LUND UNIVERSITY

LTH

FACULTY OF ENGINEERING

ETIN 35

IC project 1

Final Report

Team 10

Hanyu Liu

March 11, 2022

Abstract

In this report, we have designed a hardware accelerator that focuses on matrix multiply, the average value on diagonal elements, and maximum value. To finish our project, we use the VHDL code to implement the hardware. To reduce the area, two multiplier units (MU) were implemented in the design. The ModelSim should be used to synthesize the RTL design and Encounter to place and route. Finally, the top design should be re-run in ModelSim to finish the post-synthesis.

Contents

1. Introduction

In this project, two matrices, input, and coefficient need to be multiplied together. The matrix of input has 4 rows and 8 columns. The matrix of coefficient has 8 rows and 4 columns. Each matrix has 32 elements inside, and the result will be posted as a matrix with 16 elements. The product of matrix multiplication is specified as equation (1). Two memory modules are used to store the coefficients and the results respectively. The input matrix is stored in the register as it is often read and overwritten. The multiplier is constructed using VHDL, and then synthesized, placed, and routed to complete the ASIC.

$$P(n) = X(n)A \qquad (1)$$

X is the element of the input matrix. A is the element of the coefficient matrix. P is the element in the result matrix, which is equal to X times A. The input matrix and coefficient matrix are specified in the following format.

$$Input = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} & x_{1,7} & x_{1,8} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} & x_{2,7} & x_{2,8} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} & x_{3,7} & x_{3,8} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} & x_{4,6} & x_{4,7} & x_{4,8} \end{bmatrix}$$

$$Coeff = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} \\ a_{8,1} & a_{8,2} & a_{8,3} & a_{8,4} \end{bmatrix}$$

To figure out one correct result which should be added by 8 multiplications. An example of multiplication has been shown in equation (2).

$$p_{1,1} = x_{1,1}a_{1,1} + x_{1,2}a_{2,1} + x_{1,3}a_{3,1} + x_{1,4}a_{4,1} + x_{1,5}a_{5,1} + x_{1,6}a_{6,1} + x_{1,7}a_{7,1} + x_{1,8}a_{8,1} \qquad (2)$$

The result of the matrix will be calculated by the previous equation and stored in the matrix of the result. The format of the result matrix will be shown as follows which size is 4x4.

$$Result = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \\ p_{4,1} & p_{4,2} & p_{4,3} & p_{4,4} \end{bmatrix}$$

After calculation, all values should be stored in the matrix and the hardware implementation can find the maximum value in this matrix and the mean value of diagonal elements. The generalized equation of mean value has been shown in equation (3). In our case, the M should be equal to 4 due to the size of the result matrix. After computing all the results, the maximum value will be found as well.

$$avg = \frac{1}{M} \sum_{i=1}^{M} p_{i,i} \quad (3)$$

As the instruction specified, each entry in the input matrix is an eight-bit unsigned number and each entry in coefficients in the matrix is a seven-bit unsigned number. For the multiplication result, it takes 7+8=15 bits. And the final result is the sum of eight multiplication results which results in the total width of each result entry being 18 bits. Because the RAM with 32 bits word length is selected, thus, one memory address can only store one result.

2. Implementation

2.1 Block diagram

Figure 2.1 has shown the block diagram in this project. There are eight modules in this matrix multiplier, “load_coeff”, “load_input”, “operation”, “store_result”, “max_value”, “controller”, and two SRAMs. There are two inputs and three outputs in this project, “data_in”, “start”, “ld2mem”, “ld2reg” and “data_out”.

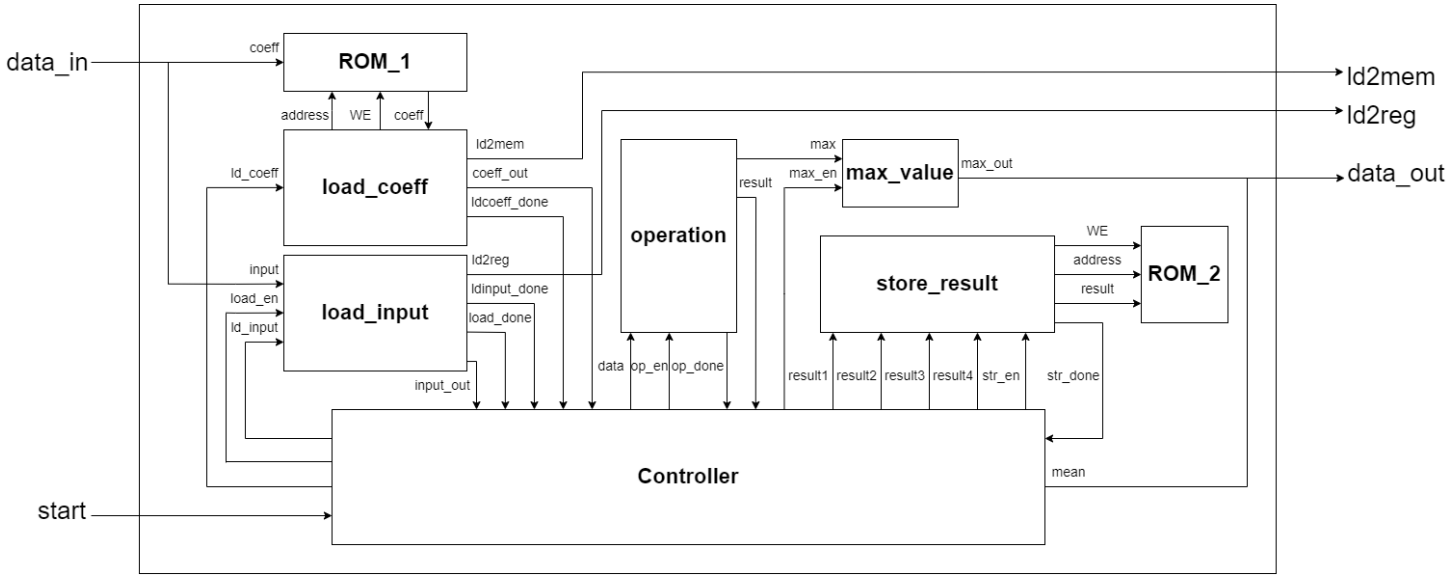

Figure 2.1: the block diagram of matrix multiplier

“data_in” is a 14-bit input port that can read both coefficient and input data. “start” is a 1-bit input enable signal which will start to compute matrix multiply after active high. “ld2mem” is a 1-bit output enable signal which will start to load coefficient through port “data_in” after active high. “ld2reg” is a 1-bit output enable signal which will start to load input data through port “data_in” after active high. “data_out” is a 19-bit output port which can send out both maximum value and mean value of diagonal elements. The following parts will introduce the function of each module.

2.2 ASMD charts

2.2.1 Controller

Figure 2.2.1 has shown the ASMD chart of the “controller”. There are eight states in this module. The controller will control all the flow and send out the average value of diagonal elements.

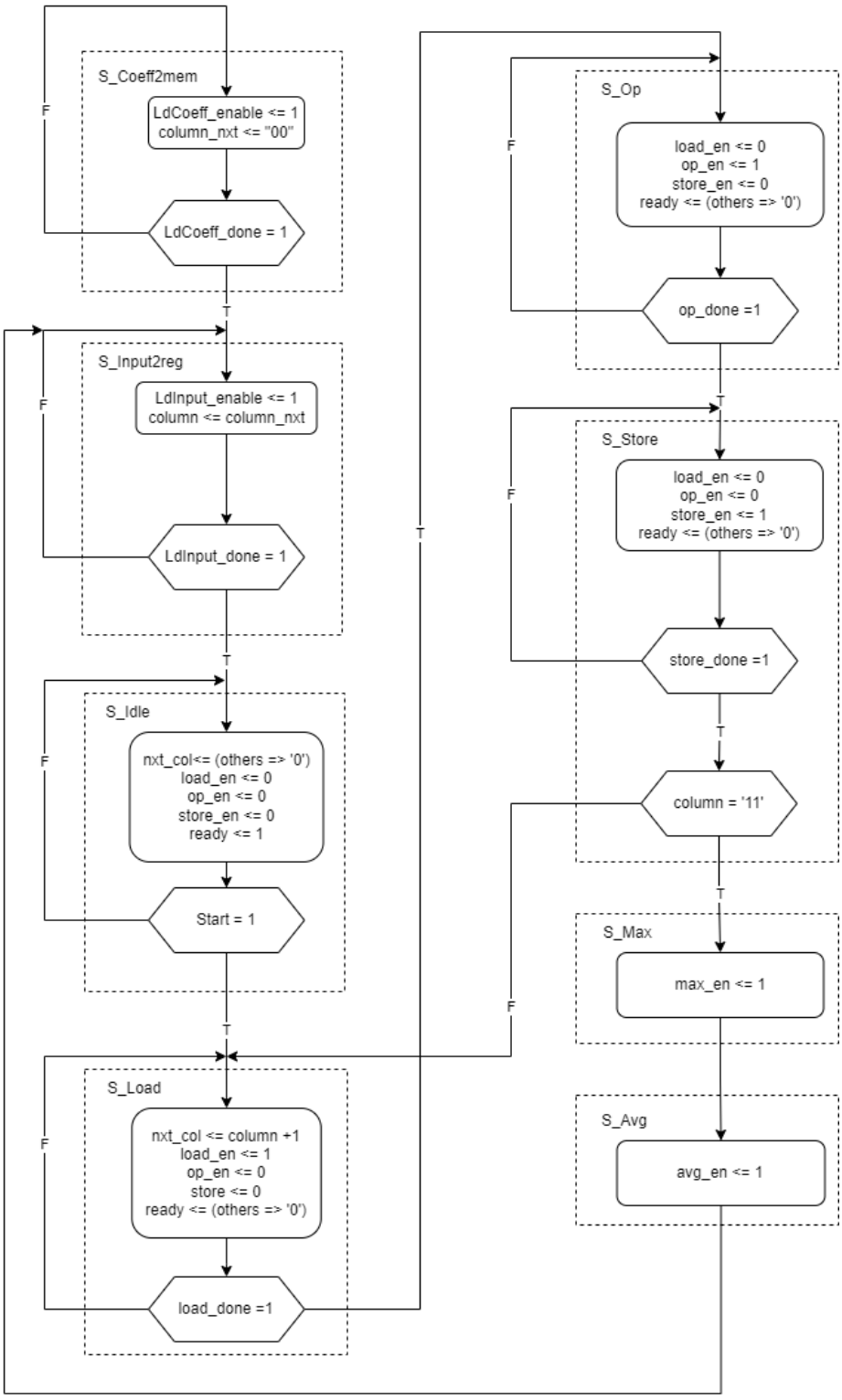

Figure 2.2.1: the ASMD chart of “controller”

S_Coeff2mem:

In this state, the controller will active signal “LdCoeff_enable” in High. This signal will drive the matrix multiplier to read the coefficients from a file to the main memory (RAM). Once the reading is finished, the controller will receive a signal which is “LdCoeff_done”.

S_Input2reg:

In this state, the controller will active signal “LdInput_enable” in High. This signal will drive the matrix multiplier to read the input data from a file directly to the registers. Once the reading is finished, the controller will receive a signal which is “LdInput_done”.

S_Idle:

The purpose of state “S_Idle” is to reset all the variables and transfer a “ready” signal out which means the matrix multiplier is ready to calculate the matrix. Once the input signal “Start” is High, the controller can move on to the next state unless it will hold on to this state.

S_Load:

This state will load the coefficient from the main memory into registers by column. The length of a coefficient is 7 bits. Thus, each address can hold two coefficients. There are 8 coefficients in one column which mean the loading will be repeated 4 times. The signal “load_done” will be received when 8 coefficients are already loaded into the registers, then the controller will move on to the next state.

S_Op:

In this state, the signal “op_en” will be active High. The controller will receive the signal “op_ready” High when multiplying is done. The 4 results in a column will be held in this state.

S_Store:

In this state, the 4 results will be stored in the main memory. At the end of this state, there is a judgment that will compare the value of the variable “column”. The variable “column” holds the value of the next column number. If the “column” is greater than or equal to 1 and less than 5, which means the program is still not finished and the controller will return to the state “S_Load”. Otherwise, the controller will move on to the next state.

S_Max:

In this state, the controller will send signal “max_en” High to show the maximum value in the result 4*4 matrix.

S_Avg:

In this state, the controller will send signal “avg_en” High to show the average of the value of the diagonal elements in the result 4*4 matrix. After processing, the controller will return to the state of “S_Idle”.

2.2.2 Load coefficient

Figure 2.2.2 has shown the ASMD chart of “load coeff”. There are four states in this module. It can write all coefficients into SRAM and load the coefficients into registers.

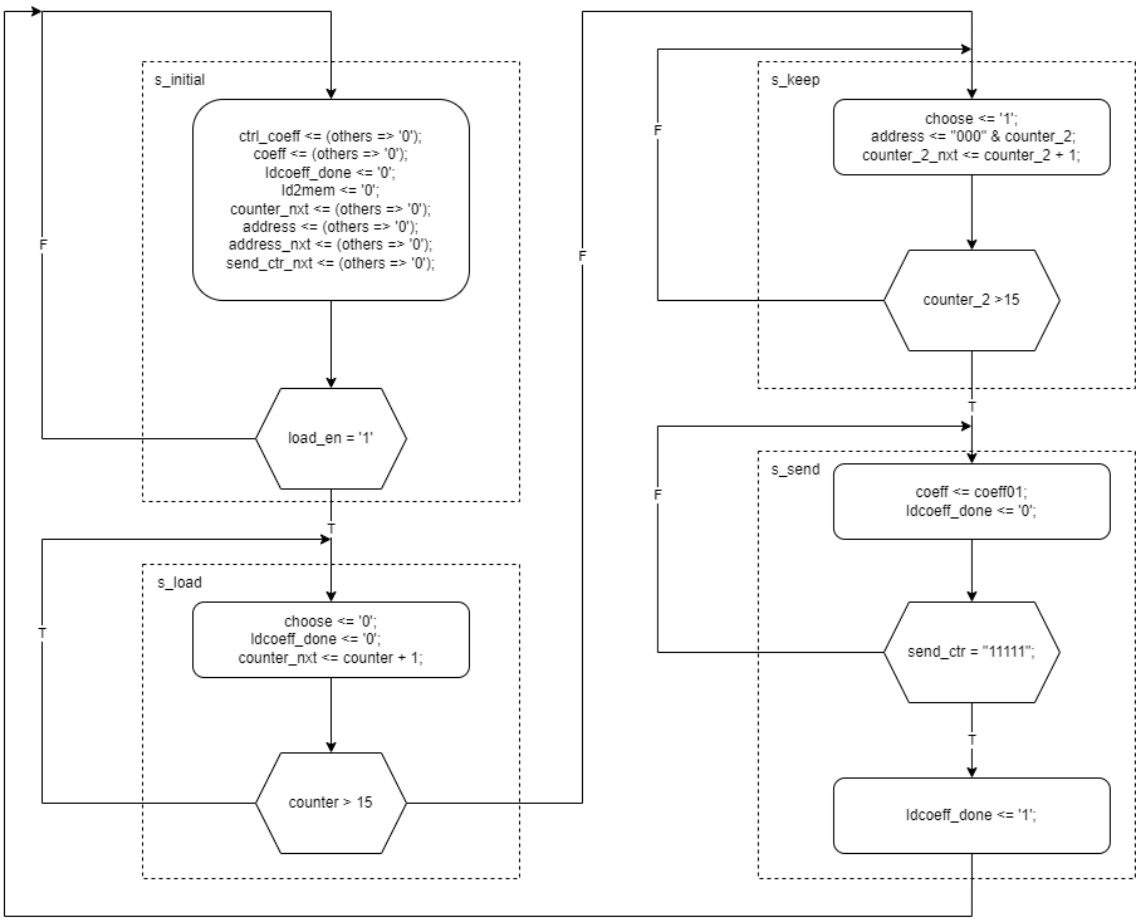

Figure 2.2.2: the ASMD chart of “load coefficient”

S_initial:

In this state, it will set all signals into default values. And it will jump to the next state after receiving “load_en” high from the controller.

S_load:

In this state, this module will send out the address and enable the write function for SRAM. To load 32 coefficients, it will only need to send sixteen addresses because two coefficients are in one address.

S_keep:

In this state, this module will send out the address and enable the read function for SRAM. It will load the coefficients into registers.

S_send:

For this state, it will send the coefficients into the controller one by one.

2.2.3 Load input

Figure 2.2.3 has shown the ASMD chart of “load input”. There are three states in this module. It will read the input data from the input port and store the input data into registers. S_initial state sets all signals as default values. S_keep state which will store the input data into registers. And the s_send state will send the input data to the controller.

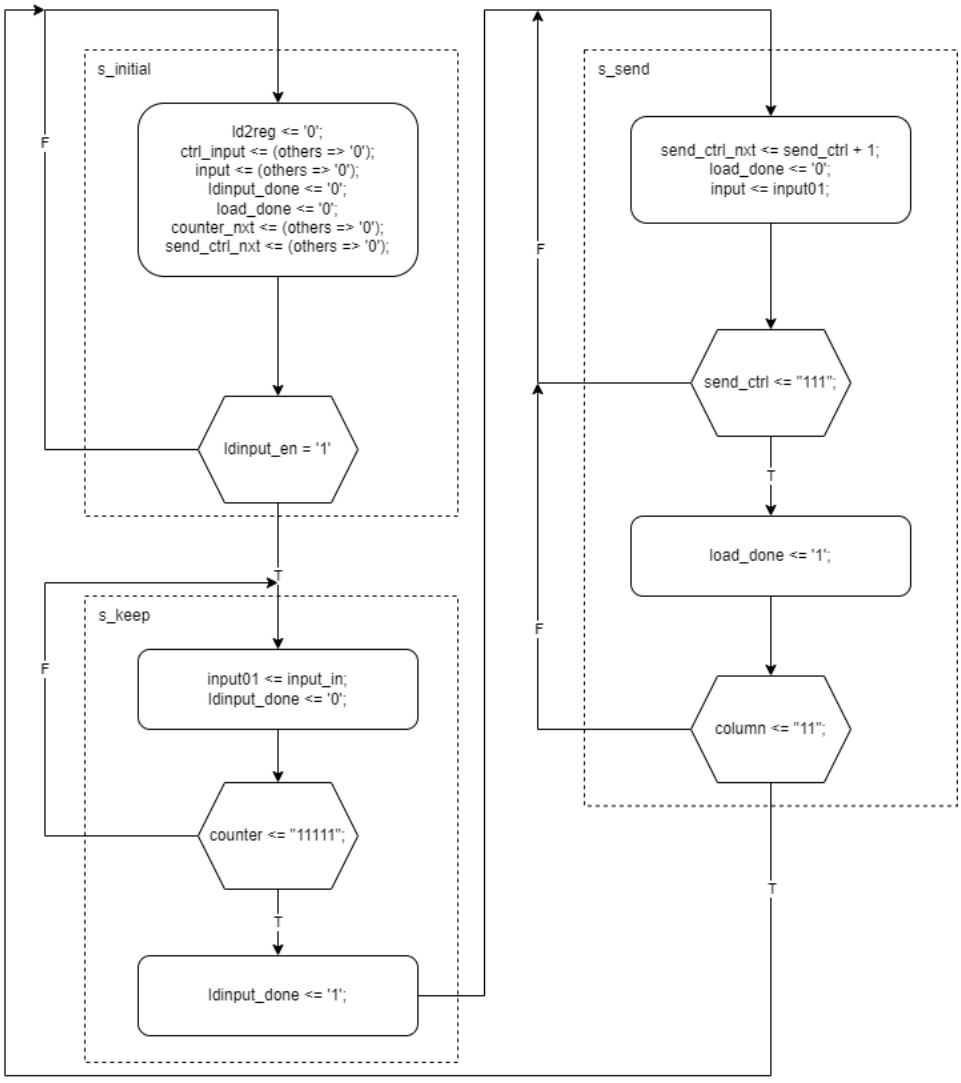

Figure 2.2.3: the ASMD chart of “load input”

2.2.4 Operation

Figure 2.2.4 has shown the ASMD chart of “operation”. Two multipliers are allowed to be used in this project. There are thirteen states in this module. S_store state will hold all the values that need to

be computed from the controller. After a series of computing, it will send back the results to the controller. in this module, it can also compare the results and find the maximum value.

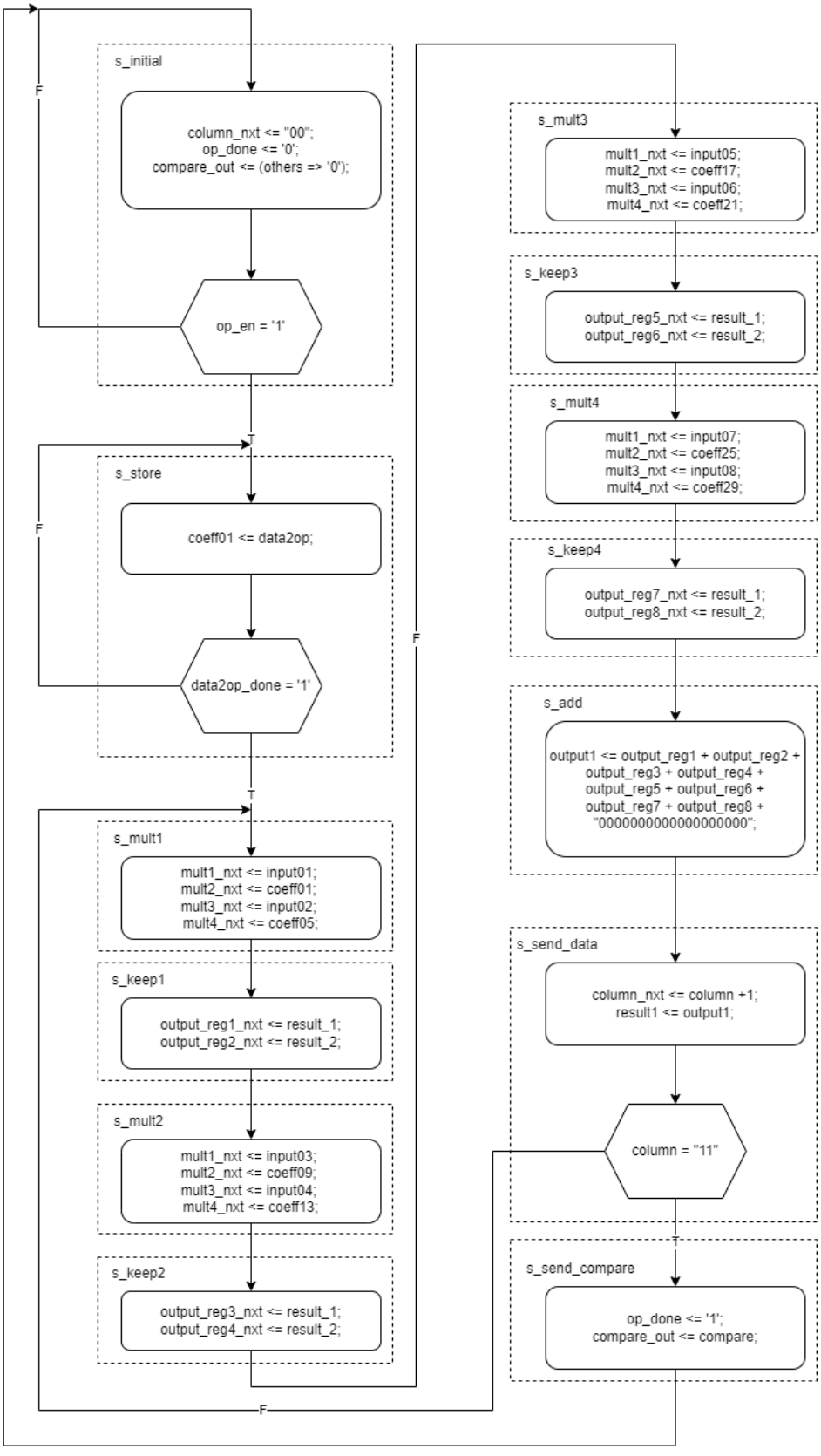

Figure 2.2.4: the ASMD chart of “operation”

2.2.5 Store result

In this part, figure 2.2.5 has shown the ASMD chart of “store result”. There are six states in this module.

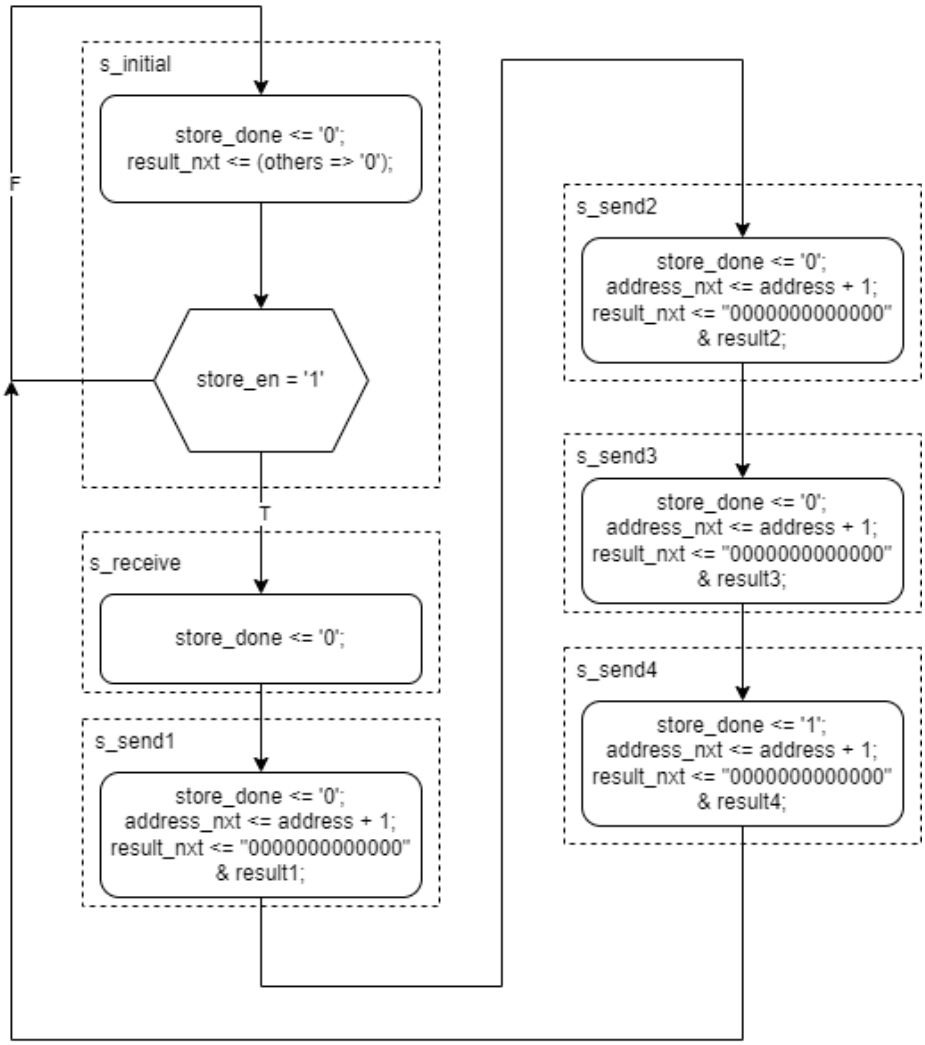

Figure 2.2.5: the ASMD chart of “store result”

S_initial:

This state will set all signals into the default value.

S_receive:

This state will hold all the results from the controller.

S_send1, S_send2, S_send3, S_send4:

For these states, it will send the results into SRAM and give the store address at the same time. The function of the write enable for SRAM will always be activated.

2.2.6 Comparison

Figure 2.2.6 has shown the ASMD chart of “comparison”. For this part, there are only two states in this module. S_compare will always receive the maximum result from the operation. It will send out the maximum value after receiving “max_en” is high.

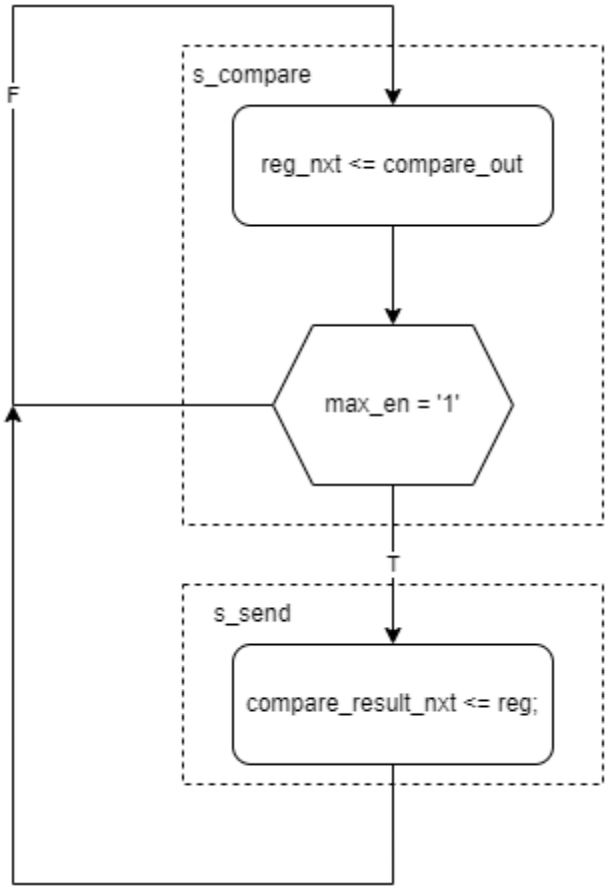

Figure 2.2.6: the ASMD chart of “comparison”

3. Synthesis

To reduce power consumption 65LPHVT libraries are used because high Vt can reduce leakage current to reduce static power consumption. The clock has been defined as shown in table 3.1. And the timing report and area report also has been shown in Tables 3.2 and 3.3.

Clock Period	10ns (100MHz)
Clock Skew	500ps
Clock Rise Time	100ps
Clock Fall Time	110ps
Clock Source Latency	500ps
Clock Network Latency	500ps

Table 3.1: clock parameters and constrains

Slack	6725ps
Critical Path	3275ps

Table 3.2: timing report

Total area(um^2)	206181

Table 3.3: area report

Considering the slack is 6725ps (Table 3.2), which occupies 67% of the clock period, there is no need to reduce the clock frequency even if the slack will decrease when doing place and route.

4. Verification

Firstly, the design can be simulated in Vivado. There is no need to consider the time and area, just prove the function. The result has been shown in figure 4.1 which can output the correct result and find the maximum value. The input matrix, coefficient matrix, and result matrix can be found in the appendix. The result in Figure 4.2 also is posted as a correct result. This simulation is from the post-synthesis. From figure 4.1 and 4.2 we can see that the results in hexadecimal are the same.

And figure 4.3 shows the first result 006E in hexadecimal can be stored in the memory. Figure 4.4 shows the thirteenth result 024E in hexadecimal can be stored in memory as well. Thus, the results can be stored in memory successfully.

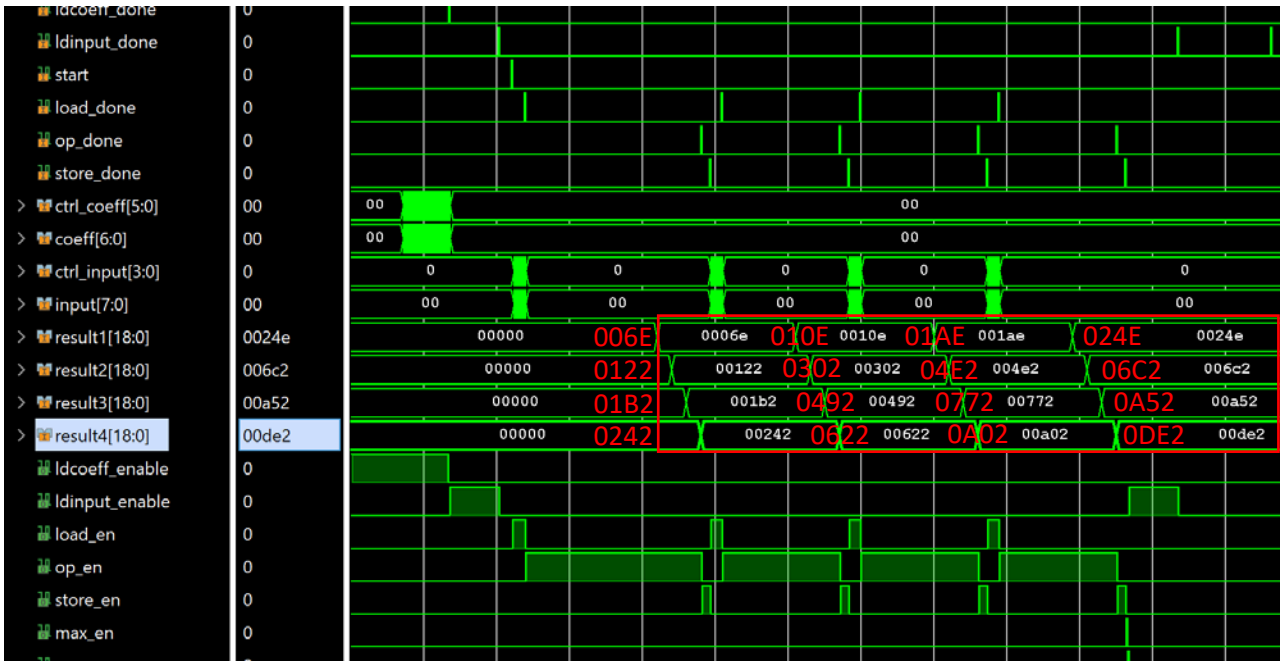

Figure 4.1: simulation result from Vivado

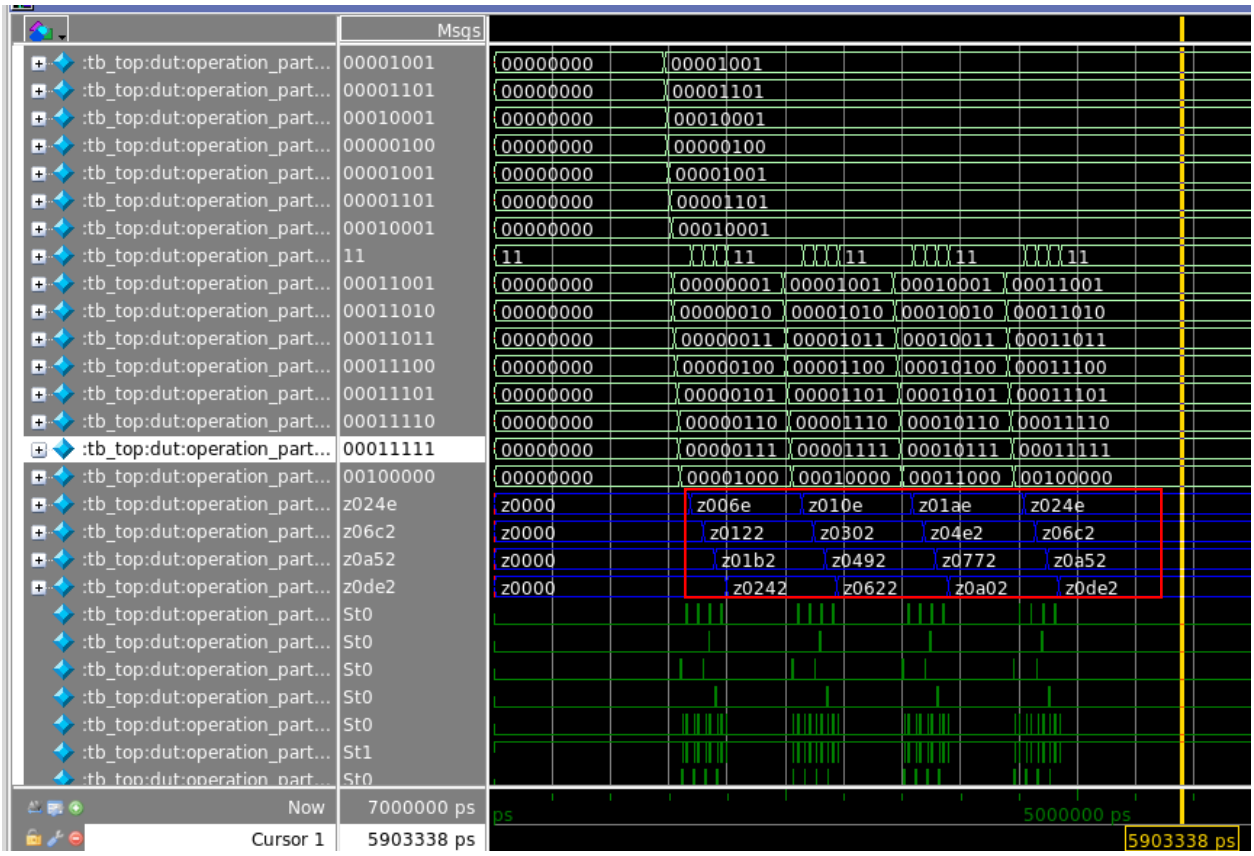

Figure 4.2: post-synthesis result in ModelSim

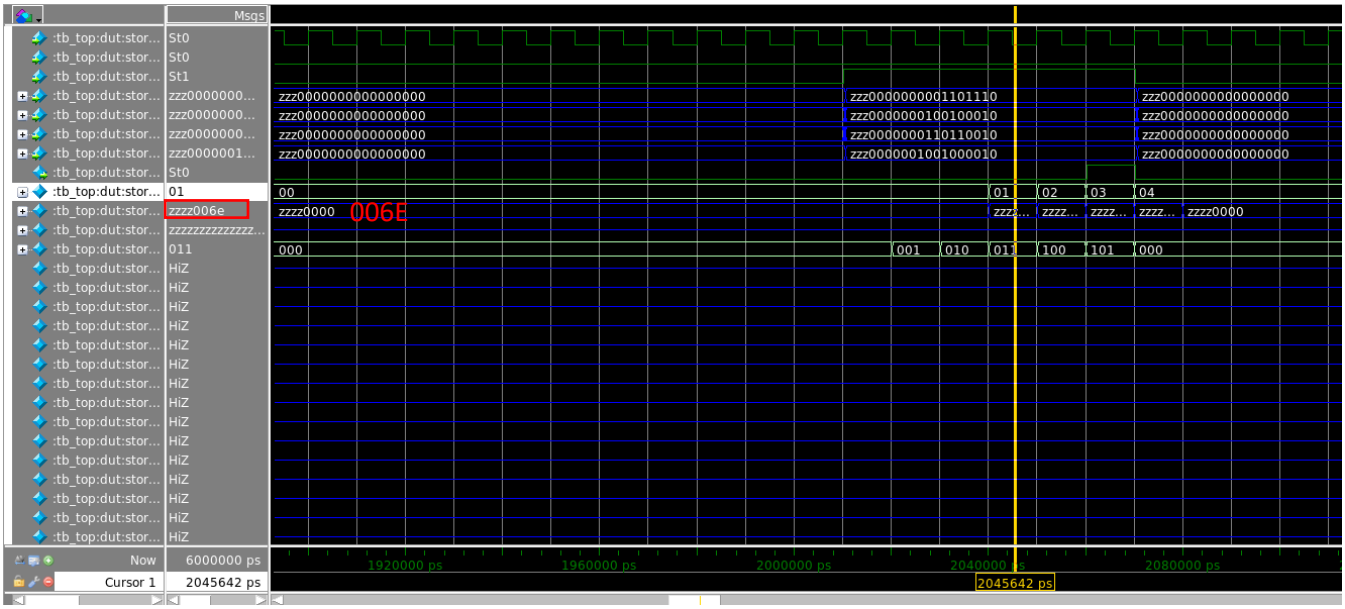

Figure 4.3: the first result 006E in hexadecimal can be stored in memory successfully

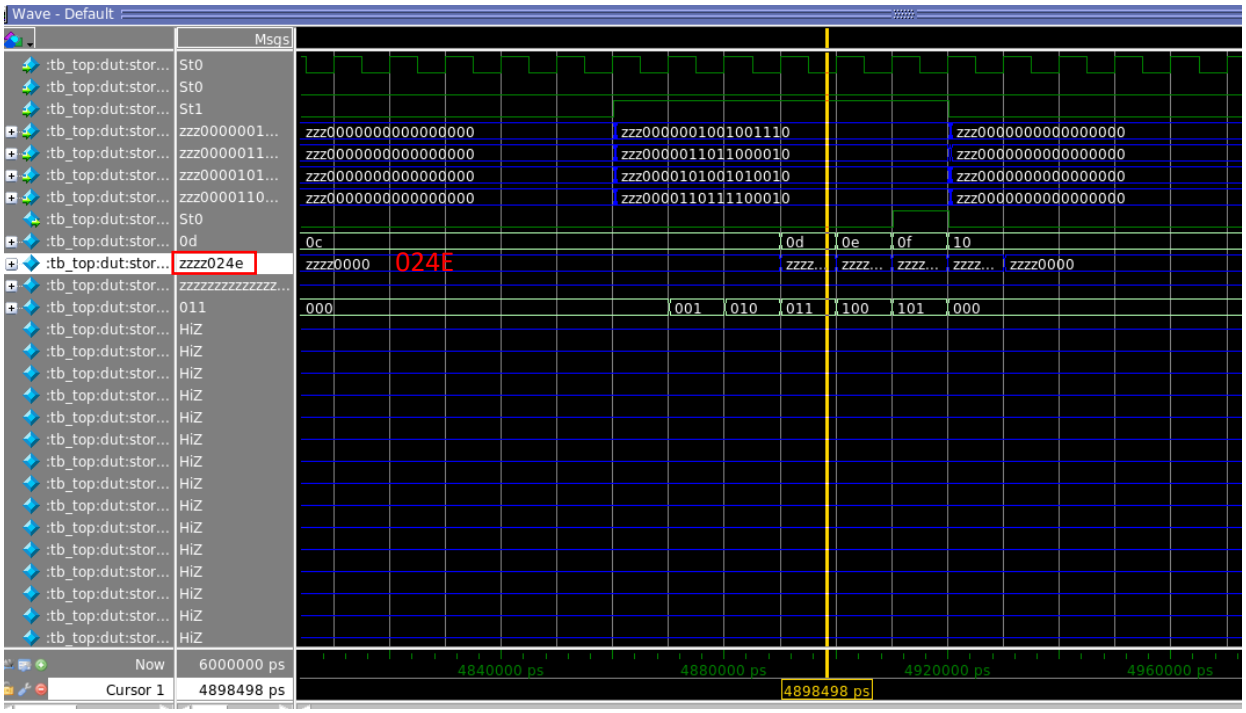

Figure 4.4: the thirteenth result 024E in hexadecimal can be stored in memory successfully

5. Place and route

In this section, the layout of matrix multiplier design needs to be used by Encounter and some data are needed.

LEF: Library Exchange Format

• header.lef

• standardCell.lef: Cell Library

• IO.lef: Pad Library

• memory.lef: custom

lib/tlf: libraries that contain timing information

sdc: Synopsys Design Constraint (generated during synthesis). Optional

Memory: memory.lib

Design (netlist): top.v

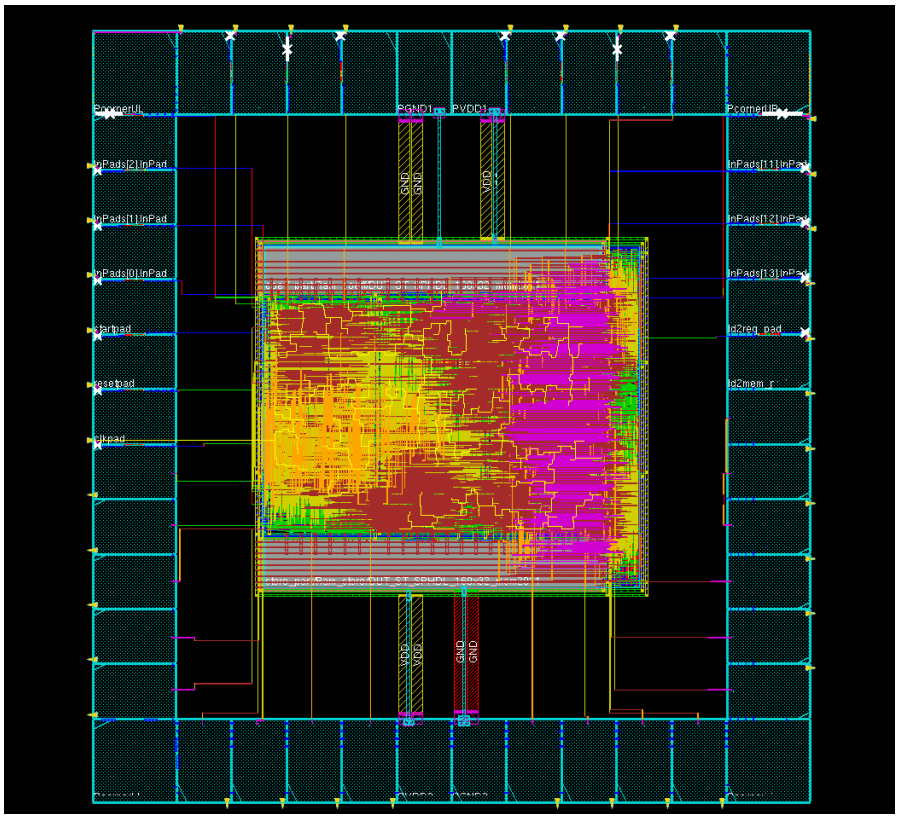

Figure 5: the layout of matrix multiplier

The scripts need to be added before the place and route. The layout of the matrix multiplier has been shown in figure 5. And the layout result has been shown in table 5.1. From table 5.1, the total area is 66405 μm^2 and the utilization of core is 94.327%.

Total Area (um^2)	66405
Core Utilization	94.327%

Table 5.1: area report after optimization

After the place and route, we need to optimize our design and fulfill the setup time and hold time requirements. Both values need to be positive which means the matrix multiplier can work successfully after placing the clock tree. As shown in Table 5.2, the setup time is 0.279ns and the hold time is 0.000ns. Both values fulfill the requirements.

Type	Time (ns)
Setup Time	0.279
Hold Time	0.000

Table 5.2: timing report after optimization

6. Appendix

6.1 Input matrix

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32

Table 6.1: input matrix

6.2 coefficient matrix

1	6	10	14
1	6	10	14
2	7	11	15
2	7	11	15
3	8	12	16
3	8	12	16
4	9	13	17
4	9	13	17

Table 6.2: coefficient matrix

6.3 result matrix

110	290	434	578
270	770	1170	1570
430	1250	1906	2526
590	1730	2642	3554

Table 6.3: result matrix

6.4 VHDL code for “top”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Top is
    port(
        clki        : in std_logic;
        reseti      : in std_logic;
```

```
        start      : in std_logic;
        datainput  : in std_logic_vector(13 downto 0);
        dataOutput : out std_logic_vector(18 downto 0);
        ld2mem     : out std_logic;
        ld2reg     : out std_logic
--        input_write_en_o : out std_logic;
--        rom_write_en_o   : out std_logic
       );
end Top;

architecture Behavioral of top is



 component CPAD_S_74x50u_IN            --input PAD
   port (
     COREIO : out std_logic;
     PADIO  : in  std_logic);
 end component;

 component CPAD_S_74x50u_OUT           --output PAD
   port (
     COREIO : in  std_logic;
     PADIO  : out std_logic);
 end component;

--   component SRAM_SP_WRAPPER
--     port (
--       ClkxCI  : in  std_logic;
--       CSxSI   : in  std_logic;           -- Active Low
--       WExSI   : in  std_logic;           --Active Low
--       AddrxDI : in  std_logic_vector (7 downto 0);
--       RYxSO   : out std_logic;
--       DataxDI : in  std_logic_vector (31 downto 0);
```

```
--        DataxDO : out std_logic_vector (31 downto 0)
--        );
--    end component;
-----------------------------------------------------------------------------------------------------------------
component controller is
    Port (
        clk, reset      : in std_logic;
        ldcoeff_done    : in std_logic;
        ldinput_done    : in std_logic;
        start           : in std_logic;
        load_done       : in std_logic;
        op_done         : in std_logic;
        store_done      : in std_logic;
    --ld coeff-----------------------------------------------------------
        ctrl_coeff      : in std_logic_vector(5 downto 0);
        coeff           : in std_logic_vector(6 downto 0);
    --ld input-----------------------------------------------------------
        ctrl_input      : in std_logic_vector(3 downto 0);
        input           : in std_logic_vector(7 downto 0);
    --number from op-----------------------------------------------------
        result1         : in std_logic_vector(18 downto 0);
        result2         : in std_logic_vector(18 downto 0);
        result3         : in std_logic_vector(18 downto 0);
        result4         : in std_logic_vector(18 downto 0);
-------------------------------------------------------------------------
        ldcoeff_enable  : out std_logic;
        ldinput_enable  : out std_logic;
        load_en         : out std_logic;
        op_en           : out std_logic;
        store_en        : out std_logic;
        max_en          : out std_logic;
        avg_en          : out std_logic;
    --ld coeff-----------------------------------------------------------
    --ld input-----------------------------------------------------------
```

```
      column_out        : out std_logic_vector(1 downto 0);
   --number to op-------------------------------------------------------
      data2op         : out std_logic_vector(7 downto 0);
      address2op      : out std_logic_vector(5 downto 0);
   ---------------------------------------------------------------
      result1_2store  : out std_logic_vector(18 downto 0);
      result2_2store  : out std_logic_vector(18 downto 0);
      result3_2store  : out std_logic_vector(18 downto 0);
      result4_2store  : out std_logic_vector(18 downto 0)
   );

end component;
-------------------------------------------------------------------------------------------------------------
component load_coeff is
  Port (
    clk, reset     : in std_logic;
   ldcoeff_enable  : in std_logic;
   coeff_read      : in std_logic_vector(13 downto 0);
   ld2mem          : out std_logic;
   --address         : out std_logic_vector(3 downto 0);
   --to controller-----------------------------------------------------
   ctrl_coeff      : out std_logic_vector(5 downto 0);
   coeff           : out std_logic_vector(6 downto 0);
   ldcoeff_done    : out std_logic
  );
end component;
-------------------------------------------------------------------------------------------------------------------
component load_input is
  Port (
      clk             : in std_logic;
      reset           : in std_logic;
      ldinput_en      : in std_logic;
      column          : in std_logic_vector(1 downto 0);
      load_en         : in std_logic;
```

```
        input_in        : in std_logic_vector(7 downto 0);
        ld2reg          : out std_logic;
        ctrl_input      : out std_logic_vector(3 downto 0);
        input           : out std_logic_vector(7 downto 0);
        ldinput_done    : out std_logic;
        load_done       : out std_logic
  );
end component;
-------------------------------------------------------------------------------------------------------------------
component operation is
  Port (
        clk, reset      : in std_logic;
        op_en           : in std_logic;
        data2op         : in std_logic_vector(7 downto 0);
        address2op      : in std_logic_vector(5 downto 0);
        op_done         : out std_logic;
        result1_out     : out std_logic_vector(18 downto 0);
        result2_out     : out std_logic_vector(18 downto 0);
        result3_out     : out std_logic_vector(18 downto 0);
        result4_out     : out std_logic_vector(18 downto 0);
        compare_out     : out std_logic_vector(18 downto 0)

  );
end component;
-------------------------------------------------------------------------------------------------------------------
------
component comparison is
  Port (
    clk, reset      : in std_logic;
    max_en          : in std_logic;
    --compare_done    : in std_logic;
    compare_out     : in std_logic_vector(18 downto 0);

    compare_result  : out std_logic_vector(18 downto 0)
```

```
  );
end component;
-------------------------------------------------------------------------------------------------------------------------
--------
component store_result is
  Port (
        clk, reset      : in std_logic;
        store_en        : in std_logic;
        result1_2store  : in std_logic_vector(18 downto 0);
        result2_2store  : in std_logic_vector(18 downto 0);
        result3_2store  : in std_logic_vector(18 downto 0);
        result4_2store  : in std_logic_vector(18 downto 0);
        store_done      : out std_logic
  );
end component;


--signal----------------------------------------------------------------------------------------------------------------
---
signal clk, reset      : std_logic;
signal starti          : std_logic;
signal dataInputi      : std_logic_vector(13 downto 0);
signal dataOutputi     : std_logic_vector(18 downto 0);
signal ld2mem_o        : std_logic;
signal ld2reg_o        : std_logic;
--signal load_en          : std_logic;
--signal coeff_in         : std_logic_vector(31 downto 0);
--signal ld2mem_o         : std_logic;
--signal address          : std_logic_vector(7 downto 0);
--signal ctrl_coeff       : std_logic_vector(5 downto 0);
--signal coeff            : std_logic_vector(6 downto 0);
--signal ldcoeff_done     : std_logic;

signal ldcoeff_done    : std_logic;
```

```
signal ldinput_done    : std_logic;
signal start_i         : std_logic;
signal load_done       : std_logic;
signal op_done         : std_logic;
signal store_done      : std_logic;
signal ctrl_coeff      : std_logic_vector(5 downto 0);
signal coeff           : std_logic_vector(6 downto 0);
signal ctrl_input      : std_logic_vector(3 downto 0);
signal input           : std_logic_vector(7 downto 0);
signal result1         : std_logic_vector(18 downto 0);
signal result2         : std_logic_vector(18 downto 0);
signal result3         : std_logic_vector(18 downto 0);
signal result4         : std_logic_vector(18 downto 0);

signal ldcoeff_enable  : std_logic;
signal ldinput_enable  : std_logic;
signal load_en         : std_logic;
signal op_en           : std_logic;
signal store_en        : std_logic;
signal max_en          : std_logic;
signal avg_en          : std_logic;
signal column_out      : std_logic_vector(1 downto 0);
signal data2op         : std_logic_vector(7 downto 0);
signal address2op      : std_logic_vector(5 downto 0);
signal result1_2store  : std_logic_vector(18 downto 0);
signal result2_2store  : std_logic_vector(18 downto 0);
signal result3_2store  : std_logic_vector(18 downto 0);
signal result4_2store  : std_logic_vector(18 downto 0);

signal compare_out     : std_logic_vector(18 downto 0);
signal compare_result  : std_logic_vector(18 downto 0);

signal CSN_rom         : std_logic;           -- Active Low
signal We              : std_logic;           --Active Low
```

```
signal addrxdi         : std_logic_vector (7 downto 0);
signal RY_rom          : std_logic;
signal data_in         : std_logic_vector (31 downto 0);
signal data_rom        : std_logic_vector (31 downto 0);

--signal coeff_read      : std_logic_vector(13 downto 0);

begin
--Rom:SRAM_SP_WRAPPER
--port map(
--    ClkxCI             => clki         ,
--    CSxSI              => CSN_rom      , -- Active Low
--    WExSI              => We           , -- Active Low
--    AddrxDI            => addrxdi      ,
--    RYxSO              => RY_rom       ,
--    DataxDI            => data_in      ,
--    DataxDO            => data_rom
--    );

controller_part: controller
port map(
        clk             => clki           ,
        reset           =>reseti          ,
        ldcoeff_done    => ldcoeff_done   ,
        ldinput_done    => ldinput_done   ,
        start           => start          ,
        load_done       => load_done      ,
        op_done         => op_done        ,
        store_done      => store_done     ,
        ctrl_coeff      => ctrl_coeff     ,
        coeff           => coeff          ,
        ctrl_input      => ctrl_input     ,
        input           => input          ,
        result1         => result1        ,
```

```
        result2         => result2          ,
        result3         => result3          ,
        result4         => result4          ,
        ldcoeff_enable  => ldcoeff_enable   ,
        ldinput_enable  => ldinput_enable   ,
        load_en         => load_en          ,
        op_en           => op_en            ,
        store_en        => store_en         ,
        max_en          => max_en           ,
        avg_en          => avg_en           ,
        column_out      => column_out       ,
        data2op         => data2op          ,
        address2op      => address2op       ,
        result1_2store  => result1_2store   ,
        result2_2store  => result2_2store   ,
        result3_2store  => result3_2store   ,
        result4_2store  => result4_2store

);

coeff_part: load_coeff
port map(
        clk             => clki             ,
        reset           => reseti           ,
        ldcoeff_enable  => ldcoeff_enable   ,
        coeff_read      => datainput        ,
        ld2mem          => ld2mem           ,
        ctrl_coeff      => ctrl_coeff       ,
        coeff           => coeff            ,
        ldcoeff_done    => ldcoeff_done

);

input_part: load_input
```

```
port map(

        clk             => clki             ,
        reset           => reseti           ,
        ldinput_en      => ldinput_enable   ,
        column          => column_out       ,
        load_en         => load_en          ,
        input_in        => datainput(7 downto 0),
        ld2reg          => ld2reg           ,
        ctrl_input      => ctrl_input       ,
        input           => input            ,
        ldinput_done    => ldinput_done     ,
        load_done       => load_done
  );


operation_part: operation
port map(
        clk             => clki             ,
        reset           => reseti           ,
        op_en           => op_en            ,
        data2op         => data2op          ,
        address2op      => address2op       ,
        op_done         => op_done          ,
        result1_out     => result1          ,
        result2_out     => result2          ,
        result3_out     => result3          ,
        result4_out     => result4          ,
        compare_out     => compare_out
);

store_part: store_result
port map(
        clk             => clki             ,
```

```
        reset          => reseti          ,
        store_en       => store_en        ,
        result1_2store  => result1_2store   ,
        result2_2store  => result2_2store   ,
        result3_2store  => result3_2store   ,
        result4_2store  => result4_2store   ,
        store_done     => store_done
);


comparison_part: comparison
port map(
        clk            => clki            ,
        reset          => reseti          ,
        max_en         => max_en          ,
        compare_out    => compare_out     ,
        compare_result  => compare_result
);

clkpad : CPAD_S_74x50u_IN
  port map (
    COREIO => clk,
    PADIO  => clki
    );
resetpad : CPAD_S_74x50u_IN
  port map (
    COREIO => reset,
    PADIO  => reseti
    );
 startpad : CPAD_S_74x50u_IN
  port map (
    COREIO => starti,
    PADIO  => start
    );
```

```
InPads : for i in 0 to 13 generate
  InPad : CPAD_S_74x50u_IN
    port map (
      COREIO => dataInputi(i),
      PADIO  => dataInput(i)
      );
end generate InPads;

OutPads : for i in 0 to 18 generate
  OutPad : CPAD_S_74x50u_OUT
    port map (
      COREIO => compare_result(i),
      PADIO  => dataOutput(i)
      );
end generate OutPads;
ld2mem_pad : CPAD_S_74x50u_OUT
 port map (
   COREIO => ld2mem_o,
   PADIO  => ld2mem
   );
ld2reg_pad : CPAD_S_74x50u_OUT
 port map (
   COREIO => ld2reg_o,
   PADIO  => ld2reg
   );




end Behavioral;
```

6.5 VHDL code for “controller”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;


entity controller is
    Port (
        clk, reset      : in std_logic;
        ldcoeff_done    : in std_logic;
        ldinput_done    : in std_logic;
        start           : in std_logic;
        load_done       : in std_logic;
        op_done         : in std_logic;
        store_done      : in std_logic;
    --ld coeff-----------------------------------------------------------
        ctrl_coeff      : in std_logic_vector(5 downto 0);
        coeff           : in std_logic_vector(6 downto 0);
    --ld input-----------------------------------------------------------
        ctrl_input      : in std_logic_vector(3 downto 0);
        input           : in std_logic_vector(7 downto 0);
    --number from op-----------------------------------------------------
        result1         : in std_logic_vector(18 downto 0);
        result2         : in std_logic_vector(18 downto 0);
        result3         : in std_logic_vector(18 downto 0);
        result4         : in std_logic_vector(18 downto 0);
-------------------------------------------------------------------------
        ldcoeff_enable  : out std_logic;
        ldinput_enable  : out std_logic;
        load_en         : out std_logic;
        op_en           : out std_logic;
        store_en        : out std_logic;
        max_en          : out std_logic;
        avg_en          : out std_logic;
    --ld coeff-----------------------------------------------------------
    --ld input-----------------------------------------------------------
```

```
        column_out        : out std_logic_vector(1 downto 0);
    --number to op-------------------------------------------------------
        data2op         : out std_logic_vector(7 downto 0);
        address2op      : out std_logic_vector(5 downto 0);
    ---------------------------------------------------------------
        result1_2store  : out std_logic_vector(18 downto 0);
        result2_2store  : out std_logic_vector(18 downto 0);
        result3_2store  : out std_logic_vector(18 downto 0);
        result4_2store  : out std_logic_vector(18 downto 0)
    );

end controller;


architecture Behavioral of controller is

    signal ldcoeff_controller   : std_logic;
    signal ldinput_controller   : std_logic;
    signal op_controller        : std_logic;

--state machine-----------------------------------------------------

    type state_type is (s_coeff2mem, s_input2reg, s_idle, s_load, s_op, s_store, s_max, s_avg);
    signal state_reg, state_nxt : state_type;

--load coeff-------------------------------------------------
    signal coeff_test   :std_logic_vector (6 downto 0);

    signal coeff01      : std_logic_vector(6 downto 0);
    signal coeff02      : std_logic_vector(6 downto 0);
    signal coeff03      : std_logic_vector(6 downto 0);
    signal coeff04      : std_logic_vector(6 downto 0);
    signal coeff05      : std_logic_vector(6 downto 0);
    signal coeff06      : std_logic_vector(6 downto 0);
```

```
signal coeff07     : std_logic_vector(6 downto 0);
signal coeff08     : std_logic_vector(6 downto 0);
signal coeff09     : std_logic_vector(6 downto 0);
signal coeff10     : std_logic_vector(6 downto 0);
signal coeff11     : std_logic_vector(6 downto 0);
signal coeff12     : std_logic_vector(6 downto 0);
signal coeff13     : std_logic_vector(6 downto 0);
signal coeff14     : std_logic_vector(6 downto 0);
signal coeff15     : std_logic_vector(6 downto 0);
signal coeff16     : std_logic_vector(6 downto 0);
signal coeff17     : std_logic_vector(6 downto 0);
signal coeff18     : std_logic_vector(6 downto 0);
signal coeff19     : std_logic_vector(6 downto 0);
signal coeff20     : std_logic_vector(6 downto 0);
signal coeff21     : std_logic_vector(6 downto 0);
signal coeff22     : std_logic_vector(6 downto 0);
signal coeff23     : std_logic_vector(6 downto 0);
signal coeff24     : std_logic_vector(6 downto 0);
signal coeff25     : std_logic_vector(6 downto 0);
signal coeff26     : std_logic_vector(6 downto 0);
signal coeff27     : std_logic_vector(6 downto 0);
signal coeff28     : std_logic_vector(6 downto 0);
signal coeff29     : std_logic_vector(6 downto 0);
signal coeff30     : std_logic_vector(6 downto 0);
signal coeff31     : std_logic_vector(6 downto 0);
signal coeff32     : std_logic_vector(6 downto 0);

signal coeff01_nxt  : std_logic_vector(6 downto 0);
signal coeff02_nxt  : std_logic_vector(6 downto 0);
signal coeff03_nxt  : std_logic_vector(6 downto 0);
signal coeff04_nxt  : std_logic_vector(6 downto 0);
signal coeff05_nxt  : std_logic_vector(6 downto 0);
signal coeff06_nxt  : std_logic_vector(6 downto 0);
signal coeff07_nxt  : std_logic_vector(6 downto 0);
```

```
    signal coeff08_nxt  : std_logic_vector(6 downto 0);
    signal coeff09_nxt  : std_logic_vector(6 downto 0);
    signal coeff10_nxt  : std_logic_vector(6 downto 0);
    signal coeff11_nxt  : std_logic_vector(6 downto 0);
    signal coeff12_nxt  : std_logic_vector(6 downto 0);
    signal coeff13_nxt  : std_logic_vector(6 downto 0);
    signal coeff14_nxt  : std_logic_vector(6 downto 0);
    signal coeff15_nxt  : std_logic_vector(6 downto 0);
    signal coeff16_nxt  : std_logic_vector(6 downto 0);
    signal coeff17_nxt  : std_logic_vector(6 downto 0);
    signal coeff18_nxt  : std_logic_vector(6 downto 0);
    signal coeff19_nxt  : std_logic_vector(6 downto 0);
    signal coeff20_nxt  : std_logic_vector(6 downto 0);
    signal coeff21_nxt  : std_logic_vector(6 downto 0);
    signal coeff22_nxt  : std_logic_vector(6 downto 0);
    signal coeff23_nxt  : std_logic_vector(6 downto 0);
    signal coeff24_nxt  : std_logic_vector(6 downto 0);
    signal coeff25_nxt  : std_logic_vector(6 downto 0);
    signal coeff26_nxt  : std_logic_vector(6 downto 0);
    signal coeff27_nxt  : std_logic_vector(6 downto 0);
    signal coeff28_nxt  : std_logic_vector(6 downto 0);
    signal coeff29_nxt  : std_logic_vector(6 downto 0);
    signal coeff30_nxt  : std_logic_vector(6 downto 0);
    signal coeff31_nxt  : std_logic_vector(6 downto 0);
    signal coeff32_nxt  : std_logic_vector(6 downto 0);

--ld input-----------------------------------------------------------
    signal input_test   : std_logic_vector(7 downto 0);

    signal input01      : std_logic_vector(7 downto 0);
    signal input02      : std_logic_vector(7 downto 0);
    signal input03      : std_logic_vector(7 downto 0);
    signal input04      : std_logic_vector(7 downto 0);
    signal input05      : std_logic_vector(7 downto 0);
```

```
    signal input06      : std_logic_vector(7 downto 0);
    signal input07      : std_logic_vector(7 downto 0);
    signal input08      : std_logic_vector(7 downto 0);

    signal input01_nxt  : std_logic_vector(7 downto 0);
    signal input02_nxt  : std_logic_vector(7 downto 0);
    signal input03_nxt  : std_logic_vector(7 downto 0);
    signal input04_nxt  : std_logic_vector(7 downto 0);
    signal input05_nxt  : std_logic_vector(7 downto 0);
    signal input06_nxt  : std_logic_vector(7 downto 0);
    signal input07_nxt  : std_logic_vector(7 downto 0);
    signal input08_nxt  : std_logic_vector(7 downto 0);

--op counter-----------------------------------------------------------
    signal counter, counter_nxt   : std_logic_vector(5 downto 0);
    signal start_count            : std_logic;
--store data-----------------------------------------------------------
    signal column       : std_logic_vector(1 downto 0);
    signal column_nxt   : std_logic_vector(1 downto 0);

    --signal column_ctrl  : std_logic;
    signal result1_reg  : std_logic_vector(18 downto 0);
    signal result2_reg  : std_logic_vector(18 downto 0);
    signal result3_reg  : std_logic_vector(18 downto 0);
    signal result4_reg  : std_logic_vector(18 downto 0);

    signal mean1    : std_logic_vector(18 downto 0);
    signal mean2    : std_logic_vector(18 downto 0);
    signal mean3    : std_logic_vector(18 downto 0);
    signal mean4    : std_logic_vector(18 downto 0);
--mean-----------------------------------------------------------------
    signal mean_reg : std_logic_vector(20 downto 0);
    signal mean_out : std_logic_vector(20 downto 0);
-------------------------------------------------------------------------
```

```
begin
--state ctrl----------------------------------
process (clk, reset)
begin
   if reset = '1' then
      state_reg <= s_coeff2mem;
      column <= (others => '0');
   elsif (clk'event and clk = '1') then
      state_reg <= state_nxt;
      column <= column_nxt;
   end if;
end process;

column_out <= column;

--state machine-------------------------------------
state_machine:process (state_reg, start,
      ldcoeff_done, ldinput_done, load_done, op_done, store_done,
      column, column_nxt,
      result1_reg, result2_reg, result3_reg, result4_reg,
      mean1, mean2, mean3, mean4, mean_reg
      )
begin
   ldcoeff_enable <= '0';
   ldinput_enable <= '0';
   load_en <= '0';
   op_en <= '0';
   store_en <= '0';
   max_en <= '0';
   avg_en <= '0';
   result1_2store <= (others => '0');
   result2_2store <= (others => '0');
   result3_2store <= (others => '0');
```

```
result4_2store <= (others => '0');
ldcoeff_controller <= '1';
ldinput_controller <= '1';
op_controller <= '0';
result1_reg <= (others => '0');
result2_reg <= (others => '0');
result3_reg <= (others => '0');
result4_reg <= (others => '0');

column_nxt <= column;

case state_reg is
   when s_coeff2mem =>
      ldinput_enable <= '0';
      column_nxt <= "00";
      if ldcoeff_done = '1' then
         state_nxt <= s_input2reg;
      else
         ldcoeff_enable <= '1';
         ldcoeff_controller <= '1';
         state_nxt <= s_coeff2mem;
      end if;

   when s_input2reg =>
      ldcoeff_enable <= '0';
      ldcoeff_controller <= '0';
      ldinput_enable <= '1';
      if ldinput_done = '1' then
         state_nxt <= s_idle;
      else
         state_nxt <= s_input2reg;
      end if;

   when s_idle =>
```

```
    load_en <= '0';
    ldinput_controller <= '0';
    op_en <= '0';
    store_en <= '0';
    max_en <= '0';
    avg_en <= '0';
    if start = '1' then
        state_nxt <= s_load;
    else
        state_nxt <= state_reg;
    end if;

when s_load =>
    load_en <= '1';
    ldinput_controller <= '1';
    op_en <= '0';
    op_controller <= '0';
    store_en <= '0';
    max_en <= '0';
    avg_en <= '0';
    if load_done = '1' then
        state_nxt <= s_op;
    else
        state_nxt <= s_load;
    end if;

when s_op =>
    load_en <= '0';
    ldinput_controller <= '0';
    op_en <= '1';
    op_controller <= '1';
    store_en <= '0';
    max_en <= '0';
    avg_en <= '0';
```

```
    if op_done = '1' then
        state_nxt <= s_store;
    else
        state_nxt <= s_op;
    end if;

when s_store =>
    load_en <= '0';
    op_en <= '0';
    op_controller <= '0';
    store_en <= '1';
    max_en <= '0';
    avg_en <= '0';
    result1_reg <= result1;
    result2_reg <= result2;
    result3_reg <= result3;
    result4_reg <= result4;

        case column is
            when "00" =>
                mean1 <= result1_reg;
                result1_2store <= result1_reg;
                result2_2store <= result2_reg;
                result3_2store <= result3_reg;
                result4_2store <= result4_reg;
                if store_done = '1' then
                    state_nxt <= s_load;
                    column_nxt <= column + 1;
                else
                    state_nxt <= s_store;
                end if;

            when "01" =>
                mean2 <= result2_reg;
```

```
        result1_2store <= result1_reg;
        result2_2store <= result2_reg;
        result3_2store <= result3_reg;
        result4_2store <= result4_reg;
        if store_done = '1' then
            state_nxt <= s_load;
            column_nxt <= column + 1;
        else
            state_nxt <= s_store;
        end if;
    when "10" =>
        mean3 <= result3_reg;
        result1_2store <= result1_reg;
        result2_2store <= result2_reg;
        result3_2store <= result3_reg;
        result4_2store <= result4_reg;
        if store_done = '1' then
            state_nxt <= s_load;
            column_nxt <= column + 1;
        else
            state_nxt <= s_store;
        end if;
    when "11" =>
        mean4 <= result4_reg;
        result1_2store <= result1_reg;
        result2_2store <= result2_reg;
        result3_2store <= result3_reg;
        result4_2store <= result4_reg;
        if store_done = '1' then
            state_nxt <= s_max;
            column_nxt <= "00";
        else
            state_nxt <= s_store;
        end if;
```

```
                when others => state_nxt <= s_store;
            end case;



        when s_max =>
            load_en <= '0';
            op_en <= '0';
            store_en <= '0';
            avg_en <= '0';
            max_en <= '1';
            state_nxt <= s_avg;

        when s_avg =>
            load_en <= '0';
            op_en <= '0';
            store_en <= '0';
            max_en <= '0';
            avg_en <= '1';
            mean_reg <= mean1 + mean2 + mean3 + mean4 + std_logic_vector(to_unsigned(0,21));
            mean_out <= "00" & mean_reg(20 downto 2);
            state_nxt <= s_input2reg;
    end case;

end process;

process(clk, reset)
begin
if reset = '1' then
        coeff01 <= (others => '0');
        coeff02 <= (others => '0');
        coeff03 <= (others => '0');
        coeff04 <= (others => '0');
```

```
coeff05 <= (others => '0');
coeff06 <= (others => '0');
coeff07 <= (others => '0');
coeff08 <= (others => '0');
coeff09 <= (others => '0');
coeff10 <= (others => '0');
coeff11 <= (others => '0');
coeff12 <= (others => '0');
coeff13 <= (others => '0');
coeff14 <= (others => '0');
coeff15 <= (others => '0');
coeff16 <= (others => '0');
coeff17 <= (others => '0');
coeff18 <= (others => '0');
coeff19 <= (others => '0');
coeff20 <= (others => '0');
coeff21 <= (others => '0');
coeff22 <= (others => '0');
coeff23 <= (others => '0');
coeff24 <= (others => '0');
coeff25 <= (others => '0');
coeff26 <= (others => '0');
coeff27 <= (others => '0');
coeff28 <= (others => '0');
coeff29 <= (others => '0');
coeff30 <= (others => '0');
coeff31 <= (others => '0');
coeff32 <= (others => '0');
input01 <= (others => '0');
input02 <= (others => '0');
input03 <= (others => '0');
input04 <= (others => '0');
input05 <= (others => '0');
input06 <= (others => '0');
```

```
    input07 <= (others => '0');
    input08 <= (others => '0');

elsif (clk'event and clk = '1') then
    coeff01 <= coeff01_nxt;
    coeff02 <= coeff02_nxt;
    coeff03 <= coeff03_nxt;
    coeff04 <= coeff04_nxt;
    coeff05 <= coeff05_nxt;
    coeff06 <= coeff06_nxt;
    coeff07 <= coeff07_nxt;
    coeff08 <= coeff08_nxt;
    coeff09 <= coeff09_nxt;
    coeff10 <= coeff10_nxt;
    coeff11 <= coeff11_nxt;
    coeff12 <= coeff12_nxt;
    coeff13 <= coeff13_nxt;
    coeff14 <= coeff14_nxt;
    coeff15 <= coeff15_nxt;
    coeff16 <= coeff16_nxt;
    coeff17 <= coeff17_nxt;
    coeff18 <= coeff18_nxt;
    coeff19 <= coeff19_nxt;
    coeff20 <= coeff20_nxt;
    coeff21 <= coeff21_nxt;
    coeff22 <= coeff22_nxt;
    coeff23 <= coeff23_nxt;
    coeff24 <= coeff24_nxt;
    coeff25 <= coeff25_nxt;
    coeff26 <= coeff26_nxt;
    coeff27 <= coeff27_nxt;
    coeff28 <= coeff28_nxt;
    coeff29 <= coeff29_nxt;
    coeff30 <= coeff30_nxt;
```

```
        coeff31 <= coeff31_nxt;
        coeff32 <= coeff32_nxt;
        input01 <= input01_nxt;
        input02 <= input02_nxt;
        input03 <= input03_nxt;
        input04 <= input04_nxt;
        input05 <= input05_nxt;
        input06 <= input06_nxt;
        input07 <= input07_nxt;
        input08 <= input08_nxt;

    end if;

end process;


--load coeff-----------------------------------------------------
ld_coeff: process(ldcoeff_controller, ctrl_coeff, coeff)
begin
    coeff01_nxt <= coeff01;
    coeff02_nxt <= coeff02;
    coeff03_nxt <= coeff03;
    coeff04_nxt <= coeff04;
    coeff05_nxt <= coeff05;
    coeff06_nxt <= coeff06;
    coeff07_nxt <= coeff07;
    coeff08_nxt <= coeff08;
    coeff09_nxt <= coeff09;
    coeff10_nxt <= coeff10;
    coeff11_nxt <= coeff11;
    coeff12_nxt <= coeff12;
    coeff13_nxt <= coeff13;
    coeff14_nxt <= coeff14;
    coeff15_nxt <= coeff15;
```

```
coeff16_nxt <= coeff16;
coeff17_nxt <= coeff17;
coeff18_nxt <= coeff18;
coeff19_nxt <= coeff19;
coeff20_nxt <= coeff20;
coeff21_nxt <= coeff21;
coeff22_nxt <= coeff22;
coeff23_nxt <= coeff23;
coeff24_nxt <= coeff24;
coeff25_nxt <= coeff25;
coeff26_nxt <= coeff26;
coeff27_nxt <= coeff27;
coeff28_nxt <= coeff28;
coeff29_nxt <= coeff29;
coeff30_nxt <= coeff30;
coeff31_nxt <= coeff31;
coeff32_nxt <= coeff32;

if ldcoeff_controller = '1' then
    case ctrl_coeff is
        when "000001" => coeff01_nxt <= coeff;
        when "000010" => coeff02_nxt <= coeff;
        when "000011" => coeff03_nxt <= coeff;
        when "000100" => coeff04_nxt <= coeff;
        when "000101" => coeff05_nxt <= coeff;
        when "000110" => coeff06_nxt <= coeff;
        when "000111" => coeff07_nxt <= coeff;
        when "001000" => coeff08_nxt <= coeff;
        when "001001" => coeff09_nxt <= coeff;
        when "001010" => coeff10_nxt <= coeff;
        when "001011" => coeff11_nxt <= coeff;
        when "001100" => coeff12_nxt <= coeff;
        when "001101" => coeff13_nxt <= coeff;
        when "001110" => coeff14_nxt <= coeff;
```

```
            when "001111" => coeff15_nxt <= coeff;
            when "010000" => coeff16_nxt <= coeff;
            when "010001" => coeff17_nxt <= coeff;
            when "010010" => coeff18_nxt <= coeff;
            when "010011" => coeff19_nxt <= coeff;
            when "010100" => coeff20_nxt <= coeff;
            when "010101" => coeff21_nxt <= coeff;
            when "010110" => coeff22_nxt <= coeff;
            when "010111" => coeff23_nxt <= coeff;
            when "011000" => coeff24_nxt <= coeff;
            when "011001" => coeff25_nxt <= coeff;
            when "011010" => coeff26_nxt <= coeff;
            when "011011" => coeff27_nxt <= coeff;
            when "011100" => coeff28_nxt <= coeff;
            when "011101" => coeff29_nxt <= coeff;
            when "011110" => coeff30_nxt <= coeff;
            when "011111" => coeff31_nxt <= coeff;
            when "100000" => coeff32_nxt <= coeff;
            when others => coeff_test <= (others => '0');
        end case;
    else
        coeff_test <= (others => '0');
    end if;

end process;



--load input----------------------------------------------------
ld_input: process(ldinput_controller, ctrl_input, input)
begin
    input01_nxt <= input01;
    input02_nxt <= input02;
    input03_nxt <= input03;
    input04_nxt <= input04;
```

```
    input05_nxt <= input05;
    input06_nxt <= input06;
    input07_nxt <= input07;
    input08_nxt <= input08;

    if ldinput_controller = '1' then
        case ctrl_input is
            when "0001" => input01_nxt <= input;
            when "0010" => input02_nxt <= input;
            when "0011" => input03_nxt <= input;
            when "0100" => input04_nxt <= input;
            when "0101" => input05_nxt <= input;
            when "0110" => input06_nxt <= input;
            when "0111" => input07_nxt <= input;
            when "1000" => input08_nxt <= input;
            when others => input_test <= (others =>'0');
        end case;
    else
        input_test <= (others => '0');
    end if;

end process;

--op send data-----------------------------------------------------------------
op_counter : process(clk, reset, start_count, counter_nxt)
begin
    if reset = '1' then
        counter <= "000001";
    elsif (clk'event and clk = '1') then
        if start_count = '1' then
            counter <= counter_nxt;
        else
            counter <= "000001";
        end if;
```

```
    end if;

end process;

op_send: process(op_controller, counter,
                coeff01, coeff02, coeff03, coeff04, coeff05, coeff06, coeff07, coeff08, coeff09, coeff10,
                coeff11, coeff12, coeff13, coeff14,coeff15, coeff16, coeff17, coeff18, coeff19, coeff20,
                coeff21, coeff22, coeff23, coeff24, coeff25, coeff26, coeff27, coeff28, coeff29, coeff30,
                coeff31, coeff32,
                input01, input02, input03, input04, input05, input06, input07, input08)
begin
    address2op <= "000001";
    data2op <= (others => '0');
    counter_nxt <= "000001";
    if op_controller ='1' then
        start_count <= '1'; --contrl the op counter
            case counter is
                when "000001" => data2op <= "0" & coeff01; counter_nxt <= counter + "000001";
address2op <= "000001";
                when "000010" => data2op <= "0" & coeff02; counter_nxt <= counter + "000001";
address2op <= "000010";
                when "000011" => data2op <= "0" & coeff03; counter_nxt <= counter + "000001";
address2op <= "000011";
                when "000100" => data2op <= "0" & coeff04; counter_nxt <= counter + "000001";
address2op <= "000100";
                when "000101" => data2op <= "0" & coeff05; counter_nxt <= counter + "000001";
address2op <= "000101";
                when "000110" => data2op <= "0" & coeff06; counter_nxt <= counter + "000001";
address2op <= "000110";
                when "000111" => data2op <= "0" & coeff07; counter_nxt <= counter + "000001";
address2op <= "000111";
                when "001000" => data2op <= "0" & coeff08; counter_nxt <= counter + "000001";
address2op <= "001000";
                when "001001" => data2op <= "0" & coeff09; counter_nxt <= counter + "000001";
address2op <= "001001";
                when "001010" => data2op <= "0" & coeff10; counter_nxt <= counter + "000001";
address2op <= "001010";
```

```
            when "001011" => data2op <= "0" & coeff11; counter_nxt <= counter + "000001";
address2op <= "001011";

            when "001100" => data2op <= "0" & coeff12; counter_nxt <= counter + "000001";
address2op <= "001100";

            when "001101" => data2op <= "0" & coeff13; counter_nxt <= counter + "000001";
address2op <= "001101";

            when "001110" => data2op <= "0" & coeff14; counter_nxt <= counter + "000001";
address2op <= "001110";

            when "001111" => data2op <= "0" & coeff15; counter_nxt <= counter + "000001";
address2op <= "001111";

            when "010000" => data2op <= "0" & coeff16; counter_nxt <= counter + "000001";
address2op <= "010000";

            when "010001" => data2op <= "0" & coeff17; counter_nxt <= counter + "000001";
address2op <= "010001";

            when "010010" => data2op <= "0" & coeff18; counter_nxt <= counter + "000001";
address2op <= "010010";

            when "010011" => data2op <= "0" & coeff19; counter_nxt <= counter + "000001";
address2op <= "010011";

            when "010100" => data2op <= "0" & coeff20; counter_nxt <= counter + "000001";
address2op <= "010100";

            when "010101" => data2op <= "0" & coeff21; counter_nxt <= counter + "000001";
address2op <= "010101";

            when "010110" => data2op <= "0" & coeff22; counter_nxt <= counter + "000001";
address2op <= "010110";

            when "010111" => data2op <= "0" & coeff23; counter_nxt <= counter + "000001";
address2op <= "010111";

            when "011000" => data2op <= "0" & coeff24; counter_nxt <= counter + "000001";
address2op <= "011000";

            when "011001" => data2op <= "0" & coeff25; counter_nxt <= counter + "000001";
address2op <= "011001";

            when "011010" => data2op <= "0" & coeff26; counter_nxt <= counter + "000001";
address2op <= "011010";

            when "011011" => data2op <= "0" & coeff27; counter_nxt <= counter + "000001";
address2op <= "011011";

            when "011100" => data2op <= "0" & coeff28; counter_nxt <= counter + "000001";
address2op <= "011100";

            when "011101" => data2op <= "0" & coeff29; counter_nxt <= counter + "000001";
address2op <= "011101";

            when "011110" => data2op <= "0" & coeff30; counter_nxt <= counter + "000001";
address2op <= "011110";

            when "011111" => data2op <= "0" & coeff31; counter_nxt <= counter + "000001";
address2op <= "011111";
```

```
            when "100000" => data2op <= "0" & coeff32; counter_nxt <= counter + "000001";
address2op <= "100000";


            when "100001" => data2op <= input01; counter_nxt <= counter + "000001"; address2op <=
"100001";
            when "100010" => data2op <= input02; counter_nxt <= counter + "000001"; address2op <=
"100010";
            when "100011" => data2op <= input03; counter_nxt <= counter + "000001"; address2op <=
"100011";
            when "100100" => data2op <= input04; counter_nxt <= counter + "000001"; address2op <=
"100100";
            when "100101" => data2op <= input05; counter_nxt <= counter + "000001"; address2op <=
"100101";
            when "100110" => data2op <= input06; counter_nxt <= counter + "000001"; address2op <=
"100110";
            when "100111" => data2op <= input07; counter_nxt <= counter + "000001"; address2op <=
"100111";
            when "101000" => data2op <= input08; counter_nxt <= "000001"; address2op <= "101000";
            when others => data2op <= (others => '0'); counter_nxt <= "000001"; address2op <=
"000001";
         end case;
   else
      start_count <= '0';
   end if;


end process;


end Behavioral;
```

6.6 VHDL code for “load_coeff”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use std.textio.all;
```

```
entity load_coeff is
  Port (
     clk, reset     : in std_logic;

    ldcoeff_enable  : in std_logic;
    --coeff from memory-----------------------------------------------------
    coeff_read      : in std_logic_vector(13 downto 0);

    --enable txt file to memory-------------------------------------------
    ld2mem          : out std_logic;
    --to controller-------------------------------------------------------
    ctrl_coeff      : out std_logic_vector(5 downto 0);
    coeff           : out std_logic_vector(6 downto 0);
    ldcoeff_done    : out std_logic

  );
end load_coeff;

architecture Behavioral of load_coeff is

component SRAM_SP_WRAPPER
  port (
    ClkxCI  : in  std_logic;
    CSxSI   : in  std_logic;            -- Active Low
    WExSI   : in  std_logic;            --Active Low
    AddrxDI : in  std_logic_vector (7 downto 0);
    RYxSO   : out std_logic;
    DataxDI : in  std_logic_vector (31 downto 0);
    DataxDO : out std_logic_vector (31 downto 0)
    );
end component;

component ff is
  generic(N:integer:=1);
```

```
  port(   D  :  in std_logic_vector(N-1 downto 0);
          Q  : out std_logic_vector(N-1 downto 0);
        clk  :  in std_logic;
        reset:  in std_logic
      );
end component;

    type state_type is (s_initial, s_load, s_keep, s_send);
    signal state_reg, state_nxt : state_type;

    signal counter      : std_logic_vector(4 downto 0) := (others => '0');
    signal counter_nxt  : std_logic_vector(4 downto 0) := (others => '0');
    signal counter_2    : std_logic_vector(4 downto 0) := (others => '0');
    signal counter_2_nxt: std_logic_vector(4 downto 0) := (others => '0');

    signal send_ctr     : std_logic_vector(4 downto 0) := (others => '0');
    signal send_ctr_nxt : std_logic_vector(4 downto 0) := (others => '0');

    signal coeff01      : std_logic_vector(6 downto 0);
    signal coeff02      : std_logic_vector(6 downto 0);
    signal coeff03      : std_logic_vector(6 downto 0);
    signal coeff04      : std_logic_vector(6 downto 0);
    signal coeff05      : std_logic_vector(6 downto 0);
    signal coeff06      : std_logic_vector(6 downto 0);
    signal coeff07      : std_logic_vector(6 downto 0);
    signal coeff08      : std_logic_vector(6 downto 0);
    signal coeff09      : std_logic_vector(6 downto 0);
    signal coeff10      : std_logic_vector(6 downto 0);
    signal coeff11      : std_logic_vector(6 downto 0);
    signal coeff12      : std_logic_vector(6 downto 0);
    signal coeff13      : std_logic_vector(6 downto 0);
    signal coeff14      : std_logic_vector(6 downto 0);
    signal coeff15      : std_logic_vector(6 downto 0);
    signal coeff16      : std_logic_vector(6 downto 0);
```

```
signal coeff17      : std_logic_vector(6 downto 0);
signal coeff18      : std_logic_vector(6 downto 0);
signal coeff19      : std_logic_vector(6 downto 0);
signal coeff20      : std_logic_vector(6 downto 0);
signal coeff21      : std_logic_vector(6 downto 0);
signal coeff22      : std_logic_vector(6 downto 0);
signal coeff23      : std_logic_vector(6 downto 0);
signal coeff24      : std_logic_vector(6 downto 0);
signal coeff25      : std_logic_vector(6 downto 0);
signal coeff26      : std_logic_vector(6 downto 0);
signal coeff27      : std_logic_vector(6 downto 0);
signal coeff28      : std_logic_vector(6 downto 0);
signal coeff29      : std_logic_vector(6 downto 0);
signal coeff30      : std_logic_vector(6 downto 0);
signal coeff31      : std_logic_vector(6 downto 0);
signal coeff32      : std_logic_vector(6 downto 0);

signal coeff01_nxt  : std_logic_vector(6 downto 0);
signal coeff02_nxt  : std_logic_vector(6 downto 0);
signal coeff03_nxt  : std_logic_vector(6 downto 0);
signal coeff04_nxt  : std_logic_vector(6 downto 0);
signal coeff05_nxt  : std_logic_vector(6 downto 0);
signal coeff06_nxt  : std_logic_vector(6 downto 0);
signal coeff07_nxt  : std_logic_vector(6 downto 0);
signal coeff08_nxt  : std_logic_vector(6 downto 0);
signal coeff09_nxt  : std_logic_vector(6 downto 0);
signal coeff10_nxt  : std_logic_vector(6 downto 0);
signal coeff11_nxt  : std_logic_vector(6 downto 0);
signal coeff12_nxt  : std_logic_vector(6 downto 0);
signal coeff13_nxt  : std_logic_vector(6 downto 0);
signal coeff14_nxt  : std_logic_vector(6 downto 0);
signal coeff15_nxt  : std_logic_vector(6 downto 0);
signal coeff16_nxt  : std_logic_vector(6 downto 0);
signal coeff17_nxt  : std_logic_vector(6 downto 0);
```

```
    signal coeff18_nxt  : std_logic_vector(6 downto 0);
    signal coeff19_nxt  : std_logic_vector(6 downto 0);
    signal coeff20_nxt  : std_logic_vector(6 downto 0);
    signal coeff21_nxt  : std_logic_vector(6 downto 0);
    signal coeff22_nxt  : std_logic_vector(6 downto 0);
    signal coeff23_nxt  : std_logic_vector(6 downto 0);
    signal coeff24_nxt  : std_logic_vector(6 downto 0);
    signal coeff25_nxt  : std_logic_vector(6 downto 0);
    signal coeff26_nxt  : std_logic_vector(6 downto 0);
    signal coeff27_nxt  : std_logic_vector(6 downto 0);
    signal coeff28_nxt  : std_logic_vector(6 downto 0);
    signal coeff29_nxt  : std_logic_vector(6 downto 0);
    signal coeff30_nxt  : std_logic_vector(6 downto 0);
    signal coeff31_nxt  : std_logic_vector(6 downto 0);
    signal coeff32_nxt  : std_logic_vector(6 downto 0);
    signal start_store  : std_logic;

    signal coeff_in     : std_logic_vector(31 downto 0);
    signal choose       : std_logic;
    signal address      : std_logic_vector(7 downto 0);
    signal address_out  : std_logic_vector(7 downto 0);
    signal RY_ram       : std_logic;
    signal dataxdi      : std_logic_vector(31 downto 0);


begin
Ram_coeff: SRAM_SP_WRAPPER
port map(
    ClkxCI          => clk          ,
    CSxSI           => '0'       , -- Active Low --only write in this module
    WExSI           => choose          , -- Active Low
    AddrxDI         => address      ,
    RYxSO           => RY_ram       ,
    DataxDI         => dataxdi      ,
```

```
    DataxDO          => coeff_in
    );

--state contrl--------------------------------
process(clk, reset)
begin
    if reset = '1' then
        state_reg <= s_initial;
        send_ctr <= (others => '0');
    elsif (clk'event and clk = '1') then
        state_reg <= state_nxt;
        send_ctr <= send_ctr_nxt;
    end if;

end process;


--state machine-----------------------------------------------
process(state_reg, counter, ldcoeff_enable, send_ctr, counter_2)
begin
    choose <= '1';
    dataxdi <= (others => '0');
    address <= (others => '0');
    ld2mem <= '0';
    ctrl_coeff <= (others => '0');
    coeff <= (others => '0');
    ldcoeff_done <= '0';
    counter_nxt <= (others => '0');
    counter_2_nxt <= (others => '0');

    send_ctr_nxt <= send_ctr;


    address <= (others => '0');
```

```
start_store <= '0';

case state_reg is
    when s_initial =>
        ctrl_coeff <= (others => '0');
        coeff <= (others => '0');
        ldcoeff_done <= '0';
        ld2mem <= '0';
        counter_nxt <= (others => '0');
        send_ctr_nxt <= (others => '0');

        if ldcoeff_enable = '1' then
            state_nxt <= s_load;
        else
            state_nxt <= s_initial;
        end if;

    when s_load => --load coefficients from txt to memory
        counter_nxt <= counter + 1;
        choose <= '0';--write
        address <= "000" & counter;
        dataxdi <= "0000000000000000" & coeff_read;
        if counter > 15 then
            state_nxt <= s_keep;
            ld2mem <= '0';
        else
            state_nxt <= s_load;
            ld2mem <= '1';
        end if;

    when s_keep =>
    choose <= '1';--read
    address <= "000" & counter_2;
```

```
counter_2_nxt <= counter_2 + 1;
if counter_2 > 15 then
   state_nxt <= s_send;
else
   state_nxt <= s_keep;
end if;

when s_send =>
   case send_ctr is
      when "00000" => ctrl_coeff <= "000001"; coeff <= coeff01; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "00001" => ctrl_coeff <= "000010"; coeff <= coeff02; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "00010" => ctrl_coeff <= "000011"; coeff <= coeff03; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "00011" => ctrl_coeff <= "000100"; coeff <= coeff04; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "00100" => ctrl_coeff <= "000101"; coeff <= coeff05; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "00101" => ctrl_coeff <= "000110"; coeff <= coeff06; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "00110" => ctrl_coeff <= "000111"; coeff <= coeff07; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "00111" => ctrl_coeff <= "001000"; coeff <= coeff08; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "01000" => ctrl_coeff <= "001001"; coeff <= coeff09; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "01001" => ctrl_coeff <= "001010"; coeff <= coeff10; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "01010" => ctrl_coeff <= "001011"; coeff <= coeff11; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "01011" => ctrl_coeff <= "001100"; coeff <= coeff12; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "01100" => ctrl_coeff <= "001101"; coeff <= coeff13; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "01101" => ctrl_coeff <= "001110"; coeff <= coeff14; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
      when "01110" => ctrl_coeff <= "001111"; coeff <= coeff15; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;
```

```
            when "01111" => ctrl_coeff <= "010000"; coeff <= coeff16; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "10000" => ctrl_coeff <= "010001"; coeff <= coeff17; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "10001" => ctrl_coeff <= "010010"; coeff <= coeff18; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "10010" => ctrl_coeff <= "010011"; coeff <= coeff19; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "10011" => ctrl_coeff <= "010100"; coeff <= coeff20; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "10100" => ctrl_coeff <= "010101"; coeff <= coeff21; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "10101" => ctrl_coeff <= "010110"; coeff <= coeff22; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "10110" => ctrl_coeff <= "010111"; coeff <= coeff23; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "10111" => ctrl_coeff <= "011000"; coeff <= coeff24; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "11000" => ctrl_coeff <= "011001"; coeff <= coeff25; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "11001" => ctrl_coeff <= "011010"; coeff <= coeff26; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "11010" => ctrl_coeff <= "011011"; coeff <= coeff27; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "11011" => ctrl_coeff <= "011100"; coeff <= coeff28; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "11100" => ctrl_coeff <= "011101"; coeff <= coeff29; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "11101" => ctrl_coeff <= "011110"; coeff <= coeff30; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "11110" => ctrl_coeff <= "011111"; coeff <= coeff31; ldcoeff_done <= '0';
send_ctr_nxt <= send_ctr + 1; state_nxt <= s_send;

            when "11111" => ctrl_coeff <= "100000"; coeff <= coeff32; ldcoeff_done <= '1';
send_ctr_nxt <= "00000"; state_nxt <= s_initial;


            when others => ctrl_coeff <= "000001"; coeff <= coeff01; ldcoeff_done <= '0'; send_ctr_nxt
<= "00001"; state_nxt <= s_send;


        end case;


    end case;
```

```
end process;

coeff01_nxt <= coeff_in(13 downto 7)    when counter_2 = "00001" else coeff01;
coeff05_nxt <= coeff_in(6 downto 0)     when counter_2 = "00001" else coeff05;
coeff09_nxt <= coeff_in(13 downto 7)    when counter_2 = "00010" else coeff09;
coeff13_nxt <= coeff_in(6 downto 0)     when counter_2 = "00010" else coeff13;
coeff17_nxt <= coeff_in(13 downto 7)    when counter_2 = "00011" else coeff17;
coeff21_nxt <= coeff_in(6 downto 0)     when counter_2 = "00011" else coeff21;
coeff25_nxt <= coeff_in(13 downto 7)    when counter_2 = "00100" else coeff25;
coeff29_nxt <= coeff_in(6 downto 0)     when counter_2 = "00100" else coeff29;
coeff02_nxt <= coeff_in(13 downto 7)    when counter_2 = "00101" else coeff02;
coeff06_nxt <= coeff_in(6 downto 0)     when counter_2 = "00101" else coeff06;
coeff10_nxt <= coeff_in(13 downto 7)    when counter_2 = "00110" else coeff10;
coeff14_nxt <= coeff_in(6 downto 0)     when counter_2 = "00110" else coeff14;
coeff18_nxt <= coeff_in(13 downto 7)    when counter_2 = "00111" else coeff18;
coeff22_nxt <= coeff_in(6 downto 0)     when counter_2 = "00111" else coeff22;
coeff26_nxt <= coeff_in(13 downto 7)    when counter_2 = "01000" else coeff26;
coeff30_nxt <= coeff_in(6 downto 0)     when counter_2 = "01000" else coeff30;
coeff03_nxt <= coeff_in(13 downto 7)    when counter_2 = "01001" else coeff03;
coeff07_nxt <= coeff_in(6 downto 0)     when counter_2 = "01001" else coeff07;
coeff11_nxt <= coeff_in(13 downto 7)    when counter_2 = "01010" else coeff11;
coeff15_nxt <= coeff_in(6 downto 0)     when counter_2 = "01010" else coeff15;
coeff19_nxt <= coeff_in(13 downto 7)    when counter_2 = "01011" else coeff19;
coeff23_nxt <= coeff_in(6 downto 0)     when counter_2 = "01011" else coeff23;
coeff27_nxt <= coeff_in(13 downto 7)    when counter_2 = "01100" else coeff27;
coeff31_nxt <= coeff_in(6 downto 0)     when counter_2 = "01100" else coeff31;
coeff04_nxt <= coeff_in(13 downto 7)    when counter_2 = "01101" else coeff04;
coeff08_nxt <= coeff_in(6 downto 0)     when counter_2 = "01101" else coeff08;
coeff12_nxt <= coeff_in(13 downto 7)    when counter_2 = "01110" else coeff12;
coeff16_nxt <= coeff_in(6 downto 0)     when counter_2 = "01110" else coeff16;
coeff20_nxt <= coeff_in(13 downto 7)    when counter_2 = "01111" else coeff20;
coeff24_nxt <= coeff_in(6 downto 0)     when counter_2 = "01111" else coeff24;
coeff28_nxt <= coeff_in(13 downto 7)    when counter_2 = "10000" else coeff28;
```

```
coeff32_nxt <= coeff_in(6 downto 0)     when counter_2 = "10000" else coeff32;



counter_ff: FF
generic map(N => 5)
  port map(   D     =>counter_nxt,
              Q     =>counter,
             clk    =>clk,
             reset   =>reset
     );
counter_2_ff: FF
  generic map(N => 5)
  port map(   D     =>counter_2_nxt,
              Q     =>counter_2,
             clk    =>clk,
             reset   =>reset
     );
coeff_01: FF
  generic map(N => 7)
  port map(   D     =>coeff01_nxt,
              Q     =>coeff01,
             clk    =>clk,
             reset   =>reset
     );
coeff_02: FF
  generic map(N => 7)
  port map(   D     =>coeff02_nxt,
              Q     =>coeff02,
             clk    =>clk,
             reset   =>reset
     );
coeff_03: FF
  generic map(N => 7)
```

```
  port map(   D     =>coeff03_nxt,
              Q     =>coeff03,
             clk    =>clk,
             reset   =>reset
      );
coeff_04: FF
  generic map(N => 7)
  port map(   D     =>coeff04_nxt,
              Q     =>coeff04,
             clk    =>clk,
             reset   =>reset
      );
coeff_05: FF
  generic map(N => 7)
  port map(   D     =>coeff05_nxt,
              Q     =>coeff05,
             clk    =>clk,
             reset   =>reset
      );
coeff_06: FF
  generic map(N => 7)
  port map(   D     =>coeff06_nxt,
              Q     =>coeff06,
             clk    =>clk,
             reset   =>reset
      );
coeff_07: FF
  generic map(N => 7)
  port map(   D     =>coeff07_nxt,
              Q     =>coeff07,
             clk    =>clk,
             reset   =>reset
      );
coeff_08: FF
```

```
  generic map(N => 7)
  port map(   D     =>coeff08_nxt,
              Q     =>coeff08,
             clk    =>clk,
             reset   =>reset
      );
coeff_09: FF
  generic map(N => 7)
  port map(   D     =>coeff09_nxt,
              Q     =>coeff09,
             clk    =>clk,
             reset   =>reset
      );
coeff_10: FF
  generic map(N => 7)
  port map(   D     =>coeff10_nxt,
              Q     =>coeff10,
             clk    =>clk,
             reset   =>reset
      );
coeff_11: FF
  generic map(N => 7)
  port map(   D     =>coeff11_nxt,
              Q     =>coeff11,
             clk    =>clk,
             reset   =>reset
      );
coeff_12: FF
  generic map(N => 7)
  port map(   D     =>coeff12_nxt,
              Q     =>coeff12,
             clk    =>clk,
             reset   =>reset
      );
```

```
coeff_13: FF
  generic map(N => 7)
  port map(   D     =>coeff13_nxt,
              Q     =>coeff13,
             clk    =>clk,
             reset   =>reset
      );
coeff_14: FF
  generic map(N => 7)
  port map(   D     =>coeff14_nxt,
              Q     =>coeff14,
             clk    =>clk,
             reset   =>reset
      );
coeff_15: FF
  generic map(N => 7)
  port map(   D     =>coeff15_nxt,
              Q     =>coeff15,
             clk    =>clk,
             reset   =>reset
      );
coeff_16: FF
  generic map(N => 7)
  port map(   D     =>coeff16_nxt,
              Q     =>coeff16,
             clk    =>clk,
             reset   =>reset
      );
coeff_17: FF
  generic map(N => 7)
  port map(   D     =>coeff17_nxt,
              Q     =>coeff17,
             clk    =>clk,
             reset   =>reset
```

```
    );
coeff_18: FF
  generic map(N => 7)
  port map(   D     =>coeff18_nxt,
              Q     =>coeff18,
            clk     =>clk,
            reset   =>reset
    );
coeff_19: FF
  generic map(N => 7)
  port map(   D     =>coeff19_nxt,
              Q     =>coeff19,
            clk     =>clk,
            reset   =>reset
    );
coeff_20: FF
  generic map(N => 7)
  port map(   D     =>coeff20_nxt,
              Q     =>coeff20,
            clk     =>clk,
            reset   =>reset
    );
coeff_21: FF
  generic map(N => 7)
  port map(   D     =>coeff21_nxt,
              Q     =>coeff21,
            clk     =>clk,
            reset   =>reset
    );
coeff_22: FF
  generic map(N => 7)
  port map(   D     =>coeff22_nxt,
              Q     =>coeff22,
            clk     =>clk,
```

```
        reset   =>reset
    );
coeff_23: FF
  generic map(N => 7)
  port map(   D     =>coeff23_nxt,
              Q     =>coeff23,
            clk     =>clk,
            reset   =>reset
    );
coeff_24: FF
  generic map(N => 7)
  port map(   D     =>coeff24_nxt,
              Q     =>coeff24,
            clk     =>clk,
            reset   =>reset
    );
coeff_25: FF
  generic map(N => 7)
  port map(   D     =>coeff25_nxt,
              Q     =>coeff25,
            clk     =>clk,
            reset   =>reset
    );
coeff_26: FF
  generic map(N => 7)
  port map(   D     =>coeff26_nxt,
              Q     =>coeff26,
            clk     =>clk,
            reset   =>reset
    );
coeff_27: FF
  generic map(N => 7)
  port map(   D     =>coeff27_nxt,
              Q     =>coeff27,
```

```
            clk     =>clk,
            reset   =>reset
        );
coeff_28: FF
  generic map(N => 7)
  port map(   D     =>coeff28_nxt,
              Q     =>coeff28,
            clk     =>clk,
            reset   =>reset
        );
coeff_29: FF
  generic map(N => 7)
  port map(   D     =>coeff29_nxt,
              Q     =>coeff29,
            clk     =>clk,
            reset   =>reset
        );
coeff_30: FF
  generic map(N => 7)
  port map(   D     =>coeff30_nxt,
              Q     =>coeff30,
            clk     =>clk,
            reset   =>reset
        );
coeff_31: FF
  generic map(N => 7)
  port map(   D     =>coeff31_nxt,
              Q     =>coeff31,
            clk     =>clk,
            reset   =>reset
        );
coeff_32: FF
  generic map(N => 7)
  port map(   D     =>coeff32_nxt,
```

```
        Q     =>coeff32,
        clk     =>clk,
        reset   =>reset
    );


end Behavioral;
```

6.7 VHDL code for “load_input”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use std.textio.all;


entity load_input is
  Port (
      clk, reset      : in std_logic;

      --from controller-----------------------------------
      ldinput_en      : in std_logic;
      column          : in std_logic_vector(1 downto 0);
      load_en         : in std_logic;
      --from read input-----------------------------------
      input_in        : in std_logic_vector(7 downto 0);
      --to read input-------------------------------------
      ld2reg          : out std_logic;
      --to controller-------------------------------------
      ctrl_input      : out std_logic_vector(3 downto 0);
      input           : out std_logic_vector(7 downto 0);
      ldinput_done    : out std_logic;
      load_done       : out std_logic
```

```
  );
end load_input;

architecture Behavioral of load_input is

component ff is
  generic(N:integer:=1);
  port(   D  :  in std_logic_vector(N-1 downto 0);
          Q  : out std_logic_vector(N-1 downto 0);
         clk  :  in std_logic;
         reset:  in std_logic
       );
end component;

    type state_type is (s_initial, s_keep, s_send);
    signal state_reg, state_nxt : state_type;

    signal counter      : std_logic_vector(5 downto 0);
    signal counter_nxt  : std_logic_vector(5 downto 0);

    signal send_ctrl    : unsigned(4 downto 0) := (others => '0');
    signal send_ctrl_nxt: unsigned(4 downto 0) := (others => '0');

    signal input01      : std_logic_vector(7 downto 0);
    signal input02      : std_logic_vector(7 downto 0);
    signal input03      : std_logic_vector(7 downto 0);
    signal input04      : std_logic_vector(7 downto 0);
    signal input05      : std_logic_vector(7 downto 0);
    signal input06      : std_logic_vector(7 downto 0);
    signal input07      : std_logic_vector(7 downto 0);
    signal input08      : std_logic_vector(7 downto 0);
    signal input09      : std_logic_vector(7 downto 0);
    signal input10      : std_logic_vector(7 downto 0);
    signal input11      : std_logic_vector(7 downto 0);
```

```
signal input12      : std_logic_vector(7 downto 0);
signal input13      : std_logic_vector(7 downto 0);
signal input14      : std_logic_vector(7 downto 0);
signal input15      : std_logic_vector(7 downto 0);
signal input16      : std_logic_vector(7 downto 0);
signal input17      : std_logic_vector(7 downto 0);
signal input18      : std_logic_vector(7 downto 0);
signal input19      : std_logic_vector(7 downto 0);
signal input20      : std_logic_vector(7 downto 0);
signal input21      : std_logic_vector(7 downto 0);
signal input22      : std_logic_vector(7 downto 0);
signal input23      : std_logic_vector(7 downto 0);
signal input24      : std_logic_vector(7 downto 0);
signal input25      : std_logic_vector(7 downto 0);
signal input26      : std_logic_vector(7 downto 0);
signal input27      : std_logic_vector(7 downto 0);
signal input28      : std_logic_vector(7 downto 0);
signal input29      : std_logic_vector(7 downto 0);
signal input30      : std_logic_vector(7 downto 0);
signal input31      : std_logic_vector(7 downto 0);
signal input32      : std_logic_vector(7 downto 0);

signal input01_nxt  : std_logic_vector(7 downto 0);
signal input02_nxt  : std_logic_vector(7 downto 0);
signal input03_nxt  : std_logic_vector(7 downto 0);
signal input04_nxt  : std_logic_vector(7 downto 0);
signal input05_nxt  : std_logic_vector(7 downto 0);
signal input06_nxt  : std_logic_vector(7 downto 0);
signal input07_nxt  : std_logic_vector(7 downto 0);
signal input08_nxt  : std_logic_vector(7 downto 0);
signal input09_nxt  : std_logic_vector(7 downto 0);
signal input10_nxt  : std_logic_vector(7 downto 0);
signal input11_nxt  : std_logic_vector(7 downto 0);
signal input12_nxt  : std_logic_vector(7 downto 0);
```

```
    signal input13_nxt  : std_logic_vector(7 downto 0);
    signal input14_nxt  : std_logic_vector(7 downto 0);
    signal input15_nxt  : std_logic_vector(7 downto 0);
    signal input16_nxt  : std_logic_vector(7 downto 0);
    signal input17_nxt  : std_logic_vector(7 downto 0);
    signal input18_nxt  : std_logic_vector(7 downto 0);
    signal input19_nxt  : std_logic_vector(7 downto 0);
    signal input20_nxt  : std_logic_vector(7 downto 0);
    signal input21_nxt  : std_logic_vector(7 downto 0);
    signal input22_nxt  : std_logic_vector(7 downto 0);
    signal input23_nxt  : std_logic_vector(7 downto 0);
    signal input24_nxt  : std_logic_vector(7 downto 0);
    signal input25_nxt  : std_logic_vector(7 downto 0);
    signal input26_nxt  : std_logic_vector(7 downto 0);
    signal input27_nxt  : std_logic_vector(7 downto 0);
    signal input28_nxt  : std_logic_vector(7 downto 0);
    signal input29_nxt  : std_logic_vector(7 downto 0);
    signal input30_nxt  : std_logic_vector(7 downto 0);
    signal input31_nxt  : std_logic_vector(7 downto 0);
    signal input32_nxt  : std_logic_vector(7 downto 0);


begin


process (clk, reset)
begin
    if reset = '1' then
        state_reg <= s_initial;
        send_ctrl <= (others => '0');
    elsif (clk'event and clk = '1') then
        state_reg <= state_nxt;
        send_ctrl <= send_ctrl_nxt;
    end if;
end process;
```

```
process(state_reg, ldinput_en, counter, column, send_ctrl, load_en)
begin
        load_done <= '0';
        ctrl_input <= (others => '0');
        input <= (others => '0');
        counter_nxt <= counter;
        send_ctrl_nxt <= send_ctrl;

   case state_reg is
      when s_initial =>
        ld2reg <= '0';
        ctrl_input <= (others => '0');
        input <= (others => '0');
        load_done <= '0';
        counter_nxt <= (others => '0');
        send_ctrl_nxt <= (others => '0');
        if ldinput_en = '1' then
           state_nxt <= s_keep;
        else
           state_nxt <= s_initial;
        end if;

      when s_keep =>
        ld2reg <= '1';
        counter_nxt <= counter + 1;
        if load_en = '1' then
           state_nxt <= s_send;
        else
           state_nxt <= s_keep;
        end if;

      when s_send =>
        ld2reg <= '0';
```

```
        case column is
            when "00" =>
                state_nxt <= s_send;
                case send_ctrl is
                    when "00000" => ctrl_input <= "0001"; input <= input01; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "00001" => ctrl_input <= "0010"; input <= input02; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "00010" => ctrl_input <= "0011"; input <= input03; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "00011" => ctrl_input <= "0100"; input <= input04; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "00100" => ctrl_input <= "0101"; input <= input05; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "00101" => ctrl_input <= "0110"; input <= input06; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "00110" => ctrl_input <= "0111"; input <= input07; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "00111" => ctrl_input <= "1000"; input <= input08; load_done <= '1';
send_ctrl_nxt <= "01000";
                    when others => load_done <= '0'; send_ctrl_nxt <= send_ctrl;
                end case;

            when "01" =>
                state_nxt <= s_send;
                case send_ctrl is
                    when "01000" => ctrl_input <= "0001"; input <= input09; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "01001" => ctrl_input <= "0010"; input <= input10; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "01010" => ctrl_input <= "0011"; input <= input11; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "01011" => ctrl_input <= "0100"; input <= input12; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "01100" => ctrl_input <= "0101"; input <= input13; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
                    when "01101" => ctrl_input <= "0110"; input <= input14; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;
```

```
                when "01110" => ctrl_input <= "0111"; input <= input15; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;

                when "01111" => ctrl_input <= "1000"; input <= input16; load_done <= '1';
send_ctrl_nxt <= "10000";

                when others => load_done <= '0'; send_ctrl_nxt <= send_ctrl;

            end case;


        when "10" =>

            state_nxt <= s_send;

            case send_ctrl is

                when "10000" => ctrl_input <= "0001"; input <= input17; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;

                when "10001" => ctrl_input <= "0010"; input <= input18; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;

                when "10010" => ctrl_input <= "0011"; input <= input19; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;

                when "10011" => ctrl_input <= "0100"; input <= input20; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;

                when "10100" => ctrl_input <= "0101"; input <= input21; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;

                when "10101" => ctrl_input <= "0110"; input <= input22; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;

                when "10110" => ctrl_input <= "0111"; input <= input23; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1;

                when "10111" => ctrl_input <= "1000"; input <= input24; load_done <= '1';
send_ctrl_nxt <= "11000";

                when others => load_done <= '0'; send_ctrl_nxt <= send_ctrl;

            end case;



        when "11" =>

            case send_ctrl is

                when "11000" => ctrl_input <= "0001"; input <= input25; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1; state_nxt <= s_send;

                when "11001" => ctrl_input <= "0010"; input <= input26; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1; state_nxt <= s_send;

                when "11010" => ctrl_input <= "0011"; input <= input27; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1; state_nxt <= s_send;
```

```
                when "11011" => ctrl_input <= "0100"; input <= input28; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1; state_nxt <= s_send;

                when "11100" => ctrl_input <= "0101"; input <= input29; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1; state_nxt <= s_send;

                when "11101" => ctrl_input <= "0110"; input <= input30; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1; state_nxt <= s_send;

                when "11110" => ctrl_input <= "0111"; input <= input31; load_done <= '0';
send_ctrl_nxt <= send_ctrl + 1; state_nxt <= s_send;

                when "11111" => ctrl_input <= "1000"; input <= input32; load_done <= '1';
send_ctrl_nxt <= (others => '0'); state_nxt <= s_initial; --at the end of a matrics and jump to the
begining

                when others => load_done <= '0'; send_ctrl_nxt <= send_ctrl; state_nxt <= s_send;

            end case;


          when others =>

            state_nxt <= s_send;

        end case;


   end case;



end process;



input01_nxt <= input_in(7 downto 0) when counter = "000001" else input01;

input02_nxt <= input_in(7 downto 0) when counter = "000010" else input02;

input03_nxt <= input_in(7 downto 0) when counter = "000011" else input03;

input04_nxt <= input_in(7 downto 0) when counter = "000100" else input04;

input05_nxt <= input_in(7 downto 0) when counter = "000101" else input05;

input06_nxt <= input_in(7 downto 0) when counter = "000110" else input06;

input07_nxt <= input_in(7 downto 0) when counter = "000111" else input07;

input08_nxt <= input_in(7 downto 0) when counter = "001000" else input08;

input09_nxt <= input_in(7 downto 0) when counter = "001001" else input09;

input10_nxt <= input_in(7 downto 0) when counter = "001010" else input10;

input11_nxt <= input_in(7 downto 0) when counter = "001011" else input11;

input12_nxt <= input_in(7 downto 0) when counter = "001100" else input12;
```

```
input13_nxt <= input_in(7 downto 0) when counter = "001101" else input13;
input14_nxt <= input_in(7 downto 0) when counter = "001110" else input14;
input15_nxt <= input_in(7 downto 0) when counter = "001111" else input15;
input16_nxt <= input_in(7 downto 0) when counter = "010000" else input16;
input17_nxt <= input_in(7 downto 0) when counter = "010001" else input17;
input18_nxt <= input_in(7 downto 0) when counter = "010010" else input18;
input19_nxt <= input_in(7 downto 0) when counter = "010011" else input19;
input20_nxt <= input_in(7 downto 0) when counter = "010100" else input20;
input21_nxt <= input_in(7 downto 0) when counter = "010101" else input21;
input22_nxt <= input_in(7 downto 0) when counter = "010110" else input22;
input23_nxt <= input_in(7 downto 0) when counter = "010111" else input23;
input24_nxt <= input_in(7 downto 0) when counter = "011000" else input24;
input25_nxt <= input_in(7 downto 0) when counter = "011001" else input25;
input26_nxt <= input_in(7 downto 0) when counter = "011010" else input26;
input27_nxt <= input_in(7 downto 0) when counter = "011011" else input27;
input28_nxt <= input_in(7 downto 0) when counter = "011100" else input28;
input29_nxt <= input_in(7 downto 0) when counter = "011101" else input29;
input30_nxt <= input_in(7 downto 0) when counter = "011110" else input30;
input31_nxt <= input_in(7 downto 0) when counter = "011111" else input31;
input32_nxt <= input_in(7 downto 0) when counter = "100000" else input32;

ldinput_done <= '1' when counter = "100000" else '0';

counter_ctrl: FF
  generic map(N => 6)
  port map(   D     =>counter_nxt,
              Q     =>counter,
            clk     =>clk,
            reset   =>reset
      );

input_01: FF
  generic map(N => 8)
  port map(   D     =>input01_nxt,
```

```
            Q     =>input01,
           clk    =>clk,
           reset  =>reset
      );
input_02: FF
  generic map(N => 8)
  port map(  D     =>input02_nxt,
            Q     =>input02,
           clk    =>clk,
           reset  =>reset
      );
input_03: FF
  generic map(N => 8)
  port map(  D     =>input03_nxt,
            Q     =>input03,
           clk    =>clk,
           reset  =>reset
      );
input_04: FF
  generic map(N => 8)
  port map(  D     =>input04_nxt,
            Q     =>input04,
           clk    =>clk,
           reset  =>reset
      );
input_05: FF
  generic map(N => 8)
  port map(  D     =>input05_nxt,
            Q     =>input05,
           clk    =>clk,
           reset  =>reset
      );
input_06: FF
  generic map(N => 8)
```

```
  port map(   D     =>input06_nxt,
              Q     =>input06,
            clk     =>clk,
            reset   =>reset
      );
input_07: FF
  generic map(N => 8)
  port map(   D     =>input07_nxt,
              Q     =>input07,
            clk     =>clk,
            reset   =>reset
      );
input_08: FF
  generic map(N => 8)
  port map(   D     =>input08_nxt,
              Q     =>input08,
            clk     =>clk,
            reset   =>reset
      );
input_09: FF
  generic map(N => 8)
  port map(   D     =>input09_nxt,
              Q     =>input09,
            clk     =>clk,
            reset   =>reset
      );
input_10: FF
  generic map(N => 8)
  port map(   D     =>input10_nxt,
              Q     =>input10,
            clk     =>clk,
            reset   =>reset
      );
input_11: FF
```

```
  generic map(N => 8)
  port map(   D     =>input11_nxt,
              Q     =>input11,
             clk    =>clk,
             reset   =>reset
      );
input_12: FF
  generic map(N => 8)
  port map(   D     =>input12_nxt,
              Q     =>input12,
             clk    =>clk,
             reset   =>reset
      );
input_13: FF
  generic map(N => 8)
  port map(   D     =>input13_nxt,
              Q     =>input13,
             clk    =>clk,
             reset   =>reset
      );
input_14: FF
  generic map(N => 8)
  port map(   D     =>input14_nxt,
              Q     =>input14,
             clk    =>clk,
             reset   =>reset
      );
input_15: FF
  generic map(N => 8)
  port map(   D     =>input15_nxt,
              Q     =>input15,
             clk    =>clk,
             reset   =>reset
      );
```

```
input_16: FF
  generic map(N => 8)
  port map(   D     =>input16_nxt,
              Q     =>input16,
             clk    =>clk,
             reset   =>reset
      );
input_17: FF
  generic map(N => 8)
  port map(   D     =>input17_nxt,
              Q     =>input17,
             clk    =>clk,
             reset   =>reset
      );
input_18: FF
  generic map(N => 8)
  port map(   D     =>input18_nxt,
              Q     =>input18,
             clk    =>clk,
             reset   =>reset
      );
input_19: FF
  generic map(N => 8)
  port map(   D     =>input19_nxt,
              Q     =>input19,
             clk    =>clk,
             reset   =>reset
      );
input_20: FF
  generic map(N => 8)
  port map(   D     =>input20_nxt,
              Q     =>input20,
             clk    =>clk,
             reset   =>reset
```

```
    );
input_21: FF
  generic map(N => 8)
  port map(   D     =>input21_nxt,
              Q     =>input21,
             clk    =>clk,
             reset   =>reset
    );
input_22: FF
  generic map(N => 8)
  port map(   D     =>input22_nxt,
              Q     =>input22,
             clk    =>clk,
             reset   =>reset
    );
input_23: FF
  generic map(N => 8)
  port map(   D     =>input23_nxt,
              Q     =>input23,
             clk    =>clk,
             reset   =>reset
    );
input_24: FF
  generic map(N => 8)
  port map(   D     =>input24_nxt,
              Q     =>input24,
             clk    =>clk,
             reset   =>reset
    );
input_25: FF
  generic map(N => 8)
  port map(   D     =>input25_nxt,
              Q     =>input25,
             clk    =>clk,
```

```
        reset   =>reset
    );
input_26: FF
  generic map(N => 8)
  port map(   D     =>input26_nxt,
              Q     =>input26,
            clk     =>clk,
            reset   =>reset
    );
input_27: FF
  generic map(N => 8)
  port map(   D     =>input27_nxt,
              Q     =>input27,
            clk     =>clk,
            reset   =>reset
    );
input_28: FF
  generic map(N => 8)
  port map(   D     =>input28_nxt,
              Q     =>input28,
            clk     =>clk,
            reset   =>reset
    );
input_29: FF
  generic map(N => 8)
  port map(   D     =>input29_nxt,
              Q     =>input29,
            clk     =>clk,
            reset   =>reset
    );
input_30: FF
  generic map(N => 8)
  port map(   D     =>input30_nxt,
              Q     =>input30,
```

```
        clk     =>clk,
        reset   =>reset
    );
input_31: FF
 generic map(N => 8)
 port map(   D     =>input31_nxt,
          Q     =>input31,
        clk     =>clk,
        reset   =>reset
    );
input_32: FF
 generic map(N => 8)
 port map(   D     =>input32_nxt,
          Q     =>input32,
        clk     =>clk,
        reset   =>reset
    );



end Behavioral;
```

6.8 VHDL code for “operation”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity operation is
 Port (
      clk, reset      : in std_logic;

      op_en           : in std_logic;
```

```
        data2op         : in std_logic_vector(7 downto 0);
        address2op      : in std_logic_vector(5 downto 0);
 ----------------------------------------------------------------
        -- to the controller----------------------------------

        op_done         : out std_logic;

        result1_out     : out std_logic_vector(18 downto 0);
        result2_out     : out std_logic_vector(18 downto 0);
        result3_out     : out std_logic_vector(18 downto 0);
        result4_out     : out std_logic_vector(18 downto 0);

        --to the compare-----------------------------------------
        compare_out     : out std_logic_vector(18 downto 0)

  );
end operation;

architecture Behavioral of operation is

    type state_type is (s_initial, s_store, s_mult1, s_keep1, s_mult2, s_keep2, s_mult3, s_keep3,
s_mult4, s_keep4, s_add, s_send_data, s_send_compare);
    signal state_reg, state_nxt : state_type;

    signal output_test  : std_logic_vector(16 downto 0);

    signal start_store  : std_logic;
    signal column       : std_logic_vector(1 downto 0);
    signal column_nxt   : std_logic_vector(1 downto 0);
    signal data2op_done : std_logic;

    signal input_test   : std_logic_vector(7 downto 0);
    signal coeff01      : std_logic_vector(7 downto 0);
    signal coeff02      : std_logic_vector(7 downto 0);
```

```
signal coeff03     : std_logic_vector(7 downto 0);
signal coeff04     : std_logic_vector(7 downto 0);
signal coeff05     : std_logic_vector(7 downto 0);
signal coeff06     : std_logic_vector(7 downto 0);
signal coeff07     : std_logic_vector(7 downto 0);
signal coeff08     : std_logic_vector(7 downto 0);
signal coeff09     : std_logic_vector(7 downto 0);
signal coeff10     : std_logic_vector(7 downto 0);
signal coeff11     : std_logic_vector(7 downto 0);
signal coeff12     : std_logic_vector(7 downto 0);
signal coeff13     : std_logic_vector(7 downto 0);
signal coeff14     : std_logic_vector(7 downto 0);
signal coeff15     : std_logic_vector(7 downto 0);
signal coeff16     : std_logic_vector(7 downto 0);
signal coeff17     : std_logic_vector(7 downto 0);
signal coeff18     : std_logic_vector(7 downto 0);
signal coeff19     : std_logic_vector(7 downto 0);
signal coeff20     : std_logic_vector(7 downto 0);
signal coeff21     : std_logic_vector(7 downto 0);
signal coeff22     : std_logic_vector(7 downto 0);
signal coeff23     : std_logic_vector(7 downto 0);
signal coeff24     : std_logic_vector(7 downto 0);
signal coeff25     : std_logic_vector(7 downto 0);
signal coeff26     : std_logic_vector(7 downto 0);
signal coeff27     : std_logic_vector(7 downto 0);
signal coeff28     : std_logic_vector(7 downto 0);
signal coeff29     : std_logic_vector(7 downto 0);
signal coeff30     : std_logic_vector(7 downto 0);
signal coeff31     : std_logic_vector(7 downto 0);
signal coeff32     : std_logic_vector(7 downto 0);

signal input01     : std_logic_vector(7 downto 0);
signal input02     : std_logic_vector(7 downto 0);
signal input03     : std_logic_vector(7 downto 0);
```

```
signal input04      : std_logic_vector(7 downto 0);
signal input05      : std_logic_vector(7 downto 0);
signal input06      : std_logic_vector(7 downto 0);
signal input07      : std_logic_vector(7 downto 0);
signal input08      : std_logic_vector(7 downto 0);

signal coeff01_nxt      : std_logic_vector(7 downto 0);
signal coeff02_nxt      : std_logic_vector(7 downto 0);
signal coeff03_nxt      : std_logic_vector(7 downto 0);
signal coeff04_nxt      : std_logic_vector(7 downto 0);
signal coeff05_nxt      : std_logic_vector(7 downto 0);
signal coeff06_nxt      : std_logic_vector(7 downto 0);
signal coeff07_nxt      : std_logic_vector(7 downto 0);
signal coeff08_nxt      : std_logic_vector(7 downto 0);
signal coeff09_nxt      : std_logic_vector(7 downto 0);
signal coeff10_nxt      : std_logic_vector(7 downto 0);
signal coeff11_nxt      : std_logic_vector(7 downto 0);
signal coeff12_nxt      : std_logic_vector(7 downto 0);
signal coeff13_nxt      : std_logic_vector(7 downto 0);
signal coeff14_nxt      : std_logic_vector(7 downto 0);
signal coeff15_nxt      : std_logic_vector(7 downto 0);
signal coeff16_nxt      : std_logic_vector(7 downto 0);
signal coeff17_nxt      : std_logic_vector(7 downto 0);
signal coeff18_nxt      : std_logic_vector(7 downto 0);
signal coeff19_nxt      : std_logic_vector(7 downto 0);
signal coeff20_nxt      : std_logic_vector(7 downto 0);
signal coeff21_nxt      : std_logic_vector(7 downto 0);
signal coeff22_nxt      : std_logic_vector(7 downto 0);
signal coeff23_nxt      : std_logic_vector(7 downto 0);
signal coeff24_nxt      : std_logic_vector(7 downto 0);
signal coeff25_nxt      : std_logic_vector(7 downto 0);
signal coeff26_nxt      : std_logic_vector(7 downto 0);
signal coeff27_nxt      : std_logic_vector(7 downto 0);
signal coeff28_nxt      : std_logic_vector(7 downto 0);
```

```
signal coeff29_nxt      : std_logic_vector(7 downto 0);
signal coeff30_nxt      : std_logic_vector(7 downto 0);
signal coeff31_nxt      : std_logic_vector(7 downto 0);
signal coeff32_nxt      : std_logic_vector(7 downto 0);

signal input01_nxt      : std_logic_vector(7 downto 0);
signal input02_nxt      : std_logic_vector(7 downto 0);
signal input03_nxt      : std_logic_vector(7 downto 0);
signal input04_nxt      : std_logic_vector(7 downto 0);
signal input05_nxt      : std_logic_vector(7 downto 0);
signal input06_nxt      : std_logic_vector(7 downto 0);
signal input07_nxt      : std_logic_vector(7 downto 0);
signal input08_nxt      : std_logic_vector(7 downto 0);

signal output_reg1  : std_logic_vector(15 downto 0);
signal output_reg2  : std_logic_vector(15 downto 0);
signal output_reg3  : std_logic_vector(15 downto 0);
signal output_reg4  : std_logic_vector(15 downto 0);
signal output_reg5  : std_logic_vector(15 downto 0);
signal output_reg6  : std_logic_vector(15 downto 0);
signal output_reg7  : std_logic_vector(15 downto 0);
signal output_reg8  : std_logic_vector(15 downto 0);

signal output_reg1_nxt  : std_logic_vector(15 downto 0);
signal output_reg2_nxt  : std_logic_vector(15 downto 0);
signal output_reg3_nxt  : std_logic_vector(15 downto 0);
signal output_reg4_nxt  : std_logic_vector(15 downto 0);
signal output_reg5_nxt  : std_logic_vector(15 downto 0);
signal output_reg6_nxt  : std_logic_vector(15 downto 0);
signal output_reg7_nxt  : std_logic_vector(15 downto 0);
signal output_reg8_nxt  : std_logic_vector(15 downto 0);

signal output1      : std_logic_vector(18 downto 0);
signal output2      : std_logic_vector(18 downto 0);
```

```
    signal output3      : std_logic_vector(18 downto 0);
    signal output4      : std_logic_vector(18 downto 0);

    signal output1_nxt  : std_logic_vector(18 downto 0);
    signal output2_nxt  : std_logic_vector(18 downto 0);
    signal output3_nxt  : std_logic_vector(18 downto 0);
    signal output4_nxt  : std_logic_vector(18 downto 0);

    signal result1      : std_logic_vector(18 downto 0);
    signal result2      : std_logic_vector(18 downto 0);
    signal result3      : std_logic_vector(18 downto 0);
    signal result4      : std_logic_vector(18 downto 0);
    signal result1_nxt  : std_logic_vector(18 downto 0);
    signal result2_nxt  : std_logic_vector(18 downto 0);
    signal result3_nxt  : std_logic_vector(18 downto 0);
    signal result4_nxt  : std_logic_vector(18 downto 0);

    signal compare      : std_logic_vector(18 downto 0) := (others => '0');
    signal compare_nxt  : std_logic_vector(18 downto 0) := (others => '0');

    signal mult1, mult2, mult3, mult4 : std_logic_vector(7 downto 0);
    signal mult1_nxt, mult2_nxt, mult3_nxt, mult4_nxt : std_logic_vector(7 downto 0);
    signal result_1, result_2         : std_logic_vector(15 downto 0);

begin

--state ctrl-----------------------------------
process (clk, reset)
begin
    if reset = '1' then
        state_reg <= s_initial;
    elsif (clk'event and clk = '1') then
        state_reg <= state_nxt;
    end if;
```

```
end process;

result_1 <= mult1 * mult2;
result_2 <= mult3 * mult4;

result1_out <= result1;
result2_out <= result2;
result3_out <= result3;
result4_out <= result4;

compare_out <= compare;
-------------------------------------------------------------------------------
process (clk, reset)
begin
   if reset = '1' then
      column <= "00";
      compare <= (others => '0');
      output_reg1 <= (others => '0');
      output_reg2 <= (others => '0');
      output_reg3 <= (others => '0');
      output_reg4 <= (others => '0');
      output_reg5 <= (others => '0');
      output_reg6 <= (others => '0');
      output_reg7 <= (others => '0');
      output_reg8 <= (others => '0');
      output1 <= (others => '0');
      output2 <= (others => '0');
      output3 <= (others => '0');
      output4 <= (others => '0');
      mult1 <= (others => '0');
      mult2 <= (others => '0');
      mult3 <= (others => '0');
      mult4 <= (others => '0');
```

```
    elsif (clk'event and clk = '1') then
        column <= column_nxt;
        compare <= compare_nxt;
        output_reg1 <= output_reg1_nxt;
        output_reg2 <= output_reg2_nxt;
        output_reg3 <= output_reg3_nxt;
        output_reg4 <= output_reg4_nxt;
        output_reg5 <= output_reg5_nxt;
        output_reg6 <= output_reg6_nxt;
        output_reg7 <= output_reg7_nxt;
        output_reg8 <= output_reg8_nxt;
        output1 <= output1_nxt;
        output2 <= output2_nxt;
        output3 <= output3_nxt;
        output4 <= output4_nxt;
        mult1 <= mult1_nxt;
        mult2 <= mult2_nxt;
        mult3 <= mult3_nxt;
        mult4 <= mult4_nxt;

    end if;
end process;

----------------------------------------
process (state_reg, data2op_done,
        output_reg1, output_reg2, output_reg3, output_reg4, output_reg5, output_reg6, output_reg7,
output_reg8,
        output1, output2, output3, output4,
        compare, result_1,result_2
        )

begin

    op_done <= '0';
```

```
result1_nxt <= result1;
result2_nxt <= result2;
result3_nxt <= result3;
result4_nxt <= result4;

column_nxt <= column;


output_reg1_nxt <= output_reg1;
output_reg2_nxt <= output_reg2;
output_reg3_nxt <= output_reg3;
output_reg4_nxt <= output_reg4;
output_reg5_nxt <= output_reg5;
output_reg6_nxt <= output_reg6;
output_reg7_nxt <= output_reg7;
output_reg8_nxt <= output_reg8;

output1_nxt <= output1;
output2_nxt <= output2;
output3_nxt <= output3;
output4_nxt <= output4;
mult1_nxt <= mult1;
mult2_nxt <= mult2;
mult3_nxt <= mult3;
mult4_nxt <= mult4;

compare_nxt <= compare;

case state_reg is

   when s_initial =>
      column_nxt <= "00";
      op_done <= '0';
      --compare_out <= (others => '0');
```

```
        if op_en = '1' then
            state_nxt <= s_store;
        else
            state_nxt<= s_initial;
        end if;

    when s_store =>
        if data2op_done = '1' then
            state_nxt <= s_mult1;
        else
            state_nxt <= s_store;
        end if;

    when s_mult1 =>
        state_nxt <= s_keep1;
        case column is
            when "00" => mult1_nxt <= input01; mult2_nxt <= coeff01; mult3_nxt <= input02;
mult4_nxt <= coeff05;
            when "01" => mult1_nxt <= input01; mult2_nxt <= coeff02; mult3_nxt <= input02;
mult4_nxt <= coeff06;
            when "10" => mult1_nxt <= input01; mult2_nxt <= coeff03; mult3_nxt <= input02;
mult4_nxt <= coeff07;
            when "11" => mult1_nxt <= input01; mult2_nxt <= coeff04; mult3_nxt <= input02;
mult4_nxt <= coeff08;
            when others => mult1_nxt <= input01; mult2_nxt <= coeff01; mult3_nxt <= input02;
mult4_nxt <= coeff05;
        end case;

    when s_keep1 =>
        state_nxt <= s_mult2;
        output_reg1_nxt <= result_1; output_reg2_nxt <= result_2;

    when s_mult2 =>
        state_nxt <= s_keep2;
        case column is
```

```
            when "00" => mult1_nxt <= input03; mult2_nxt <= coeff09; mult3_nxt <= input04;
mult4_nxt <= coeff13;

            when "01" => mult1_nxt <= input03; mult2_nxt <= coeff10; mult3_nxt <= input04;
mult4_nxt <= coeff14;

            when "10" => mult1_nxt <= input03; mult2_nxt <= coeff11; mult3_nxt <= input04;
mult4_nxt <= coeff15;

            when "11" => mult1_nxt <= input03; mult2_nxt <= coeff12; mult3_nxt <= input04;
mult4_nxt <= coeff16;

            when others => mult1_nxt <= input03; mult2_nxt <= coeff09; mult3_nxt <= input04;
mult4_nxt <= coeff13;

        end case;


    when s_keep2 =>

        state_nxt <= s_mult3;

        output_reg3_nxt <= result_1; output_reg4_nxt <= result_2;


    when s_mult3 =>

        state_nxt <= s_keep3;

        case column is

            when "00" => mult1_nxt <= input05; mult2_nxt <= coeff17; mult3_nxt <= input06;
mult4_nxt <= coeff21;

            when "01" => mult1_nxt <= input05; mult2_nxt <= coeff18; mult3_nxt <= input06;
mult4_nxt <= coeff22;

            when "10" => mult1_nxt <= input05; mult2_nxt <= coeff19; mult3_nxt <= input06;
mult4_nxt <= coeff23;

            when "11" => mult1_nxt <= input05; mult2_nxt <= coeff20; mult3_nxt <= input06;
mult4_nxt <= coeff24;

            when others => mult1_nxt <= input05; mult2_nxt <= coeff17; mult3_nxt <= input06;
mult4_nxt <= coeff21;

        end case;


    when s_keep3 =>

        state_nxt <= s_mult4;

        output_reg5_nxt <= result_1; output_reg6_nxt <= result_2;


    when s_mult4 =>

        state_nxt <= s_keep4;

        case column is
```

```
            when "00" => mult1_nxt <= input07; mult2_nxt <= coeff25; mult3_nxt <= input08;
mult4_nxt <= coeff29;

            when "01" => mult1_nxt <= input07; mult2_nxt <= coeff26; mult3_nxt <= input08;
mult4_nxt <= coeff30;

            when "10" => mult1_nxt <= input07; mult2_nxt <= coeff27; mult3_nxt <= input08;
mult4_nxt <= coeff31;

            when "11" => mult1_nxt <= input07; mult2_nxt <= coeff28; mult3_nxt <= input08;
mult4_nxt <= coeff32;

            when others => mult1_nxt <= input07; mult2_nxt <= coeff25; mult3_nxt <= input08;
mult4_nxt <= coeff29;

          end case;


       when s_keep4 =>

          state_nxt <= s_add;

          output_reg7_nxt <= result_1; output_reg8_nxt <= result_2;


       when s_add =>

          case column is

             when "00" =>

                output1_nxt <= output_reg1 + output_reg2 + output_reg3 + output_reg4 + output_reg5 +
output_reg6 + output_reg7 + output_reg8 + "0000000000000000000";

                state_nxt <= s_send_data;

             when "01" =>

                output2_nxt <= output_reg1 + output_reg2 + output_reg3 + output_reg4 + output_reg5 +
output_reg6 + output_reg7 + output_reg8 + "0000000000000000000";

                state_nxt <= s_send_data;

             when "10" =>

                output3_nxt <= output_reg1 + output_reg2 + output_reg3 + output_reg4 + output_reg5 +
output_reg6 + output_reg7 + output_reg8 + "0000000000000000000";

                state_nxt <= s_send_data;

             when "11" =>

                output4_nxt <= output_reg1 + output_reg2 + output_reg3 + output_reg4 + output_reg5 +
output_reg6 + output_reg7 + output_reg8 + "0000000000000000000";

                state_nxt <= s_send_data;

             when others =>

                state_nxt <= s_add;

          end case;
```

```
when s_send_data =>
   case column is
      when "00" =>
         state_nxt <= s_mult1;
         column_nxt <= column + 1;
         result1_nxt <= output1;
         if compare < output1 then
            compare_nxt <= output1;
         else
            compare_nxt <= compare;
         end if;

      when "01" =>
         state_nxt <= s_mult1;
         column_nxt <= column + 1;
         result2_nxt <= output2;
         if compare < output2 then
            compare_nxt <= output2;
         else
            compare_nxt <= compare;
         end if;

      when "10" =>
         state_nxt <= s_mult1;
         column_nxt <= column + 1;
         result3_nxt <= output3;
         if compare < output3 then
            compare_nxt <= output3;
         else
            compare_nxt <= compare;
         end if;

      when "11" =>
```

```
                state_nxt <= s_send_compare;
                column_nxt <= "00";
                result4_nxt <= output4;
                if compare < output4 then
                   compare_nxt <= output4;
                else
                   compare_nxt <= compare;
                end if;

             when others =>
                state_nxt <= s_mult1;
          end case;

       when s_send_compare =>
          op_done <= '1';
          state_nxt <= s_initial;
   end case;


end process;

-------------------------------------------------------------------------------

process(clk, reset)
begin
   if reset = '1' then
      coeff01 <= (others => '0');
      coeff02 <= (others => '0');
      coeff03 <= (others => '0');
      coeff04 <= (others => '0');
      coeff05 <= (others => '0');
      coeff06 <= (others => '0');
      coeff07 <= (others => '0');
      coeff08 <= (others => '0');
```

```
coeff09 <= (others => '0');
coeff10 <= (others => '0');
coeff11 <= (others => '0');
coeff12 <= (others => '0');
coeff13 <= (others => '0');
coeff14 <= (others => '0');
coeff15 <= (others => '0');
coeff16 <= (others => '0');
coeff17 <= (others => '0');
coeff18 <= (others => '0');
coeff19 <= (others => '0');
coeff20 <= (others => '0');
coeff21 <= (others => '0');
coeff22 <= (others => '0');
coeff23 <= (others => '0');
coeff24 <= (others => '0');
coeff25 <= (others => '0');
coeff26 <= (others => '0');
coeff27 <= (others => '0');
coeff28 <= (others => '0');
coeff29 <= (others => '0');
coeff30 <= (others => '0');
coeff31 <= (others => '0');
coeff32 <= (others => '0');
input01 <= (others => '0');
input02 <= (others => '0');
input03 <= (others => '0');
input04 <= (others => '0');
input05 <= (others => '0');
input06 <= (others => '0');
input07 <= (others => '0');
input08 <= (others => '0');
result1 <= (others => '0');
result2 <= (others => '0');
```

```
    result3 <= (others => '0');
    result4 <= (others => '0');
elsif (clk'event and clk = '1') then
    coeff01 <= coeff01_nxt;
    coeff02 <= coeff02_nxt;
    coeff03 <= coeff03_nxt;
    coeff04 <= coeff04_nxt;
    coeff05 <= coeff05_nxt;
    coeff06 <= coeff06_nxt;
    coeff07 <= coeff07_nxt;
    coeff08 <= coeff08_nxt;
    coeff09 <= coeff09_nxt;
    coeff10 <= coeff10_nxt;
    coeff11 <= coeff11_nxt;
    coeff12 <= coeff12_nxt;
    coeff13 <= coeff13_nxt;
    coeff14 <= coeff14_nxt;
    coeff15 <= coeff15_nxt;
    coeff16 <= coeff16_nxt;
    coeff17 <= coeff17_nxt;
    coeff18 <= coeff18_nxt;
    coeff19 <= coeff19_nxt;
    coeff20 <= coeff20_nxt;
    coeff21 <= coeff21_nxt;
    coeff22 <= coeff22_nxt;
    coeff23 <= coeff23_nxt;
    coeff24 <= coeff24_nxt;
    coeff25 <= coeff25_nxt;
    coeff26 <= coeff26_nxt;
    coeff27 <= coeff27_nxt;
    coeff28 <= coeff28_nxt;
    coeff29 <= coeff29_nxt;
    coeff30 <= coeff30_nxt;
    coeff31 <= coeff31_nxt;
```

```
        coeff32 <= coeff32_nxt;
        input01 <= input01_nxt;
        input02 <= input02_nxt;
        input03 <= input03_nxt;
        input04 <= input04_nxt;
        input05 <= input05_nxt;
        input06 <= input06_nxt;
        input07 <= input07_nxt;
        input08 <= input08_nxt;
        result1 <= result1_nxt;
        result2 <= result2_nxt;
        result3 <= result3_nxt;
        result4 <= result4_nxt;
    end if;
end process;

store_data : process( address2op, data2op)--start_store,
begin
        coeff01_nxt <= coeff01;
        coeff02_nxt <= coeff02;
        coeff03_nxt <= coeff03;
        coeff04_nxt <= coeff04;
        coeff05_nxt <= coeff05;
        coeff06_nxt <= coeff06;
        coeff07_nxt <= coeff07;
        coeff08_nxt <= coeff08;
        coeff09_nxt <= coeff09;
        coeff10_nxt <= coeff10;
        coeff11_nxt <= coeff11;
        coeff12_nxt <= coeff12;
        coeff13_nxt <= coeff13;
        coeff14_nxt <= coeff14;
        coeff15_nxt <= coeff15;
        coeff16_nxt <= coeff16;
```

```
coeff17_nxt <= coeff17;
coeff18_nxt <= coeff18;
coeff19_nxt <= coeff19;
coeff20_nxt <= coeff20;
coeff21_nxt <= coeff21;
coeff22_nxt <= coeff22;
coeff23_nxt <= coeff23;
coeff24_nxt <= coeff24;
coeff25_nxt <= coeff25;
coeff26_nxt <= coeff26;
coeff27_nxt <= coeff27;
coeff28_nxt <= coeff28;
coeff29_nxt <= coeff29;
coeff30_nxt <= coeff30;
coeff31_nxt <= coeff31;
coeff32_nxt <= coeff32;
input01_nxt <= input01;
input02_nxt <= input02;
input03_nxt <= input03;
input04_nxt <= input04;
input05_nxt <= input05;
input06_nxt <= input06;
input07_nxt <= input07;
input08_nxt <= input08;

case address2op is
   when "000001" =>  coeff01_nxt <= data2op; data2op_done <= '0';
   when "000010" =>  coeff02_nxt <= data2op; data2op_done <= '0';
   when "000011" =>  coeff03_nxt <= data2op; data2op_done <= '0';
   when "000100" =>  coeff04_nxt <= data2op; data2op_done <= '0';
   when "000101" =>  coeff05_nxt <= data2op; data2op_done <= '0';
   when "000110" =>  coeff06_nxt <= data2op; data2op_done <= '0';
   when "000111" =>  coeff07_nxt <= data2op; data2op_done <= '0';
   when "001000" =>  coeff08_nxt <= data2op; data2op_done <= '0';
```

```
when "001001" =>  coeff09_nxt <= data2op; data2op_done <= '0';
when "001010" =>  coeff10_nxt <= data2op; data2op_done <= '0';
when "001011" =>  coeff11_nxt <= data2op; data2op_done <= '0';
when "001100" =>  coeff12_nxt <= data2op; data2op_done <= '0';
when "001101" =>  coeff13_nxt <= data2op; data2op_done <= '0';
when "001110" =>  coeff14_nxt <= data2op; data2op_done <= '0';
when "001111" =>  coeff15_nxt <= data2op; data2op_done <= '0';
when "010000" =>  coeff16_nxt <= data2op; data2op_done <= '0';
when "010001" =>  coeff17_nxt <= data2op; data2op_done <= '0';
when "010010" =>  coeff18_nxt <= data2op; data2op_done <= '0';
when "010011" =>  coeff19_nxt <= data2op; data2op_done <= '0';
when "010100" =>  coeff20_nxt <= data2op; data2op_done <= '0';
when "010101" =>  coeff21_nxt <= data2op; data2op_done <= '0';
when "010110" =>  coeff22_nxt <= data2op; data2op_done <= '0';
when "010111" =>  coeff23_nxt <= data2op; data2op_done <= '0';
when "011000" =>  coeff24_nxt <= data2op; data2op_done <= '0';
when "011001" =>  coeff25_nxt <= data2op; data2op_done <= '0';
when "011010" =>  coeff26_nxt <= data2op; data2op_done <= '0';
when "011011" =>  coeff27_nxt <= data2op; data2op_done <= '0';
when "011100" =>  coeff28_nxt <= data2op; data2op_done <= '0';
when "011101" =>  coeff29_nxt <= data2op; data2op_done <= '0';
when "011110" =>  coeff30_nxt <= data2op; data2op_done <= '0';
when "011111" =>  coeff31_nxt <= data2op; data2op_done <= '0';
when "100000" =>  coeff32_nxt <= data2op; data2op_done <= '0';

when "100001" =>  input01_nxt <= data2op; data2op_done <= '0';
when "100010" =>  input02_nxt <= data2op; data2op_done <= '0';
when "100011" =>  input03_nxt <= data2op; data2op_done <= '0';
when "100100" =>  input04_nxt <= data2op; data2op_done <= '0';
when "100101" =>  input05_nxt <= data2op; data2op_done <= '0';
when "100110" =>  input06_nxt <= data2op; data2op_done <= '0';
when "100111" =>  input07_nxt <= data2op; data2op_done <= '0';
when "101000" =>  input08_nxt <= data2op; data2op_done <= '1';
when others => input_test <= (others => '0'); data2op_done <= '1';
```

```
        end case;

end process;



end Behavioral;
```

6.9 VHDL code for “store_result”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_signed.all;


entity store_result is
  Port (
        clk, reset      : in std_logic;

        store_en        : in std_logic;
        result1_2store  : in std_logic_vector(18 downto 0);
        result2_2store  : in std_logic_vector(18 downto 0);
        result3_2store  : in std_logic_vector(18 downto 0);
        result4_2store  : in std_logic_vector(18 downto 0);

        store_done      : out std_logic
  );
end store_result;

architecture Behavioral of store_result is

component SRAM_SP_WRAPPER
  port (
    ClkxCI  : in  std_logic;
```

```
    CSxSI   : in  std_logic;            -- Active Low
    WExSI   : in  std_logic;            --Active Low
    AddrxDI : in  std_logic_vector (7 downto 0);
    RYxSO   : out std_logic;
    DataxDI : in  std_logic_vector (31 downto 0);
    DataxDO : out std_logic_vector (31 downto 0)
    );
end component;


    type state_type is (s_initial, s_recieve, s_send1, s_send2, s_send3, s_send4);
    signal state_reg, state_nxt : state_type;

    signal address      : std_logic_vector (7 downto 0);-- := (others => '0');
    signal address_nxt  : std_logic_vector (7 downto 0) := (others => '0');
    signal result       : std_logic_vector(31 downto 0);
    signal result_nxt   : std_logic_vector(31 downto 0);


    signal result1  : std_logic_vector (18 downto 0);
    signal result2  : std_logic_vector (18 downto 0);
    signal result3  : std_logic_vector (18 downto 0);
    signal result4  : std_logic_vector (18 downto 0);


    signal address_out  : std_logic_vector(7 downto 0);
    signal result_out   : std_logic_vector(31 downto 0);

    signal RY_ram       : std_logic;
    signal data_out     : std_logic_vector(31 downto 0);

begin

address_out <= address;
```

```
result_out <= result;

Ram_store: SRAM_SP_WRAPPER
port map(
   ClkxCI          => clk          ,
   CSxSI           => '0'          , -- Active Low --only write in this module
   WExSI           => '0'          , -- Active Low
   AddrxDI         => address_out  ,
   RYxSO           => RY_ram       ,
   DataxDI         => result_out   ,
   DataxDO         => data_out
   );

process (clk, reset)
begin
   if reset = '1' then
      state_reg <= s_initial;
   elsif (clk'event and clk = '1') then
      state_reg <= state_nxt;
   end if;
end process;

process (clk, reset)
begin
   if reset = '1' then
      address <= (others => '0');
      result <= (others => '0');
   elsif (clk'event and clk = '1') then
      address <= address_nxt;
      result <= result_nxt;
   end if;
end process;

process(state_reg, store_en, address)
```

```
begin
    address_nxt <= address;
    result_nxt <= (others => '0');
    case state_reg is
        when s_initial =>
            store_done <= '0';
            result_nxt <= (others => '0');
            if store_en = '1' then
                state_nxt <= s_recieve;
            else
                state_nxt <= s_initial;
            end if;


        when s_recieve =>
            store_done <= '0';
            --address_nxt <= address + 1;
            state_nxt <= s_send1;

        when s_send1 =>
            store_done <= '0';
            address_nxt <= address + 1;
            result_nxt <= "0000000000000" & result1;
            state_nxt <= s_send2;

        when s_send2 =>
            store_done <= '0';
            address_nxt <= address + 1;
            result_nxt <= "0000000000000" & result2;
            state_nxt <= s_send3;

        when s_send3 =>
            store_done <= '0';
            address_nxt <= address + 1;
```

```
            result_nxt <= "0000000000000" & result3;
            state_nxt <= s_send4;

        when s_send4 =>
            store_done <= '1';
            address_nxt <= address + 1;
            result_nxt <= "0000000000000" & result4;
            state_nxt <= s_initial;

    end case;

end process;

result1 <= result1_2store;
result2 <= result2_2store;
result3 <= result3_2store;
result4 <= result4_2store;



end Behavioral;
```

6.10 VHDL code for “max_value”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_signed.all;


entity comparison is
  Port (
    clk, reset      : in std_logic;
    max_en          : in std_logic;
    --compare_done    : in std_logic;
```

```
    compare_out     : in std_logic_vector(18 downto 0);

    compare_result  : out std_logic_vector(18 downto 0)

  );
end comparison;

architecture Behavioral of comparison is
    signal reg      : std_logic_vector(18 downto 0) := (others => '0');
    signal reg_nxt  : std_logic_vector(18 downto 0);-- := (others => '0');

    signal compare_result_nxt : std_logic_vector(18 downto 0);

    type state_type is (s_compare, s_send);
    signal state_reg, state_nxt : state_type;

begin

process (clk, reset)
begin
    if reset = '1' then
        state_reg <= s_compare;
    elsif (clk'event and clk = '1') then
        state_reg <= state_nxt;
    end if;
end process;

process (clk, reset)
begin
    if reset = '1' then
        reg <= (others => '0');
        compare_result <= (others => '0');
    elsif (clk'event and clk = '1') then
        reg <= reg_nxt;
```

```
        compare_result <= compare_result_nxt;
    end if;
end process;

state_machine: process (state_reg, max_en, reg)
begin
    compare_result_nxt <= (others => '0');
    case state_reg is

        when s_compare =>
            if max_en = '1' then
                state_nxt <= s_send;
            else
                state_nxt <= s_compare;
            end if;

        when s_send =>
            compare_result_nxt <= reg;
            state_nxt <= s_compare;
    end case;
end process;

reg_nxt <= compare_out when reg < compare_out else reg;

end Behavioral;
```