FPGA Features, Clock Management, DSP Blocks, DDR, and SRL

Ridge Tejuco
California State University Northridge
Department of Electrical and Computer Engineering
Northridge, California
Ridge. Tejuco.881@my.csun.edu

I. Introduction

The purpose of this experiment is to become familiar with the various features and resources found on the ZedBoard. These features include the digital clock managers, DSP blocks, and SRL and block ram configurations. The circuit used for this experiment is a multiply accumulate circuit which takes two data streams as inputs and multiplies them by given constants. The two data streams are then mixed.

The multiply accumulate circuit is implemented in three different ways, each using different FPGA resources. The first is a direct implementation of the circuit using DSP blocks. Each data stream is a sine wave, and the coefficients are values ranging from 0 to 1. The second way is an interleaver circuit which is an optimized implementation that includes clocks at each stage of the multiply accumulate circuit and multiplexers for each data stream. This implementation reduces the amount of resources needed for the operation, increases the output rate, and increases the functionality of the design. The last implementation incorporates equalizing delay. Shift right logical elements are placed at the inputs to synchronize the data streams.

II. Procedure and Results

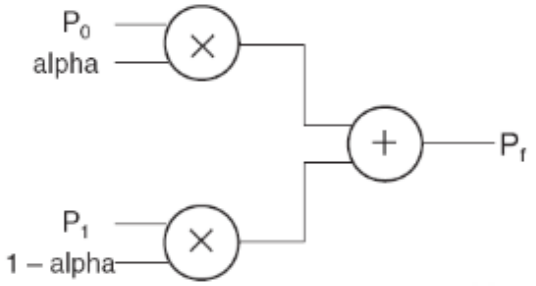

Fig. 3.1 Multiply Accumulate block diagram

Fig. 3.1 shows the high level block diagram of the design. Data streams P_0 and P_1 are sine waves, and alpha is a coefficient between 0 and 1. These inputs are represented by 8 bit values. The output P_f is 17 bits.

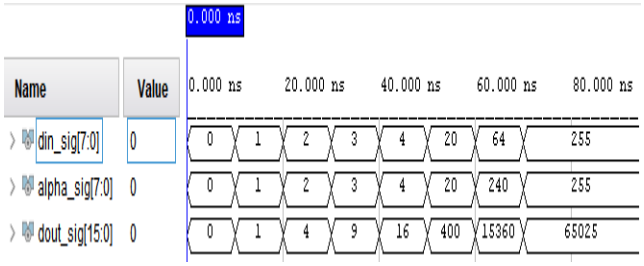

Fig. 3.2 Multiplier test bench results

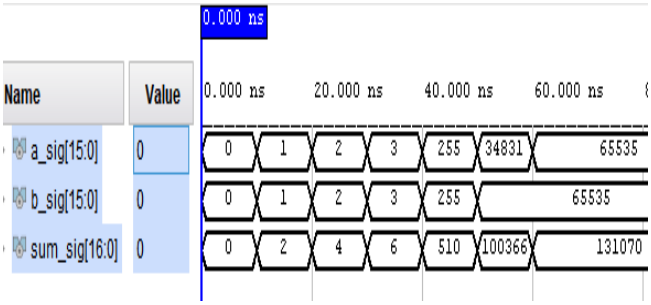

Fig. 3.3 Adder test bench results

Fig 3.2 and Fig 3.3 show the results of the test benches for the multipliers and adders. The waveforms show the inputs and outputs in decimal format. Going through each results shows the correct operation of each element.

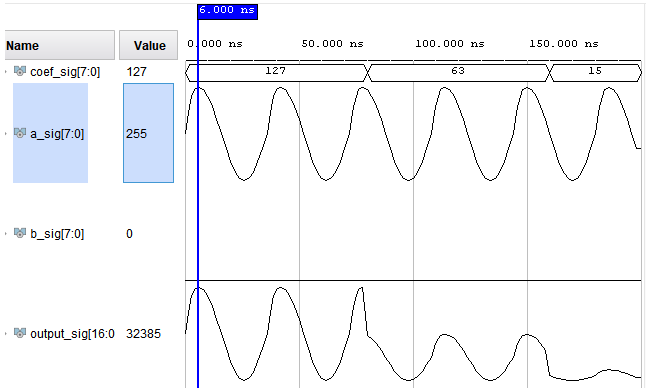

Fig 3.4 Multiply accumulate test bench waveform with a sine wave at signal a

Looking closely at Fig 3.4, the results of the MAC block can be seen. Signal a was a sine wave with a range of values

from 0 to 255. In this case, 0 represents a value of -1 in the sine wave and 255 represents a value of 1. For simplification, signal b was kept at a constant 0, and has no contribution to the output. For the first 80 ns, the coefficient is 127 so about half of signal a and b are added. From 80 ns to 160 ns, the coefficient is 63 so about 25% of signal a is seen at the output. Lastly from 160 ns to 200 ns, only about 5% of signal a is seen.

Fig 3.5 is given to show the reverse input where signal a is set to a value of 0, and signal b is a sine wave. For the first 80 ns, still about 50% of the sine wave is output. From 80 to 160 ns, about 75% of the sine wave is seen at the output. Then for the last 40 ns, about 95% of the sine wave can be seen at the output. These results prove the functionality of the MAC block.

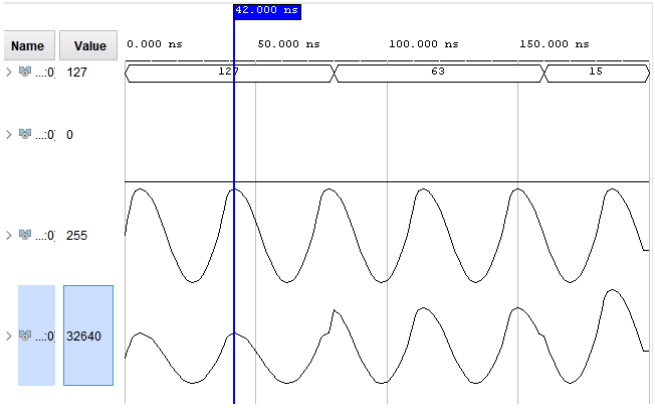

Fig 3.5 Reversed multiply Accumulate testbench waveform with a sine wave at signal b

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	157	0	53200	0.30
LUT as Logic	157	0	53200	0.30
LUT as Memory	0	0	17400	0.00

Fig 3.6 Resource utilization report showing slice logic

Fig 3.6 shows a snippet from the utilization report. 157 LUTs were used meaning that Slice logic was used rather than the DSP blocks. In order to force an implementation of DSP blocks, the "use_dsp" attribute can be set. For this implementation, the use_dsp attribute could not result in forcing the use of DSP blocks.

The next implementation incorporates clocked inputs and outputs at each stage. Fig 3.7 shows the block diagram for the implementation. The multiplexers alternate between inputs and the mode of operation for new data and for accumulating the previous data stream.

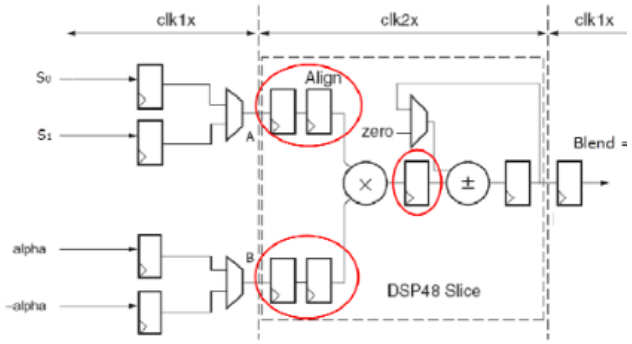

Fig. 3.7 High performance multiply accumulator design

Fig. 3.8 shows the results of the high performance interleaver circuit design. The stimulus in the testbench for the first implementation was reused. Looking closely at the output, the output waveform is identical to the output waveform in Fig 3.4. This shows that the functionality of the first two implementations are the same.

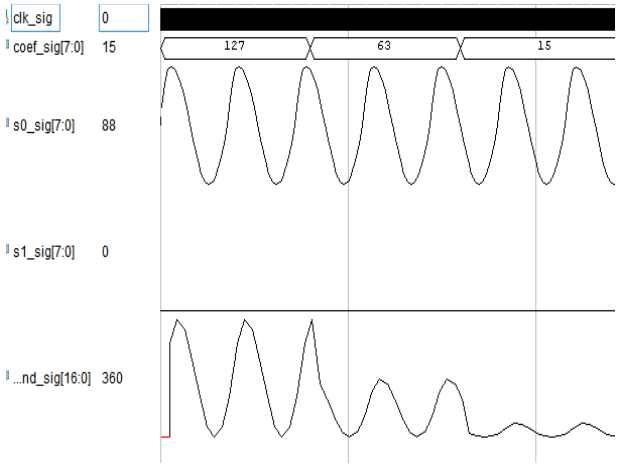

Fig. 3.8 Output waveforms of interleaver design

```
-- adder.vhd
-- ECE 524 Fall 2020
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity adder is
    Port (
        a_in, b_in: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(16 downto 0)
    );
end adder;
architecture Behavioral of adder is
begin
    sum <= std_logic_vector(unsigned("0"&a_in)+unsigned("0"&b_in));
end Behavioral;
```

```
-- adder_tb.vhd
-- ECE524 FALL2020
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity adder_tb is
--  Port ( );
end adder_tb;
architecture Behavioral of adder_tb is
component adder is
    Port (
        a_in, b_in: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(16 downto 0)
    );
end component;
signal a_sig, b_sig: std_logic_vector(15 downto 0);
signal sum_sig: std_logic_vector(16 downto 0);
constant cp: time := 10 ns;
begin
    DUT: adder
    port map(
        a_in => a_sig,
        b_in => b_sig,
        sum => sum_sig
    );
    process
    begin
    a_sig <= "0000000000000000";
    b_sig <= "0000000000000000";
    wait for cp;
    a_sig <= "0000000000000001";
    b_sig <= "0000000000000001";
    wait for cp;
    a_sig <= "0000000000000010";
    b_sig <= "0000000000000010";
    wait for cp;
    a_sig <= "0000000000000011";
    b_sig <= "0000000000000011";
    wait for cp;
    a_sig <= "0000000011111111";
    b_sig <= "0000000011111111";
    wait for cp;
    a_sig <= "1000100000001111";
    b_sig <= "1111111111111111";
    wait for cp;
    a_sig <= "1111111111111111";
    b_sig <= "1111111111111111";
    wait;
    end process;
end Behavioral;
```

```
-- mac.vhd
-- ECE524 FALL2020
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity mac is
    Port (
        coef, a, b: in std_logic_vector(7 downto 0);
        output: out std_logic_vector(16 downto 0)
    );
    attribute use_dsp : string;
end mac;
architecture Behavioral of mac is
    component multiplier is
        Port (
            din,alpha: in std_logic_vector(7 downto 0);
            dout: out std_logic_vector(15 downto 0)
        );
    end component;
    component adder is
        Port (
            a_in, b_in: in std_logic_vector(15 downto 0);
            sum: out std_logic_vector(16 downto 0)
        );
    end component;
    signal mult_out1, mult_out2: std_logic_vector(15 downto 0);
    signal coef_not: std_logic_vector(7 downto 0);
    signal output_sig: std_logic_vector(16 downto 0);
    attribute use_dsp of multiplier,adder: component is "yes";
begin
    coef_not <= std_logic_vector("11111111" - unsigned(coef));
    MULT1: multiplier
    port map(
        din => a, alpha => coef,
        dout => mult_out1
    );
    MULT2: multiplier
    port map(
        din => b, alpha => coef_not,
        dout => mult_out2
    );
    ADD: adder
    port map(
        a_in => mult_out1,
        b_in => mult_out2,
        sum => output_sig);
    output <= output_sig;
end Behavioral;
```

```
-- mac_tb.vhd
-- ECE524 FALL2020
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity mac_tb is
--  Port ( );
end mac_tb;
architecture Behavioral of mac_tb is
component mac is
    Port (
        coef, a, b: in std_logic_vector(7 downto 0);
        output: out std_logic_vector(16 downto 0)
    );
end component;
signal coef_sig,a_sig,b_sig: std_logic_vector(7 downto 0);
signal sin_sig: unsigned( 7 downto 0);
signal output_sig: std_logic_vector(16 downto 0);
constant cp: time := 20 ns;
begin
    DUT: mac
    port map(
        coef => coef_sig,
        a => a_sig,
        b => b_sig,
        output => output_sig
    );
    SIN_GEN: process
    begin
        sin_sig <= to_unsigned(128,8);
        wait for cp/20;
        sin_sig <= to_unsigned(167,8);
        wait for cp/20;
        sin_sig <= to_unsigned(203,8);
        wait for cp/20;
        sin_sig <= to_unsigned(231,8);
        wait for cp/20;
        sin_sig <= to_unsigned(249,8);
        wait for cp/10;
        sin_sig <= to_unsigned(255,8);
        wait for cp/10;
        sin_sig <= to_unsigned(249,8);
        wait for cp/10;
        sin_sig <= to_unsigned(231,8);
        wait for cp/10;
        sin_sig <= to_unsigned(203,8);
        wait for cp/10;
        sin_sig <= to_unsigned(167,8);
        wait for cp/10;
        sin_sig <= to_unsigned(128,8);
        wait for cp/10;
        sin_sig <= to_unsigned(88,8);
        wait for cp/10;
        sin_sig <= to_unsigned(53,8);
        wait for cp/10;
        sin_sig <= to_unsigned(24,8);
        wait for cp/10;
```

```
        sin_sig <= to_unsigned(6,8);
        wait for cp/10;
        sin_sig <= to_unsigned(0,8);
        wait for cp/10;
        sin_sig <= to_unsigned(6,8);
        wait for cp/10;
        sin_sig <= to_unsigned(24,8);
        wait for cp/10;
        sin_sig <= to_unsigned(53,8);
        wait for cp/10;
        sin_sig <= to_unsigned(88,8);
        wait for cp/10;
    end process;

    STIM: process
    begin
    coef_sig <= "01111111";
    wait for 4*cp;
    coef_sig <= "00111111";
    wait for 4*cp;
    coef_sig <= "00001111";
    wait for 4*cp;
    wait;
    end process;

    b_sig <= std_logic_vector(sin_sig);
    a_sig <= "00000000";
end Behavioral;
```

```
-- multiplier.vhd
-- ECE 524 Fall 2020
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity multiplier is
    Port (
        din,alpha: in std_logic_vector(7 downto 0);
        dout: out std_logic_vector(15 downto 0)
    );
end multiplier;
architecture Behavioral of multiplier is
signal ualpha, udin: unsigned(7 downto 0);
begin
    ualpha <= unsigned(alpha);
    udin <= unsigned(din);
    dout <= std_logic_vector(ualpha*udin);
end Behavioral;
```

```
-- multiplier_tb.vhd
-- ECE524 FALL2020
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity multiplier_tb is
--  Port ( );
end multiplier_tb;
architecture Behavioral of multiplier_tb is
component multiplier is
    Port (
        din,alpha: in std_logic_vector(7 downto 0);
        dout: out std_logic_vector(15 downto 0)
    );
end component;
signal din_sig,alpha_sig: std_logic_vector(7 downto 0);
signal dout_sig: std_logic_vector(15 downto 0);
constant cp: time := 10 ns;
begin
    DUT: multiplier
    port map( din => din_sig,
        alpha => alpha_sig,
        dout => dout_sig
    );
    process
    begin
        din_sig <= "00000000";
        alpha_sig <= "00000000";
        wait for cp;
        din_sig <= "00000001";
        alpha_sig <= "00000001";
        wait for cp;
        din_sig <= "00000010";
        alpha_sig <= "00000010";
        wait for cp;
        din_sig <= "00000011";
        alpha_sig <= "00000011";
        wait for cp;
        din_sig <= "00000100";
        alpha_sig <= "00000100";
        wait for cp;
        din_sig <= "00010100";
        alpha_sig <= "00010100";
        wait for cp;
        din_sig <= "01000000";
        alpha_sig <= "11110000";
        wait for cp;
        din_sig <= "11111111";
        alpha_sig <= "11111111";
        wait;
    end process;
end Behavioral;
```

```
-- opt_mac.vhd
-- ECE524 FALL2020
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity opt_mac is
    Port (
        s0,s1,coeff: in std_logic_vector(7 downto 0);
        clk,clk2,clkhalf: in std_logic;
        Blend: out std_logic_vector(16 downto 0)
    );
end opt_mac;
architecture Behavioral of opt_mac is
component multiplier is
    Port (
        din,alpha: in std_logic_vector(7 downto 0);
        dout: out std_logic_vector(15 downto 0)
    );
end component;
component adder is
    Port (
        a_in, b_in: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(16 downto 0)
    );
end component;
signal s_in,s0_buf,s1_buf: std_logic_vector(7 downto 0);
signal alpha_buf, alpha_bufnot, alpha_in: std_logic_vector(7 downto 0);
signal mult_out: std_logic_vector(15 downto 0);
signal add_in: std_logic_vector(15 downto 0);
signal add_out,sum_buf: std_logic_vector(16 downto 0);
begin
    MULT: multiplier
    port map(
        din => s_in,
        alpha => alpha_in,
        dout => mult_out
    );
    ADD: adder
    port map(
        a_in => mult_out,
        b_in => add_in,
        sum => add_out
    );
    process(clk)
    begin
        if(rising_edge(clk)) then
            s1_buf <= s1;
            s0_buf <= s0;
            alpha_buf <= coeff;
            alpha_bufnot <= std_logic_vector("11111111"-unsigned(coeff));
            Blend <= sum_buf;
        end if;
    end process;
    process(clk2)
    begin
        if rising_edge(clk2) then
            sum_buf <= add_out;
```

```
        end if;
    end process;

    process(clkhalf)
    begin
    if(clkhalf = '0')then
        add_in <= (others => '0');
        alpha_in <= alpha_buf;
        s_in <= s0_buf;
    else
        add_in <= sum_buf(15 downto 0);
        alpha_in <= alpha_bufnot;
        s_in <= s1_buf;
    end if;
    end process;
end Behavioral;
```

```
-- opt_mac_tb.vhd
-- ECE524 FALL2020
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity opt_mac_tb is
--  Port ( );
end opt_mac_tb;
architecture Behavioral of opt_mac_tb is
component opt_mac is
    Port (
        s0,s1,coeff: in std_logic_vector(7 downto 0);
        clk,clk2,clkhalf: in std_logic;
        Blend: out std_logic_vector(16 downto 0)
    );
end component;
signal s0_sig, s1_sig, coef_sig: std_logic_vector(7 downto 0);
signal sin_sig: unsigned(7 downto 0);
signal clk_sig,clk2_sig,clkhalf_sig: std_logic := '0';
signal Blend_sig: std_logic_vector(16 downto 0);
constant cp: time := 40 ns;
begin
    DUT: opt_mac
    port map(
        s0 => s0_sig,
        s1 => s1_sig,
        coeff => coef_sig,
        clk => clk_sig,
        clk2 => clk2_sig,
        clkhalf => clkhalf_sig,
        Blend => Blend_sig
    );

    CLK: process(clk_sig)
    begin
        clk_sig <= not clk_sig after cp/20;
    end process;
    CLK2: process(clk2_sig)
    begin
    clk2_sig <= not clk2_sig after cp/40;
    end process;
    MODE_GEN: process(clkhalf_sig)
    begin
    clkhalf_sig <= not clkhalf_sig after cp/10;
    end process;

    STIM: process
    begin
    coef_sig <= "01111111";
    wait for 4*cp;
    coef_sig <= "00111111";
    wait for 4*cp;
    coef_sig <= "00001111";
    wait for 4*cp;
    wait;
    end process;
```

```
    SIN_GEN: process
    begin
        sin_sig <= to_unsigned(128,8);
        wait for cp/20;
        sin_sig <= to_unsigned(167,8);
        wait for cp/20;
        sin_sig <= to_unsigned(203,8);
        wait for cp/20;
        sin_sig <= to_unsigned(231,8);
        wait for cp/20;
        sin_sig <= to_unsigned(249,8);
        wait for cp/10;
        sin_sig <= to_unsigned(255,8);
        wait for cp/10;
        sin_sig <= to_unsigned(249,8);
        wait for cp/10;
        sin_sig <= to_unsigned(231,8);
        wait for cp/10;
        sin_sig <= to_unsigned(203,8);
        wait for cp/10;
        sin_sig <= to_unsigned(167,8);
        wait for cp/10;
        sin_sig <= to_unsigned(128,8);
        wait for cp/10;
        sin_sig <= to_unsigned(88,8);
        wait for cp/10;
        sin_sig <= to_unsigned(53,8);
        wait for cp/10;
        sin_sig <= to_unsigned(24,8);
        wait for cp/10;
        sin_sig <= to_unsigned(6,8);
        wait for cp/10;
        sin_sig <= to_unsigned(0,8);
        wait for cp/10;
        sin_sig <= to_unsigned(6,8);
        wait for cp/10;
        sin_sig <= to_unsigned(24,8);
        wait for cp/10;
        sin_sig <= to_unsigned(53,8);
        wait for cp/10;
        sin_sig <= to_unsigned(88,8);
        wait for cp/10;
    end process;

    s0_sig <= std_logic_vector(sin_sig);
    s1_sig <= "00000000";
end Behavioral;
```