IN std_logic;
                pos       : IN std_logic;
                sig_edge  : OUT std_logic ---a_pe
        );
END edge_det;
ARCHITECTURE moore_fsm OF edge_det IS

        TYPE state IS (state_a1, state_a2, state_c, state_d);
        SIGNAL present_state, next_state : state;
BEGIN
        state_reg : PROCESS (clk, reset_bar)
        BEGIN
                IF reset_bar = '0' THEN
                        IF sig = '1' THEN
                                present_state <= state_a1;
                        ELSE
                                present_state <= state_a2;
                        END IF;
                ELSIF rising_edge(clk) THEN
                        present_state <= next_state;
                END IF;
        END PROCESS;

        outputs : PROCESS (present_state)
        BEGIN
                IF pos = '1' THEN
                        CASE present_state IS ---if pos 1 then only do
rising edges
                                WHEN state_c => sig_edge <= '1';
                                WHEN OTHERS => sig_edge  <= '0';
                        END CASE;

                ELSE
                        CASE present_state IS
                                WHEN state_d => sig_edge <= '1'; ---if
pos 0 then only do falling edges
                                WHEN OTHERS => sig_edge  <= '0';
                        END CASE;
                END IF;
END PROCESS;

nxt_state : PROCESS (present_state, sig)
BEGIN
        CASE present_state IS
                ---A1
                WHEN state_a1 =>
                        IF sig = '0' THEN
                                next_state <= state_d;
                        ELSE
                                next_state <= state_a1;
                        END IF;

                        ---A2
                WHEN state_a2 =>
                        IF sig = '0' THEN
                                next_state <= state_a2;
                        ELSE
                                next_state <= state_c;
                        END IF;

                        ---D
                WHEN state_d =>
                        IF sig = '1' THEN
                                next_state <= state_c;
                        ELSE
                                next_state <= state_a2;
                        END IF;

                        ---C
                WHEN state_c =>
                        IF sig = '1' THEN
                                next_state <= state_a1;
                        ELSE
                                next_state <= state_d;
                        END IF;


        END CASE;
END PROCESS;
END moore_fsm;
-------------------------------------------FREQUENCY REG
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY frequency_reg IS
        GENERIC (a : POSITIVE := 14);
        PORT
        (
        load      : IN std_logic;
```

```
                clk       : IN std_logic;
                reset_bar : IN std_logic;
                d         : IN std_logic_vector(a - 1 DOWNTO 0);
                q         : OUT std_logic_vector(a - 1 DOWNTO 0)
                );

        END frequency_reg;
        ARCHITECTURE dataflow OF frequency_reg IS
        BEGIN
                edge_load : PROCESS (clk, reset_bar)
                BEGIN
                        IF reset_bar = '0' THEN
                                q <= (OTHERS => '0');
                        ELSIF (rising_edge(clk) AND load = '1') THEN
                                q <= d;
                        END IF;
                END PROCESS;


                END dataflow;

                -------------------------------------------PHASE ACCUM
                LIBRARY ieee;
                USE ieee.std_logic_1164.ALL;
                USE ieee.numeric_std.ALL;

                ENTITY phase_accumulator IS
                        GENERIC
                        (
                        a : POSITIVE := 14;
                        m : POSITIVE := 7
                        );

                        PORT
                        (
                        clk       : IN std_logic;
                        reset_bar : IN std_logic;
                        up        : IN std_logic;
                        d         : IN std_logic_vector(a - 1 DOWNTO
0);
                        max       : OUT std_logic;
                        min       : OUT std_logic;
                        q         : OUT std_logic_vector(m - 1 DOWNTO
0)

                        );

                END phase_accumulator;

                ARCHITECTURE dataflow OF phase_accumulator IS
```

```
                        SIGNAL state : std_logic := '0';
                        SIGNAL one   : std_logic := '1';
                        SIGNAL two   : std_logic := '0';
                        SIGNAL x : std_logic_vector( a-1 downto 0);
                BEGIN
                        PROCESS (clk)
                        VARIABLE up_down  : std_logic := '0';
                        VARIABLE cnt      : INTEGER RANGE -16384 TO
16384 := 1;
                        VARIABLE cnt_compare :INTEGER Range -16384 to
16384 := 0 ;
                        VARIABLE leftover : INTEGER RANGE -16384 TO
16384 := 0;
                        VARIABLE max_v    : std_logic := '0';
                        VARIABLE min_v    : std_logic := '0';

                        BEGIN


                                        IF rising_edge(clk) THEN
                                        ------POS SECTION



                                        IF up_down = '0' THEN
                             cnt_compare := cnt;
                                                 cnt_compare :=
cnt_compare + to_integer(unsigned(d));

                                                IF ( cnt = 127) THEN
                                                up_down := '1';
                                                max <= '1';
                                                min <= '0';

                                                ELSIF (cnt_compare >
127) THEN ---counting up failsafe for overflow ---IF the integer value
of the current count and whatever is currently at d is greater than
127
                                                        up_down  :=
'1';
                                                        leftover :=
cnt_compare - 127;
                                                        cnt      :=
127 - leftover;
                                                        max <= '1';

                                                        min <= '0';


                                                ELSE
```

```
                                                            cnt := cnt +
to_integer( unsigned(d) ); ---take count and add the d input to it

                                                            max <= '0';

                                                            min <= '0';
                                                    END IF;

                                            ELSE

                                                    cnt_compare := cnt;
                                                    cnt_compare :=
cnt_compare - to_integer( unsigned(d) );


                                                    IF ( cnt = 0 or cnt =
1 ) THEN
                                                    up_down := '0';
                                                    max <= '0';
                                                    min <= '1';
                                            --      state <= NOT state;
--                                                  ---pos_neg <= state;
--                                                  one <= '0';
--                                                  two <= '1';



                                                    ELSIF ( cnt_compare <
0 ) THEN ---counting down failsafe for underflow ---IF the integer
value of the current count take away whatever is currently at d is
less than 0
                                                            up_down  :=
'0';
                                                            leftover := (0
- cnt_compare); --- Take the current count and minus d. Then absolute
value that and save it as an integer.
                                                            cnt      :=
leftover ;
                                                            max <= '0';

                                                    min <= '1';
                                                    ELSE
                                                            cnt :=
cnt_compare; ---current count minus what is located at d
                                                             max <= '0';

                                                            min <= '0';

                                                    END IF;
                                            END IF;
```

```
                                END IF;




                                q <=
std_logic_vector(to_unsigned(cnt,7));
                        END PROCESS;

                        ---direction :process (up_down)
                        ---begin
                        --- case up_down is ---stimulating up on
direction change
                        ---when '0' => up <= '1'; --- when up_down is
0 that means its counting up, reaching 1 makes it count down
                        ---when others => up <= '0';
                        --- end case;
                        ---end process;

                        -- maximum :process (max_v)
                        -- begin
                        -- case max_v is ---stimulating up on
direction change
                        -- when '1' => max <= '1'; --- when up_down is
0 that means its counting up, reaching 1 makes it count down
                        -- when others => max <= '0';
                        -- end case;
                        -- end process;
                        --
                        -- minimum :process (min_v)
                        -- begin
                        -- case min_v is ---stimulating up on
direction change
                        -- when '1' => min <= '1'; --- when up_down is
0 that means its counting up, reaching 1 makes it count down
                        -- when others => min <= '0';
                        -- end case;
                        -- end process;

                        END dataflow;
                        -------------------------------------------
PHASE ACCUM FSM
                        LIBRARY ieee;
                        USE ieee.std_logic_1164.ALL;
                        USE ieee.numeric_std.ALL;

                        ENTITY phase_accumulator_fsm IS
                                PORT
                                (
                                clk       : IN std_logic;
                                reset_bar : IN std_logic;
                                up        : OUT std_logic;
                                pos       : OUT std_logic := '0';
                                max       : IN std_logic;
                                min       : IN std_logic


                                );

                        END phase_accumulator_fsm;
                        ARCHITECTURE moore_fsm OF
phase_accumulator_fsm IS







                                TYPE state IS (q0, q1, q2, q3);
                                SIGNAL present_state, next_state :
state;
                                ---SIGNAL pos_val                   :
std_logic := '0';
                        BEGIN










                                state_reg : PROCESS (clk, reset_bar)
                                BEGIN
                                        ---variable up_down :
std_logic := '0';


                                        IF reset_bar = '0' THEN
                                                present_state <= q0;
                                        ELSIF rising_edge(clk) THEN
                                                present_state <=
next_state;
```

```
                                        END IF;
                                END PROCESS;




                                outputs : PROCESS (present_state)
                                BEGIN
                                        CASE present_state IS
                                                when q0 => up <= '1' ;
pos <= '1';
                                                when q1 => up <= '0' ;
pos <= '1';
                                                when q2 => up <= '1' ;
pos <= '0';
                                                when q3 => up <= '0' ;
pos <= '0';

                                        END CASE;

                                END PROCESS;




                                nxt_state : PROCESS (present_state,
max, min)
                                BEGIN
                                        CASE present_state IS
                                                ---q0
                                                WHEN q0 =>
                                                        IF max = '1'
THEN

next_state <= q1;
                                                        ELSE

next_state <= q0;
                                                        END IF;
                                                        ---q1
                                                WHEN q1 =>
                                                 if min = '1' then
                                                        next_state <=
q2;
                                                        ELSE

next_state <= q1;
                                                        END IF;

                                                        ---q2
                                                WHEN q2 =>

                                                if max = '1' then
                                                 next_state <= q3;
                                                ELSE

next_state <= q2;
                                                        END IF;

                                                        ---q3
                                                WHEN q3 =>
                                                        if min = '1'
then

                                                 next_state <= q0;
                                                ELSE

next_state <= q3;
                                                        END IF;
                                                END CASE;
                                END PROCESS;







                                END moore_fsm;


                                --------------------------------------
-----SIN
                                LIBRARY ieee;
                                USE ieee.std_logic_1164.ALL;
                                USE ieee.numeric_std.ALL;

                                ENTITY lookup_table IS

                                        GENERIC
                        (
                        a : POSITIVE := 14;
                        m : POSITIVE := 7
                        );

                                        PORT
                                        (
                                        addr    : IN
std_logic_vector(m-1 DOWNTO 0);
                                        sin_val : OUT
std_logic_vector(6 DOWNTO 0)
                                        );
                                END ENTITY lookup_table;

                                ARCHITECTURE rtl OF lookup_table IS
                                        TYPE mem_type IS ARRAY (0 TO
127) OF std_logic_vector(6 DOWNTO 0);
                                        CONSTANT lookup : mem_type :=
(
                                                "0000000", "0000001",
"0000011", "0000100", "0000110", "0000111", "0001001", "0001010",
                                                "0001100", "0001110",
"0001111", "0010001", "0010010", "0010100", "0010101", "0010111",
                                                "0011000", "0011010",
"0011100", "0011101", "0011111", "0100000", "0100010", "0100011",
                                                "0100101", "0100110",
"0101000", "0101001", "0101011", "0101100", "0101110", "0101111",
                                                "0110000", "0110010",
"0110011", "0110101", "0110110", "0111000", "0111001", "0111010",
                                                "0111100", "0111101",
"0111111", "1000000", "1000001", "1000011", "1000100", "1000101",
                                                "1000111", "1001000",
"1001001", "1001010", "1001100", "1001101", "1001110", "1001111",
                                                "1010001", "1010010",
"1010011", "1010100", "1010101", "1010111", "1011000", "1011001",
                                                "1011010", "1011011",
"1011100", "1011101", "1011110", "1011111", "1100000", "1100001",
                                                "1100010", "1100011",
"1100100", "1100101", "1100110", "1100111", "1101000", "1101001",
                                                "1101010", "1101011",
"1101100", "1101100", "1101101", "1101110", "1101111", "1110000",
                                                "1110000", "1110001",
"1110010", "1110011", "1110011", "1110100", "1110101", "1110101",
                                                "1110110", "1110110",
"1110111", "1110111", "1111000", "1111001", "1111001", "1111010",
                                                "1111010", "1111010",
"1111011", "1111011", "1111100", "1111100", "1111100", "1111101",
                                                "1111101", "1111101",
"1111110", "1111110", "1111110", "1111110", "1111111", "1111111",
                                                "1111111", "1111111",
"1111111", "1111111", "1111111", "1111111", "1111111", "1111111");
```

```
                                        BEGIN sin_val <= lookup
( to_integer(unsigned(addr)));
                                        END rtl;

                                        ------------------------------
-----------adder_subtractor
                                        LIBRARY ieee;
                                        USE ieee.std_logic_1164.ALL;
                                        USE ieee.numeric_std.ALL;
                                        ENTITY adder_subtractor IS
                                                PORT
                                        (
                                                inputVec  : IN
std_logic_vector(6 DOWNTO 0);
                                                pos       : IN
std_logic;
                                                outputVec : OUT
std_logic_vector(7 DOWNTO 0)
                                                );
                                        END ENTITY;

                                        ARCHITECTURE behavioral OF
adder_subtractor IS
                                        BEGIN
                                                PROCESS (inputVec,
pos)
                                                BEGIN
                                                        IF (pos = '0')
THEN

outputVec <= std_logic_vector("10000000" + Unsigned(inputVec));
                                                        ELSE

outputVec <= std_logic_vector("10000000" - Unsigned(inputVec));
                                                        END IF;
                                                END PROCESS;
                                                END ARCHITECTURE;



                                                ----------------------
---------------------DDS_w_freq_sel
                                                LIBRARY IEEE;
                                                USE
IEEE.std_logic_1164.ALL;
                                                USE
IEEE.numeric_std.ALL;
                                                USE work.ALL;

                                                ENTITY
```

```
dds_w_freq_select IS
                                                         GENERIC
                                                         (
                                                         a :
POSITIVE := 14;
                                                         m :
POSITIVE := 7
                                                         );
                                                         PORT
                                                         (

clk            : IN std_logic;-- system clock

reset_bar      : IN std_logic;-- asynchronous reset

freq_val       : IN std_logic_vector (a - 1 DOWNTO 0);

load_frequency : IN std_logic;

dac_sine_value : OUT std_logic_vector(7 DOWNTO 0);-- to DAC

pos_sine       : OUT std_logic;

pos            : IN std_logic := '1'
                                                         );
                                                         attribute
loc : string;
attribute loc of clk       : signal is "C13";
attribute loc of reset_bar : signal is "A13";
attribute loc of freq_val  : signal is
"F8,C12,E10,F9,E8,E7,D7,C5,E6,A10,D9,B6,B5,B4";
attribute loc of load_frequency : signal is "A3";
attribute loc of dac_sine_value  : signal is "
N3,M1,L1,L5,J1,J2,H3,H1";
attribute loc of pos_sine  : signal is "F5";



                                                 END dds_w_freq_select;
                                                 ARCHITECTURE
structural OF dds_w_freq_select IS

                                                         ---signal
pos_neg : std_logic;

                                                         SIGNAL
up           : std_logic;
                                                         SIGNAL
pos_s        : std_logic;
```

```
                                                        SIGNAL
max           : std_logic;
                                                        SIGNAL
min           : std_logic;
                                                        SIGNAL
sig_edge      : std_logic;
                                                        SIGNAL
freq_2_phase : std_logic_vector (a - 1 DOWNTO 0);
                                                        SIGNAL
address       : std_logic_vector(m - 1 DOWNTO 0);
                                                        SIGNAL
sin_val       : std_logic_vector(6 DOWNTO 0);
                                                        ----EDGE plug
into freq_reg (sig_pos -> load , D in [vector] , q out to phase
accum )
                                                BEGIN
                                                        ---
counter :entity counter port map (clk => clk, reset_bar => reset_bar,
pos_neg => pos_neg, address => address );

                                                        edge_det :
ENTITY edge_det
                                                                PORT
MAP(clk => clk, reset_bar => reset_bar, sig => load_frequency,
sig_edge => sig_edge, pos => pos);


frequency_reg : ENTITY frequency_reg

        PORT MAP(clk => clk, reset_bar => reset_bar, load => sig_edge,
d => freq_val, q => freq_2_phase);


        phase_accum : ENTITY phase_accumulator

                PORT MAP(clk => clk, reset_bar => reset_bar, up => up,
d => freq_2_phase, max => max, min => min, q => address);


                phase_accumulator_fsm : ENTITY phase_accumulator_fsm

                        PORT MAP(clk => clk, reset_bar => reset_bar,
up => pos_sine, pos => pos_s, max => max, min => min);


                        sin_table : ENTITY lookup_table

                                PORT MAP(addr => address, sin_val =>
sin_val);
```

```
                                adder_subtractor : ENTITY
adder_subtractor
                                        PORT MAP(pos => pos_s,
inputVec => sin_val, outputVec => dac_sine_value);



END structural;
```