Design of an Averager in FPGA

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 design an averaging circuit with parameters for the width of the data stream and the number of data samples. The experiment utilizes a delay line of m data samples. The delay line is implemented with shift register lookup tables or SRL components. The design also implements a logarithm function for calculating the number of bits required to shift right. Fig. 5.1 shows the block diagram for the averaging circuit.

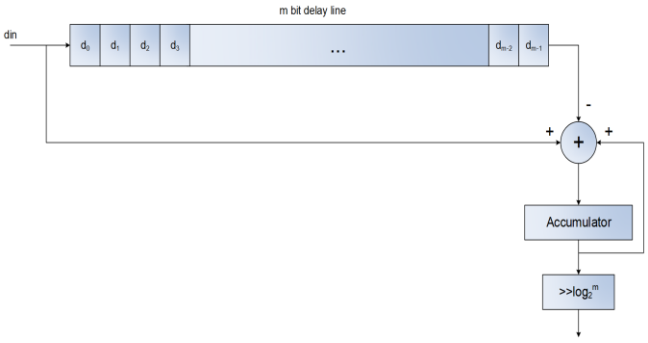

Fig 5.1 Block diagram of the averaging circuit

The Datapath for the circuit is modeled in (1). The data sample "d_{in}" is added to the previous value of the accumulator and the last value in the data line "d_{m-1}" is subtracted. This ensures that new values of the data stream replaces old values of the accumulator for the next calculation.

$$Accumulator = d_{in} + Accumulator - d_{m-1} \quad (1)$$

In order to get the complete addition of m bits, the circuit must wait for m clock cycles. After m clock cycles, the value stored in the accumulator is modeled by (2).

$$d_0 + d_1 + d_2 + \cdots + d_{m-2} + d_{m-1} \quad (2)$$

After the accumulator contains the correct sum of all the data samples, a shifter circuit is used to divide the sum by m. By shifting right by $\log_2(m)$ bits , the circuit is functionally equivalent to dividing by m as seen in (3).

$$\frac{d_0 + d_1 + d_2 + \cdots + d_{m-2} + d_{m-1}}{m} \quad (3)$$

In order to infer SRL components into the VHDL design, certain attributes must be used. The Xilinx synthesis guide details the specific attributes required for this experiment. The SHREG_EXTRACT attribute is used to infer SRL structures given values of YES or NO. The SRL_STYLE attribute infers tells the synthesis tool how to infer the SRL components. Fig. 5.2 shows a list of available values for the style. The values used for this experiment are "srl" and "block". Additionally, in order to infer SRL components, the code must not have a set or reset signal and must be "serial-in , serial-out".

SRL_STYLE

SRL_STYLE instructs the synthesis tool on how to infer SRLs that are found in the design. Accepted values are:

- `register`: The tool does not infer an SRL, but instead only uses registers.
- `srl`: The tool infers an SRL without any registers before or after.
- `srl_reg`: The tool infers an SRL and leaves one register after the SRL.
- `reg_srl`: The tool infers an SRL and leaves one register before the SRL.
- `reg_srl_reg`: The tool infers an SRL and leaves one register before and one register after the SRL.
- `block`: The tool infers the SRL inside a block RAM.

Fig. 5.2 Documentation of SRL synthesis

II. Proedure and Results

First, the shifter circuit was designed and tested. Fig. 5.3 shows the results of the shifter test bench with a chosen value of m = 4. This results in a shift of 2 bits to the right because $\log_2(4) = 2$. The test bench was done again for a value of m = 7. This time the input was shifted by 3 bits. This is the expected value because a shift of 3 is needed to represent the division of 7 samples. The logarithm function of base 2 used in this design will round up to the next power of 2. In this case, the division is rounded up to 2^3 or 8.

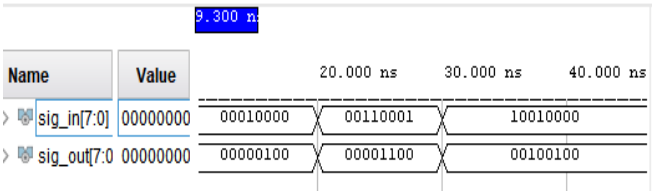

Fig. 5.3 Wave form results of the shifter test bench for m = 4

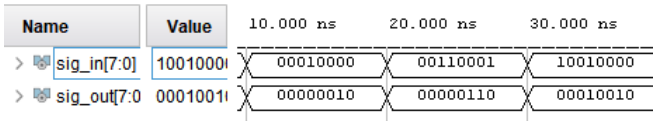

Fig. 5.4 Wave form results of the shifter test bench for m = 7

Next, the adder was designed and tested. The designed used unsigned representation. Fig 5.5 shows the results of the test bench. 4 sets of input vectors were tested. Each showing the correct operation of addition and subtraction. The last set of vectors shows an overflow in subtraction error where the data of the accumulator is less than the old data value being subtracted. While overflow error is possible, it is never expected in the design because it the accumulator should never be smaller than the total of the values in the delay line.

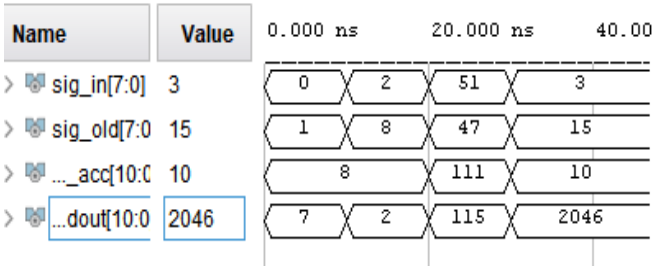

Fig. 5.5 Wave form results of the adder test bench

Next, the delay line was designed and tested. Fig. 5.6 shows the input vectors for the test bench. In Fig. 5.7, after 8 clock cycles the input is seen at the output. This shows the correct functionality of the delay line.

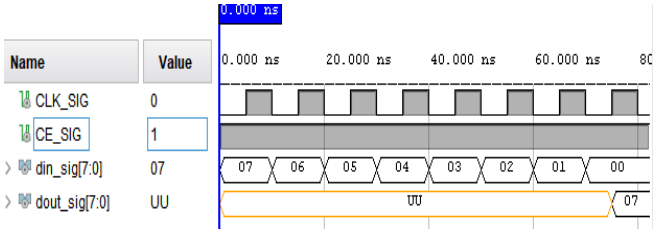

Fig 5.6 The input for the test bench of the delay line

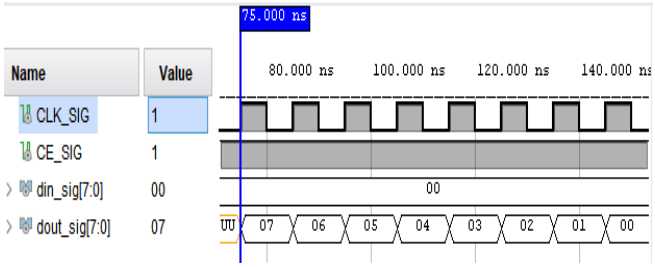

Fig 5.7 Output of the delay line test bench

The synthesis report for the delay line indicates that SRL components were inferred. Fig. 5.8 shows the slice logic section of the report. The synthesis of the SRL delay line was done again for powers of 2 up to a 64 bit delay line. Fig 5.9 shows a graph of the area utilization for each width.

```
1. Slice Logic
--------------

+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs                 |    4 |     0 |     17600 |  0.02 |
|   LUT as Logic             |    0 |     0 |     17600 |  0.00 |
|   LUT as Memory            |    4 |     0 |      6000 |  0.07 |
|     LUT as Distributed RAM |    0 |     0 |           |       |
|     LUT as Shift Register  |    4 |     0 |           |       |
| Slice Registers            |   16 |     0 |     35200 |  0.05 |
|   Register as Flip Flop    |   16 |     0 |     35200 |  0.05 |
|   Register as Latch        |    0 |     0 |     35200 |  0.00 |
| F7 Muxes                   |    0 |     0 |      8800 |  0.00 |
| F8 Muxes                   |    0 |     0 |      4400 |  0.00 |
+----------------------------+------+-------+-----------+-------+
```

Fig. 5.8 Slice logic of delay line implementation

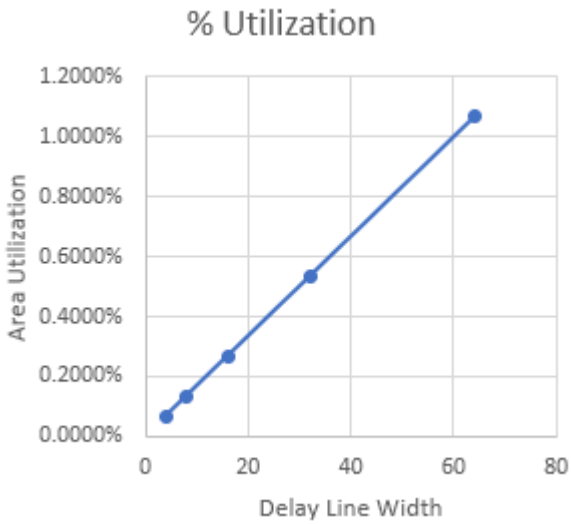

Fig. 5. 9 Area utilization per bit of delay line

Lastly, the averager circuit was put together and tested for a delay line of 4 bits and a width of 8 bits. For the first 4 clock cycles, the delay line was reset by input values of 0, and the accumulator was reset by holding the reset signal high. This process is shown in Fig. 5.10.

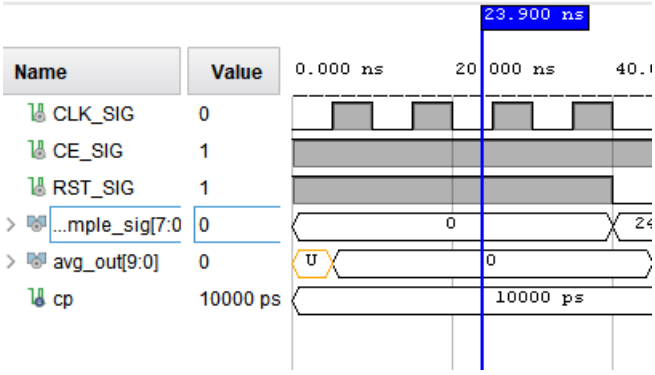

Fig. 5.10 Reset input for the first 4 clock cycles of the averager testbench

Fig. 5.11 shows the input data stream for the averager circuit. The first number input is 240. When clocked the output of the averager circuit becomes 60, which is the correct result when divided by 4. The next number added is 160 for a total of 400 in the accumulator. The correct result of 100 is displayed at the output. For the next input a value of 14 is added. The average of 414 is 103.5, but only 103 is displayed due to round errors. Futhermore, these test vectors show the correct functionallity but is not exhaustive. In order to sanely prove the functionallity of the circuit, a random number generator is introduced.

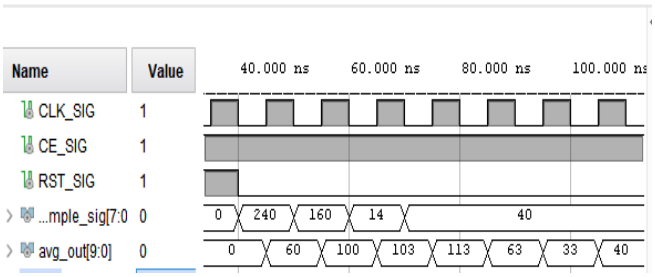

Fig. 5.11 Results of the Averager test bench

```
 -- averager.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.math_real.all;
entity averager is
    generic (W: integer := 8;M: integer := 4);
    Port(CLOCK,RESET,clk_en: in std_logic;
        sample_in: in std_logic_vector(W-1 downto 0);
        average: out std_logic_vector(integer(ceil(log2(real(M))))+W-1 downto 0)
    );
end averager;
architecture Structural of averager is
function clog2(x: positive) return integer is
    variable i: integer;
begin
    i := 0;
    while (2**i < x and i < 31) loop
        i := i + 1;
    end loop;
    return i;
end clog2;
signal SRL_OUT: std_logic_vector(W-1 downto 0);
signal ACC_OUT, ADD_OUT,AVG_OUT: std_logic_vector(clog2(M)+W-1 downto 0);
component SRL_REG is
    generic(w: integer := 8;m:integer := 8);
    Port(clk,ce: in std_logic;
        din: in std_logic_vector(w-1 downto 0);
        dout: out std_logic_vector(w-1 downto 0)
    );
end component;
component adder is
    generic(w,m: integer);
    Port(din: in std_logic_vector(w-1 downto 0);
        old: in std_logic_vector(w-1 downto 0);
        acc: in std_logic_vector(m-1 downto 0);
        dout: out std_logic_vector(m-1 downto 0)
    );
end component;
component acc_reg is
    generic (w: integer);
    Port (clk,rst: in std_logic;
        data_in: in std_logic_vector(w-1 downto 0);
        data_out: out std_logic_vector(w-1 downto 0)
    );
end component;
component shifter is
    generic( w,m: integer);
    Port (
        cin: in std_logic_vector(w-1 downto 0);
        cout: out std_logic_vector(w-1 downto 0)
    );
end component;
begin
    U1: SRL_REG
    generic map (w => W,m => M)
    port map(clk => CLOCK,
        ce => clk_en,
        din => sample_in,
```

```
        dout => SRL_OUT
    );
    U2: adder
    generic map (w => W,m => clog2(M)+W)
    port map(din => sample_in,
        old => SRL_OUT,
        acc => ACC_OUT,
        dout => ADD_OUT
    );
    U3: acc_reg
    generic map(w => clog2(M)+W)
    port map(clk => CLOCK,
        rst => RESET,
        data_in => ADD_OUT,
        data_out => ACC_OUT
    );
    U4: shifter
    generic map(w => clog2(M)+W,m => M)
    port map(
        cin => ACC_OUT,
        cout => AVG_OUT
    );
    average <= AVG_OUT;
end Structural;
```

```
-- averager_tb.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.math_real.all;
entity averager_tb is
--  Port ( );
end averager_tb;
architecture Behavioral of averager_tb is
component averager is
    generic (W: integer := 8;M: integer := 4);
    Port(CLOCK,RESET,clk_en: in std_logic;
        sample_in: in std_logic_vector(W-1 downto 0);
        average: out std_logic_vector(integer(ceil(log2(real(M))))+W-1 downto 0)
    );
end component;
signal CLK_SIG, CE_SIG, RST_SIG: std_logic := '0';
signal sample_sig: std_logic_vector(7 downto 0);
signal avg_out: std_logic_vector(9 downto 0);
constant cp: time := 10 ns;
begin
    DUT: averager
    generic map(W => 8,M => 4)
    port map(CLOCK => CLK_SIG,
        RESET => RST_SIG,
        clk_en => CE_SIG,
        sample_in => sample_sig,
        average => avg_out
    );
    process(CLK_SIG)
    begin
        CLK_SIG <= not CLK_SIG after cp/2;
    end process;
    process
    begin
        CE_SIG <= '1';
        RST_SIG <= '1';
        sample_sig <= "00000000";
        wait for 4*cp;
        RST_SIG <= '0';
        sample_sig <= "11110000";
        wait for cp;
        sample_sig <= "10100000";
        wait for cp;
        sample_sig <= "00001110";
        wait for cp;
        sample_sig <= "00101000";
        wait;
    end process;
end Behavioral;
```

```
-- shifter.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity shifter is
    generic( w,m: integer);
    Port (
        cin: in std_logic_vector(w-1 downto 0);
        cout: out std_logic_vector(w-1 downto 0)
    );
end shifter;
architecture Behavioral of shifter is
function clog2(x: positive) return integer is
    variable i: integer;
begin
    i := 0;
    while (2**i < x and i < 31) loop
        i := i + 1;
    end loop;
    return i;
end clog2;
signal ext: std_logic_vector (clog2(m)-1 downto 0);
begin
    ext <= (others => '0');
    cout <= ext & cin(w-1 downto clog2(m));
end Behavioral;
```

```
-- shifter_tb.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity shifter_tb is
--  Port ( );
end shifter_tb;
architecture Behavioral of shifter_tb is
component shifter is
    generic( w,m: integer);
    Port (
        cin: in std_logic_vector(w-1 downto 0);
        cout: out std_logic_vector(w-1 downto 0)
    );
end component;
signal sig_in, sig_out: std_logic_vector(7 downto 0);
constant cp: time := 10 ns;
begin
    DUT:shifter
    generic map(w => 8,m => 7)
    port map(
        cin => sig_in,
        cout => sig_out
    );
    process
    begin
        sig_in <= "00000000";
        wait for cp;
        sig_in <= "00010000";
        wait for cp;
        sig_in <= "00110001";
        wait for cp;
        sig_in <= "10010000";
        wait;
    end process;
end Behavioral;
```

```
-- adder.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity adder is
    generic(w,m: integer);
    Port(din: in std_logic_vector(w-1 downto 0);
        old: in std_logic_vector(w-1 downto 0);
        acc: in std_logic_vector(m-1 downto 0);
        dout: out std_logic_vector(m-1 downto 0)
    );
end adder;
architecture Behavioral of adder is
signal ext: unsigned(m-w-1 downto 0);
begin
    ext <= (others => '0');
    dout <= std_logic_vector(unsigned(acc)+(ext&unsigned(din))-(ext&unsigned(old)));
end Behavioral;
```

```
-- adder_tb.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity adder_tb is
--  Port ( );
end adder_tb;
architecture Behavioral of adder_tb is
component adder is
    generic(w,m: integer);
    Port(din: in std_logic_vector(w-1 downto 0);
        old: in std_logic_vector(w-1 downto 0);
        acc: in std_logic_vector(m-1 downto 0);
        dout: out std_logic_vector(m-1 downto 0)
    );
end component;
signal sig_in, sig_old: std_logic_vector(7 downto 0);
signal sig_acc, sig_dout: std_logic_vector(10 downto 0);
constant cp: time := 10 ns;
begin
    DUT: adder
    generic map(w => 8, m => 11)
    port map(
        din => sig_in,
        old => sig_old,
        acc => sig_acc,
        dout => sig_dout
    );
    process
    begin
        sig_in <= "00000000";
        sig_old <= "00000001";
        sig_acc <= "00000001000";
        wait for cp;
        sig_in <= "00000010";
        sig_old <= "00001000";
        sig_acc <= "00000001000";
        wait for cp;
        sig_in <= "00110011";
        sig_old <= "00101111";
        sig_acc <= "00001101111";
        wait for cp;
        sig_in <= "00000011";
        sig_old <= "00001111";
        sig_acc <= "00000001010";
        wait;
    end process;
end Behavioral;
```

```
-- SRL_REG.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity SRL_REG is
    generic(w: integer := 8;m:integer := 8);
    Port(clk,ce: in std_logic;
        din: in std_logic_vector(w-1 downto 0);
        dout: out std_logic_vector(w-1 downto 0)
    );
end SRL_REG;
architecture Behavioral of SRL_REG is
type reg_srl is array(m-1 downto 0) of std_logic_vector(w-1 downto 0);
signal tmp: reg_srl;
begin
    process(clk)
    begin
        if(ce = '1') then
            if rising_edge(clk) then
                tmp <= tmp(m-2 downto 0)&din;
            end if;
        end if;
    end process;
    dout <= tmp(m-1);
end Behavioral;
```

```
-- SRL_REG_tb.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity SRL_REG_tb is
--  Port ( );
end SRL_REG_tb;
architecture Behavioral of SRL_REG_tb is
component SRL_REG is
    generic(w: integer := 8;m:integer := 8);
    Port(clk,ce: in std_logic;
        din: in std_logic_vector(w-1 downto 0);
        dout: out std_logic_vector(w-1 downto 0)
    );
end component;
signal CLK_SIG,CE_SIG: std_logic := '0';
signal din_sig,dout_sig: std_logic_vector(7 downto 0);
constant cp: time := 10 ns;
begin
    DUT: SRL_REG
    generic map(w => 8, m => 8)
    port map(
        clk => CLK_SIG,
        ce => CE_SIG,
        din => din_sig,
        dout => dout_sig
    );
    process(CLK_SIG)
    begin
        CLK_SIG <= not CLK_SIG after cp/2;
    end process;
    process
    begin
        CE_SIG <= '1';
        din_sig <= "00000111";
        wait for cp;
        din_sig <= "00000110";
        wait for cp;
        din_sig <= "00000101";
        wait for cp;
        din_sig <= "00000100";
        wait for cp;
        din_sig <= "00000011";
        wait for cp;
        din_sig <= "00000010";
        wait for cp;
        din_sig <= "00000001";
        wait for cp;
        din_sig <= "00000000";
        wait;
    end process;
end Behavioral;
```