The Data path and Control path

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 consideration for the data path and control path. Fig. 4.1 shows the block diagram for the averaging circuit. The data path is the flow of data stream within the design. The 8 bit data is read from memory with an address provided by the counter. The data flows from the “a” register through the adder and into the sum register. The sum register accumulates the next 31 8-bit values and then outputs an average of the 32 bit numbers.

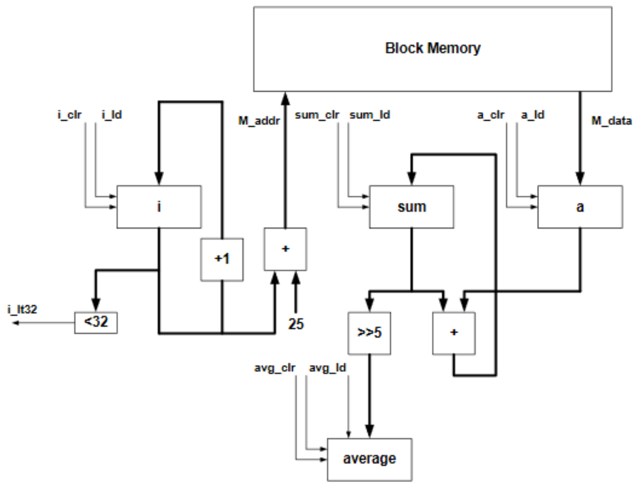

Fig 4.1 Block diagram of the averaging circuit

In order to control this data stream, an FSM was designed for the control unit. The FSM controls the load and clear operations for each register. Fig. 4.2 shows the block diagram of the control unit. The only input is a logic signal that indicates that the address counter is greater than 32, at which point all 32 8-bit values have been read into register “a”.

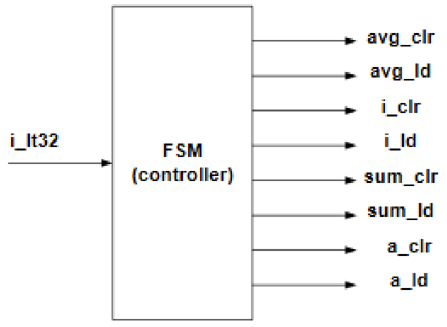

Fig. 4.2 Block diagram for the control unit

Each output of the control unit sends a signal to each stage of the datapath. Fig. 4.3 shows the State diagram of the FSM. Looking closely, the initial stage is a CLEAR state, where each register is first reset to 0. If an input of 1 is received, there is a glitch in the input, therefore the state remains the same. If an input of 0 is received, the next state is the ADD_NEXT state where the next addr_count is loaded, the first value is read from memory. Since, there was a value of 0 in register ‘a’, the sum register was loaded with 0 + 0. As long as input “i_lt32” is 0, then the ADD_NEXT state will continue to accumulate the next 31 values. Once the counter reaches 32, the last 8 bit value has been read from the memory. The last value still needs to be summed, therefore an additional LAST_SUM state is required. At this point, an input of ‘0’ is ideally impossible, therefore an input of 0 will return to the CLEAR state, and input of 1 will continue to the LOAD_AVG state. In the LOAD_AVG state, the only output is the “avg_ld” signal. From here, the machine will return to the clear state to start the cycle over.

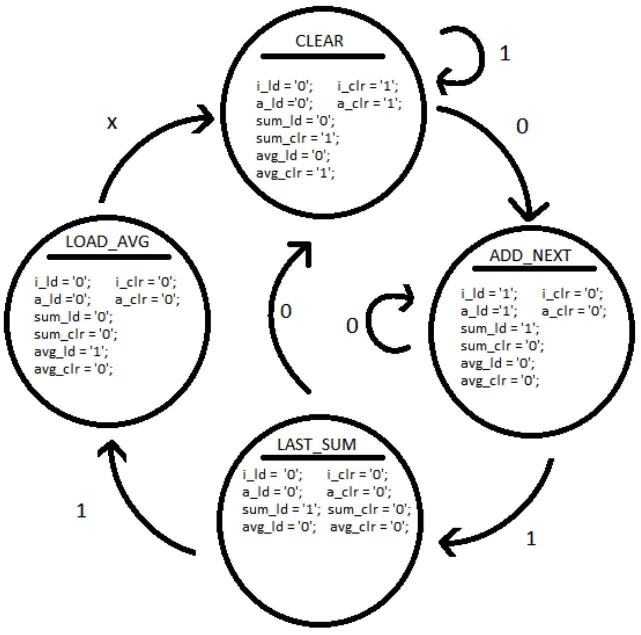

Fig. 4.3 State diagram of FSM

II. Procedure and Results

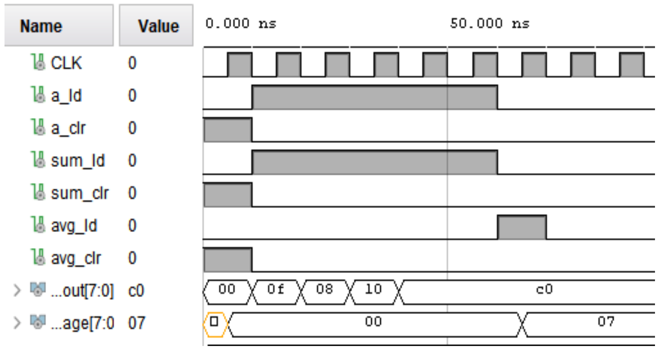

Fig. 4.3 Wave form results for the Accumulator

First, the accumulator was designed and tested. Fig. 4.3 shows the testbench results of the accumulator design. At 5 ns, the registers were cleared. Then at 10 ns, the “a_ld” and “sum_ld” signals were set to ‘1’. For the next 5 cycles, the values “0f”, “08”, “10”, and “c0” were added together. Notice that the last value took two clock cycles to reach the sum register. In this example, these values add up to $(231)_{10}$. These values are shifted right 5 times which results in an output of 7. This is the expected result.

Lastly, the average circuit was modified to find the average of 50 8-bit values. In order to make these changes, the address counter must count to 50 instead of 32. In order to fit the summation of all 50 values, the accumulator must be at least 14 bits wide. The last change needed is the average logic. This is done by multiplying the sum by 5 then dividing bits by 256. Because this operation is not the same as dividing by 50, there is rounding error. (1) shows that the rounding error is about 2.3%. This error can be reduced by increasing the number of bits used in the multiplication process.

$$\frac{\left(\left(\frac{5}{256}\right)-\left(\frac{1}{50}\right)\right)}{\frac{1}{50}} = 2.3\% \qquad (1)$$

```vhdl
-- averager_tb.vhd
-- Ridge Tejuco
-- ECE 524 Fall 2020
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.NUMERIC_STD.ALL;
entity averager_tb is
--  Port ( );
end averager_tb;
architecture Behavioral of averager_tb is
component averager is
    Port(
        clock_in,reset,mem_en: in std_logic;
        avg_out: out std_logic_vector(7 downto 0)
    );
end component;
signal CLK,RST,MEM_ENA: std_logic := '0';
signal AVG: std_logic_vector(7 downto 0);
constant CP: time := 10 ns;
begin
    DUT: averager
    port map(
        clock_in => CLK,
        reset => RST,
        mem_en => MEM_ENA,
        avg_out => AVG
    );
    CLK_GEN: process(CLK)
    begin
        CLK <= not CLK after CP/2;
    end process;
    STIM: process
    begin
        RST <= '1';
        MEM_ENA <= '1';
        wait for CP;
        RST <= '0';
        wait;
    end process;
end Behavioral;
```

```
-- averager.vhd
-- Ridge Tejuco
-- ECE 524 Fall 2020
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity averager is
    Port (
        clock_in,reset,mem_en: in std_logic;
        avg_out: out std_logic_vector(7 downto 0)
    );
end averager;
architecture Behavioral of averager is
signal i_lt32, i_ld, i_clr: std_logic;
signal acc_ld, acc_clr,s_ld,s_clr: std_logic;
signal average_ld,average_clr: std_logic;
signal m_addr: std_logic_vector(5 downto 0);
signal mem_out: std_logic_vector(7 downto 0);
component addr_counter is
    Port (clock,clear,load: in std_logic;
        lt32: out std_logic;
        addr: out std_logic_vector (5 downto 0)
    );
end component;
component accumulator is
    port(
        clk,a_load,a_clear: in std_logic;
        sum_load, sum_clear: in std_logic;
        avg_load, avg_clear: in std_logic;
        DIN: in std_logic_vector(7 downto 0);
        AVG: out std_logic_vector(7 downto 0)
    );
end component;
component fsm is
    Port(
        in_lt32,clk,rst: in std_logic;
        in_ld,in_clr: out std_logic;
        a_ld,a_clr: out std_logic;
        sum_ld, sum_clr: out std_logic;
        avg_ld, avg_clr: out std_logic
    );
end component;
component blk_mem_gen_0 IS
  PORT (
    clka : IN STD_LOGIC;
    ena : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
end component;
begin
    INC: addr_counter
    port map(
        clock => clock_in,
        clear => i_clr,
        load => i_ld,
        lt32 => i_lt32,
        addr => m_addr
    );
    ACC: accumulator
    port map(
        clk => clock_in,
        a_load => acc_ld,
        a_clear => acc_clr,
        sum_load => s_ld,
        sum_clear => s_clr,
        avg_load => average_ld,
        avg_clear => average_clr,
        DIN => mem_out,
        AVG => avg_out
    );
    CNTL: fsm
    port map(
        in_lt32 => i_lt32,
        clk => clock_in,
        rst => reset,
        in_ld => i_ld,
        in_clr => i_clr,
        a_ld => acc_ld,
        a_clr => acc_clr,
        sum_ld => s_ld,
        sum_clr => s_clr,
        avg_ld => average_ld,
        avg_clr => average_clr
    );
    MEM: blk_mem_gen_0
    port map(
        clka => clock_in,
        ena => mem_en,
        addra => m_addr,
        douta => mem_out
    );
end Behavioral;
```

```
-- fsm.vhd
-- Ridge Tejuco
-- ECE 524 Fall 2020
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity fsm is
    Port(
        in_lt32,clk,rst: in std_logic;
        in_ld,in_clr: out std_logic;
        a_ld,a_clr: out std_logic;
        sum_ld, sum_clr: out std_logic;
        avg_ld, avg_clr: out std_logic
    );
end fsm;
architecture Behavioral of fsm is
TYPE state_type is (CLEAR,ADD_NEXT,LAST_SUM,LOAD_AVG);
signal NEXT_STATE,STATE: state_type;
begin
    process(clk)
    begin
        if(rst = '1') then
            STATE <= CLEAR;
        elsif rising_edge(clk) then
            STATE <= NEXT_STATE;
        end if;
    end process;

    -- next state/output logic
    process(STATE,in_lt32)
    begin
        case STATE is
        when CLEAR =>
            if(in_lt32 = '1') then
                NEXT_STATE <= CLEAR;
            else
                NEXT_STATE <= ADD_NEXT;
            end if;

            in_clr <= '1';
            a_clr <= '1';
            sum_clr <= '1';
            avg_clr <= '1';
            in_ld <= '0';
            a_ld <= '0';
            sum_ld <= '0';
            avg_ld <= '0';

        when ADD_NEXT =>
            if(in_lt32 = '0') then
                NEXT_STATE <= ADD_NEXT;
            else
                NEXT_STATE <= LAST_SUM;
            end if;

            in_clr <= '0';
            a_clr <= '0';
            sum_clr <= '0';
            avg_clr <= '0';
            in_ld <= '1';
            a_ld <= '1';
            sum_ld <= '1';
            avg_ld <= '0';
        when LAST_SUM =>
            if(in_lt32 = '0') then
                NEXT_STATE <= CLEAR;
            else
                NEXT_STATE <= LOAD_AVG;
            end if;

            in_clr <= '0';
            a_clr <= '0';
            sum_clr <= '0';
            avg_clr <= '0';
            in_ld <= '0';
            a_ld <= '0';
            sum_ld <= '1';
            avg_ld <= '0';

        when others =>
            NEXT_STATE <= CLEAR;
            in_clr <= '0';
            a_clr <= '0';
            sum_clr <= '0';
            avg_clr <= '0';
            in_ld <= '0';
            a_ld <= '0';
            sum_ld <= '0';
            avg_ld <= '1';
        end case;
    end process;
end Behavioral;
```

```vhdl
-- addr_counter.vhd
-- Ridge Tejuco
-- ECE 524 Fall 2020
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity addr_counter is
    Port (clock,clear,load: in std_logic;
        lt32: out std_logic;
        addr: out std_logic_vector (5 downto 0)
    );
end addr_counter;
architecture Behavioral of addr_counter is
signal cnt_out: std_logic_vector(5 downto 0);
signal inc: unsigned (5 downto 0);
component counter is
    port(
        clk,clr,ld: in std_logic;
        DIN: in std_logic_vector(5 downto 0);
        DOUT: out std_logic_vector(5 downto 0)
    );
end component;
begin
    inc <= unsigned(cnt_out) + "000001"; -- + 1
    addr <= std_logic_vector((unsigned(cnt_out)) + "011001"); -- + 25
    CNT: counter
    port map(
        clk => clock,
        clr => clear,
        ld => load,
        DIN => std_logic_vector(inc),
        DOUT => cnt_out
    );
    process(cnt_out)
    begin
        if(unsigned(cnt_out) < "100000") then
            lt32 <= '1';
        else
            lt32 <= '0';
        end if;
    end process;
end Behavioral;
```

```vhdl
-- accumulator.vhd
-- Ridge Tejuco
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity accumulator is
    port(
        clk,a_load,a_clear: in std_logic;
        sum_load, sum_clear: in std_logic;
        avg_load, avg_clear: in std_logic;
        DIN: in std_logic_vector(7 downto 0);
        AVG: out std_logic_vector(7 downto 0)
    );
end accumulator;
architecture Behavioral of accumulator is
component register_a is
    generic(N: integer := 7);
    Port (
        CLK,RST,LD: in std_logic;
        DIN: in std_logic_vector(N downto 0);
        DOUT: out std_logic_vector(N downto 0)
    );
end component;
signal a: std_logic_vector(7 downto 0);
signal sum: std_logic_vector(12 downto 0);
signal a_sum: unsigned(12 downto 0);
signal shift_sum: std_logic_vector(7 downto 0);
begin
    a_sum <= unsigned("00000"&a) + unsigned(sum);
    shift_sum <= sum(12 downto 5);
    REG_A: register_a
    generic map(N => 7)
    port map(
        CLK => clk,
        RST => a_clear,
        LD => a_load,
        DIN => DIN,
        DOUT => a
    );
    REG_SUM: register_a
    generic map(N => 12)
    port map(
        CLK => clk,
        RST => sum_clear,
        LD => sum_load,
        DIN => std_logic_vector(a_sum),
        DOUT => sum
    );
    REG_AVG: register_a
    generic map(N => 7)
    port map(
        CLK => clk,
        RST => avg_clear,
        LD => avg_load,
        DIN => shift_sum,
        DOUT => AVG
    );
end Behavioral;
```

```vhdl
-- accumulator_tb.vhd
-- Ridge Tejuco
-- ECE 524 Fall 2020
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.NUMERIC_STD.ALL;
entity accumulator_tb is
--  Port ( );
end accumulator_tb;
architecture Behavioral of accumulator_tb is

signal CLK: std_logic := '0';
constant CP: time := 10 ns;
signal a_ld, a_clr: std_logic;
signal sum_ld, sum_clr: std_logic;
signal avg_ld, avg_clr: std_logic;
signal mem_out: std_logic_vector(7 downto 0);
signal average: std_logic_vector(7 downto 0);
component accumulator is
    port(

        clk,a_load,a_clear: in std_logic;
        sum_load, sum_clear: in std_logic;
        avg_load, avg_clear: in std_logic;
        DIN: in std_logic_vector(7 downto 0);
        AVG: out std_logic_vector(7 downto 0)
    );
end component;
begin
    DUT: accumulator
    port map(
        clk => CLK,
        a_load => a_ld, a_clear => a_clr,
        sum_load => sum_ld, sum_clear => sum_clr,
        avg_load => avg_ld, avg_clear => avg_clr,
        DIN => mem_out, AVG => average
    );
    CLK_GEN: process(CLK)
    begin
        CLK <= not CLK after CP/2;
    end process;
    STIM: process
    begin
        a_clr <= '1';
        sum_clr <= '1';
        avg_clr <= '1';
        a_ld <= '0';
        sum_ld <= '0';
        avg_ld <= '0';
        mem_out <= "00000000";
        wait for CP;
        a_clr <= '0';
        sum_clr <= '0';
        avg_clr <= '0';

        a_ld <= '1';
        sum_ld <= '1';
        mem_out <= "00001111";
        wait for CP;
        mem_out <= "00001000";
        wait for CP;
        mem_out <= "00010000";
        wait for CP;
        mem_out <= "11000000";
        wait for 2*CP;
        a_ld <= '0';
        sum_ld <= '0';
        avg_ld <= '1';
        wait for CP;
        avg_ld <= '0';
        wait;
    end process;
end Behavioral;
```

```
-- register_a.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity register_a is
    generic(N: integer := 7);
    Port (
        CLK,RST,LD: in std_logic;
        DIN: in std_logic_vector(N downto 0);
        DOUT: out std_logic_vector(N downto 0)
    );
end register_a;
architecture Behavioral of register_a is
begin
    process(CLK)
    begin
        if rising_edge(CLK) then
            if(RST = '1') then
                DOUT <= (others => '0');
            elsif LD = '1' then
                DOUT <= DIN;
            end if;
        end if;
    end process;
end Behavioral;
```