Fall 2019

California State University, Northridge

Department of Electrical & Computer Engineering



Lab Experiment 4

*Datapath & Control Path*

Juan Silva & Tumen Khureltulga

(Group 16)

October 29, 2019

ECE 524L

**Table of Contents**

**I. Introduction and Problem Statements…….………………………...…Page 2**

**II. Procedure……..…………………………………………….…………...Page 2**

**III. Testing Strategy………………………………….....………………......Page 4**

**IV. Results…………....…………………………………....…………….......Page 4**

**V. Analysis…………....……………………………....……………….……Page 6**

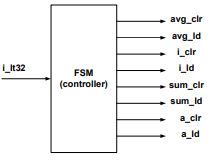
**VI. Appendix……………...………………………....………………….….Page 7**

1. **Introduction and Problem Statements**

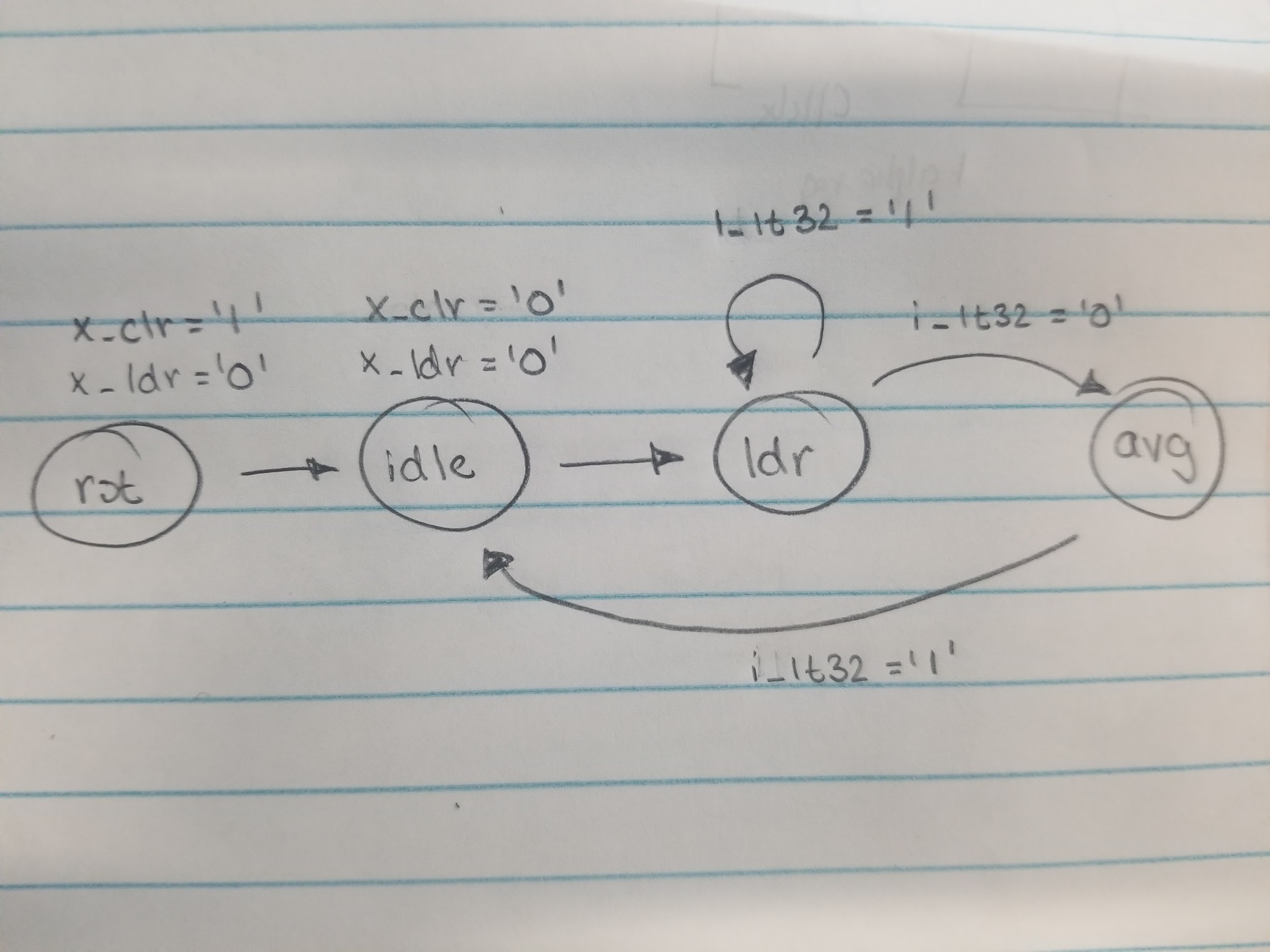
In this lab experiment, a circuit is designed that will calculate the average of 32 eight-bit numbers. These numbers are stored into memory for reading and summed until 32 numbers are loaded into ROM. Then, the average is taken and the process will repeat again. The purpose of this lab is to model this circuit using a control path and data path. The control path will determine the activity of the circuit while the datapath models the flow of the data. This is done to simplify the circuit design to smaller components. The datapath can be constructed using sequential logic for the ROM and combinational logic for the summer and counter. The control path can be constructed using a finite state machine (FSM).

1. **Procedure**

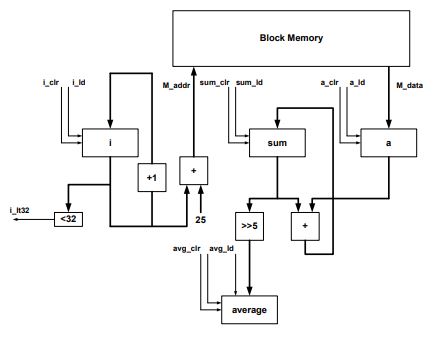
The block RAM is first designed to ensure that data is stored and read correctly to set up the circuit. The IP catalog is used to infer a single-port RAM and a coefficient file (COE) is used to store 8-bit data onto RAM. Once the memory is instantiated properly, the remaining part of the circuit is modeled using structural modeling. Given the finite state machine block diagram, the design for the control unit is shown in *Figures 1.1* and *Figure 1.2* respectively. Each state has a function based on the control signal that is assigned to that state. Four states are used to compute the average of 32 numbers: RESET, IDLE, LOAD, and AVERAGE. Each state will be driven by a control path and will follow a datapath.   
 The datapath begins with a counter that will count how many times a number is added into memory shown in *Figure 4.3*. The top level diagram If 32 numbers aren’t stored into memory, then the counter will increment until the condition is met. Next, an adder will take the sum of the value read from memory. This process will repeat until the average control signal is set. The average is taken and the cycle is complete. In the RESET state, all the control signals are set and datapath signals are initialized to zero.Afterwards, it transitions to the next state IDLE, where it starts the counter. If the counter is less than 32, then the state will transition to the LOAD state where the sum and memory loading occurs. Overflow is checked while the sum is being calculated.The LOAD state will repeat until the control signal for 32 numbers total transitions to the AVERAGE state. The average is then calculated outputs the solution. The state changes back to IDLE and the process repeats all over again.



**Figure 4.1 -** State Machine Block Diagram



**Figure 4.2 -** Finite State Machine State Diagram

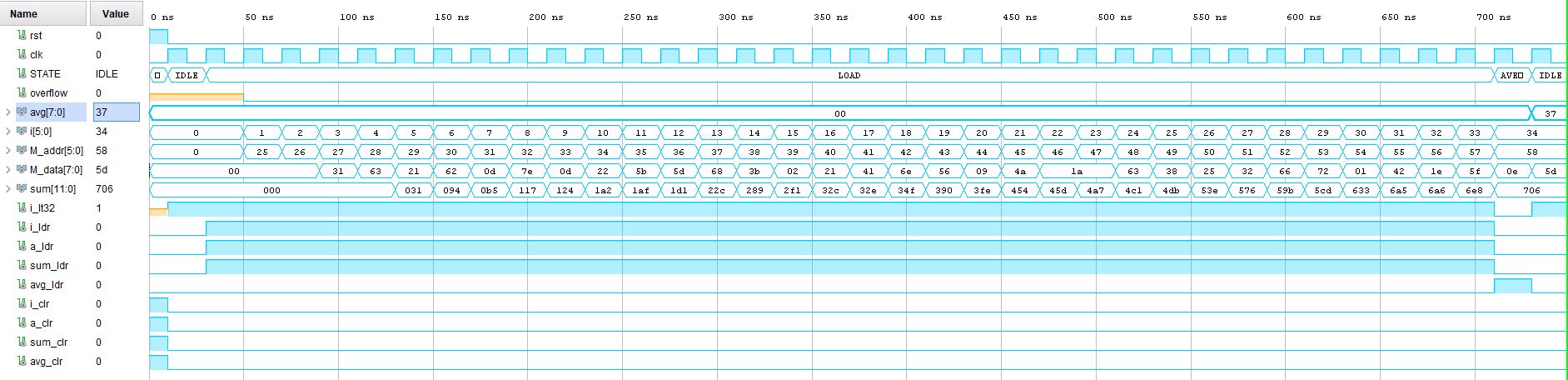


**Figure 4.3 -** Top level circuit design displaying control path and data path..

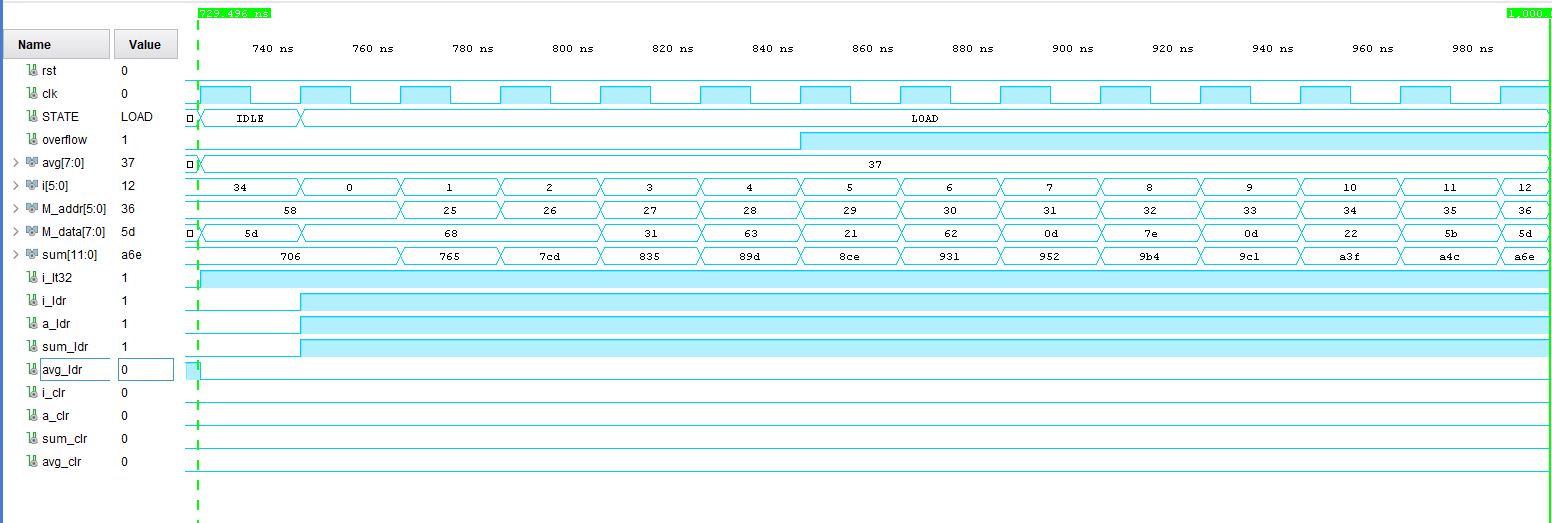
1. **Testing Strategy**

To verify that the memory functions correctly, a quick testbench is used to generate a waveform and view the data being read from RAM. To verify that each state transition is accessible, the same testbench sets the reset signal to high in order to initialize the FSM and cycle through each state. Both of these results are visible in the waveform in *Figure 4.4*.   
 To verify the average is calculated correctly, an Excel sheet containing the same values from the COE file was created and shown in *Table 4.1*. The cells in base16 are converted to base10 and the sum is taken. This sum is converted back to base16 and used for comparison with the waveform output. If both numbers match, then the circuit is correct. Overflow was not detected with the first iteration of averaging. If the circuit continued taking the average a second time, overflow occurs which is shown in *Figure 4.5*.

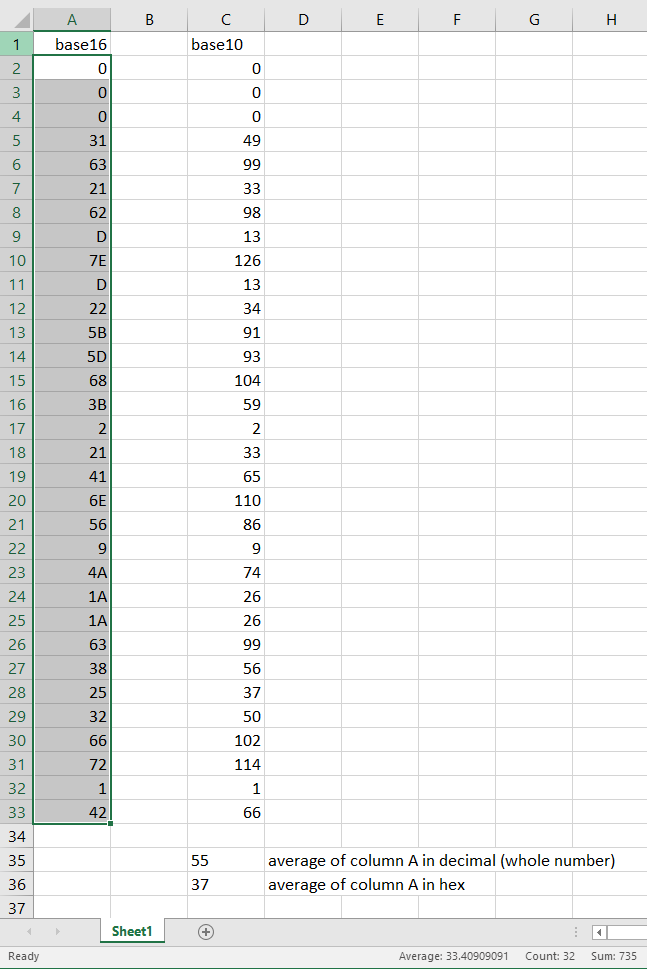
1. **Results & Data**



**Figure 4.4 -** Generated Waveform for Average of 32 Numbers.



**Figure 4.5 -** Overflow Detection



**Table 4.1 -** Excel Spreadsheet to Verify Average Calculation.

1. **Analysis**

Breaking down the circuit into a control path and datapath serves multiple uses. The top level model can be represented by a datapath where data flows and a control path that assigns behavior at a specified point. The use of a control unit manages all activities of the circuit, in this case, loading numbers, taking the sum, taking the average, and clearing. States serve a beneficial purpose by defining a proper flow of data. In the LOAD state, data is circulated from memory onto an accumulator and the process repeats until the control signal detects a change in activity such as checking if 32 numbers are loaded onto the accumulator.   
 Division by 50 was an objective that was required but unable to be completed. Code was written for the averaging of 50 numbers but was unsuccessful in producing a valid result. A similar testing procedure was used to test for the average of 50 numbers using a COE and Excel file but the test failed. That being said, using the divisor operator in VHDL would produce valid results visible in the testbench. However, this operation is unsynthesizable because no hardware can be inferred. Division by 2^N can infer a N-bit shifter. However, a number like 50 simply cannot infer any hardware. The correct implementation would use a MUX where the select line would be a comparator that checks if the remainder is less than the divisor. Otherwise, the number will be subtracted by 50 until that condition is true. This scheme is synthesizable as a MUX, a subtractor, and a comparator all infer hardware.

1. **Appendix**

| **Source Codes** | **Testbench** |
| --- | --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  entity computeAverage is  Port ( clk : in STD\_LOGIC;  rst : in STD\_LOGIC;  overflow : out STD\_LOGIC;  avg : out STD\_LOGIC\_VECTOR(7 downto 0));  end computeAverage;  architecture fsm of computeAverage is  --instantiate memory  COMPONENT blk\_mem\_gen\_0  PORT (  clka : IN STD\_LOGIC;  addra : IN STD\_LOGIC\_VECTOR(5 DOWNTO 0);  douta : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0)  );  END COMPONENT;  --Define states for FSM  type t\_STATE is (RESET, IDLE, LOAD, AVERAGE);  signal STATE : t\_STATE;  --Data signals  signal index: unsigned(5 downto 0) := "000000"; --memory index addres  signal data: unsigned(7 downto 0); --data read from memory  signal sum\_ovfl : std\_logic\_vector(12 downto 0); --checks overflow  signal sum: unsigned(11 downto 0); --max size for sum of 8 bit + sign bit  signal i: unsigned(5 downto 0); --counter for i signal  signal temp: std\_logic\_vector(11 downto 0) := (others => '0'); --sum value to be shifted by 5 bits  --Control signals  signal i\_lt32 : std\_logic; --checks if 32 numbers added  signal i\_ldr, a\_ldr, sum\_ldr, avg\_ldr: std\_logic; --loading signals for each state  signal i\_clr, a\_clr, sum\_clr, avg\_clr: std\_logic; --clearing signals for each state  --Component signals  signal M\_addr: STD\_LOGIC\_VECTOR(5 downto 0) := "000000"; --RAM address  signal M\_data: STD\_LOGIC\_VECTOR(7 downto 0); --RAM data  begin  UUT: blk\_mem\_gen\_0  PORT MAP (clka => clk, addra => M\_addr, douta => M\_data);    process(clk, rst)  begin  --Initialization when reset applied  if(rst = '1') then  i\_ldr <= '0'; a\_ldr <= '0'; sum\_ldr <= '0'; avg\_ldr <= '0';  i\_clr <= '1'; a\_clr <= '1'; sum\_clr <= '1'; avg\_clr <= '1';    i <= (others => '0'); --set count to 0.  sum <= (others => '0'); --set sum to 0.  data <= "00000000";  avg <= (others => '0');  STATE <= RESET;    elsif(rising\_edge(clk)) then    case STATE is    --Initial conditions  when RESET =>    i <= (others => '0'); --set count to 0.  sum <= (others => '0'); --set sum to 0.  data <= "00000000";    i\_lt32 <= '1'; --i < 32 is true  i\_clr <= '0'; a\_clr <= '0'; sum\_clr <= '0'; avg\_clr <= '0';  STATE <= IDLE;    --Idle state to prepare for LOAD state  when IDLE =>  i <= (others => '0');  index <= (others => '0');  --Jump to LOAD if 32 numbers aren't loaded  if(i\_lt32 = '1') then  i\_ldr <= '1'; a\_ldr <= '1'; sum\_ldr <= '1'; avg\_ldr <= '0';  STATE <= LOAD;  else  STATE <= IDLE;  end if;    --Read from RAM and accumulate  when LOAD =>    --Increment, read, then add  if(i\_ldr = '1' and a\_ldr = '1' and sum\_ldr = '1') then    --Increment counter and index  i <= i + 1;  index <= index + 1;    --Read from memory  M\_addr <= std\_logic\_vector(index + 25);  data <= unsigned(M\_data);    --Accumulate & shift  sum\_ovfl <= std\_logic\_vector(sum(11) & sum);    --Divide by 32  temp <= std\_logic\_vector(sum srl 5);    --Divide by 50.  --if(sum > 50) then  --temp <= std\_logic\_vector(sum - 50);  --else  --avg <= temp;  --end if  sum <= sum + data;    if(sum\_ovfl(12) = '1') then  overflow <= '1';  else  overflow <= '0';  end if;  else  sum <= sum;  data <= data;  i <= i;  index <= index;  end if;    --Check if 32 numbers loaded  if(i > "00100000") then  --if(i > "‭00110010‬") then  i\_ldr <= '0'; a\_ldr <= '0'; sum\_ldr <= '0'; avg\_ldr <= '1';  i\_lt32 <= '0';  STATE <= AVERAGE;  end if;    --Average is calculated  when AVERAGE =>    if(avg\_ldr = '1') then  avg <= temp(7 downto 0);  i\_lt32 <= '1';  i\_ldr <= '0'; a\_ldr <= '0'; sum\_ldr <= '0'; avg\_ldr <= '0';  STATE <= IDLE;  else  sum <= sum;  STATE <= AVERAGE;  end if;  when others =>  STATE <= RESET;  end case;  end if;    end process;  end fsm; | library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  entity tb\_average is  -- Port ( );  end tb\_average;  architecture Behavioral of tb\_average is  component computeAverage is  Port ( clk : in STD\_LOGIC;  rst : in STD\_LOGIC;  overflow : out std\_logic;  avg : out STD\_LOGIC\_VECTOR(7 downto 0));  end component;  signal clk : std\_logic := '0';  signal avg : std\_logic\_vector(7 downto 0);  signal rst : std\_logic := '0';  signal overflow : std\_logic := '0';  constant clk\_pulse : time := 20 ns;  begin  UUT: computeAverage  port map(clk => clk, avg => avg, rst => rst, overflow => overflow);    rst <= '1', '0' after clk\_pulse / 2;    --test states  clock: process(clk)  begin  clk <= not clk after clk\_pulse / 2;  end process clock;  end Behavioral; |

| **samples32.coe** | **samples50.coe** |
| --- | --- |
| memory\_initialization\_radix=16;  memory\_initialization\_vector=  0,  0,  0,  31,  63,  21,  62,  D,  7E,  D,  22,  5B,  5D,  68,  3B,  2,  21,  41,  6E,  56,  9,  4A,  1A,  1A,  63,  38,  25,  32,  66,  72,  1,  42, | memory\_initialization\_radix=16;  memory\_initialization\_vector=  0,  0,  0,  1E,  5F,  E,  5D,  68,  50,  78,  53,  6F,  1A,  28,  37,  36,  31,  63,  21,  62,  D,  7E,  D,  22,  5B,  5D,  68,  3B,  2,  21,  41,  6E,  56,  9,  4A,  1A,  1A,  63,  38,  25,  32,  66,  72,  1,  42,  B4,  3,  34,  92,  81; |