Text

Description automatically generated

ETIN 40

IC project 2

Final Report

Hanyu Liu

February 27, 2023

**Abstract**

In this report, we verified the hardware accelerator which was designed in ICP 1. The SRAM which was used in ICP 1 was replaced by distributed memory generator with Xilinx IP. The input data and coefficients will be embedded into the memory as coe file. Therefore, the finite state machine needs to be re-designed to fulfill the new requirements. The verification will be completed by Vivado and a logic analyzer.

Contents

[**1. Introduction** 4](#_Toc128402404)

[**2. Implementation** 5](#_Toc128402405)

[**2.1 Fixed block diagram** 5](#_Toc128402406)

[**2.2 Fixed ASMD charts** 6](#_Toc128402407)

[**2.2.1 Controller** 6](#_Toc128402408)

[**2.2.2 Load coefficient** 8](#_Toc128402409)

[**2.2.3 Load input** 9](#_Toc128402410)

[**3. Verification** 11](#_Toc128402411)

[**4. Appendix** 15](#_Toc128402412)

[**4.1 VHDL code for controller** 15](#_Toc128402413)

[**4.2 VHDL code for load\_coeff** 18](#_Toc128402414)

[**4.3 VHDL code for load\_input** 21](#_Toc128402415)

[**4.4 VHDL code for multiply** 27](#_Toc128402416)

[**4.5 VHDL code for store** 32](#_Toc128402417)

[**4.6 VHDL code for max** 34](#_Toc128402418)

[**4.7 VHDL code for top** 36](#_Toc128402419)

**1. Introduction**

The project aims to verify the IC project 1 hardware accelerator that realizes the multiplication of an input matrix with a size of 14x8 and a coefficient matrix with a size of 8x14. The input data and coefficients are pre-stored in SRAMs with Xilinx IP, respectively. And the results will be stored in another SRAM after calculation. The hardware accelerator is constructed by using VHDL language and verified by a logic analyzer.

To verify the design in IC project 2, the state machine has been changed in modules “controller”, “load\_coeff” and “load\_input”. The block diagram has also been changed accordingly which has shown in the following paragraph. All the verifications are based on the FPGA of Xilinx “xc7a100tcsg324-1”.

**2. Implementation**

In this part, the fixed block diagram and fixed ASMD charts will be introduced respectively. Comparing the ASMD charts in IC project 1, some states were deleted to adapt to the new requirements in IC project 2. A different design will be introduced in this part.

**2.1 Fixed block diagram**

The block diagram of the top module is shown in Figure 2.1. This matrix multiplier can be divided into 9 modules, which are “load\_coeff”, “load\_input”, “multiply”, “store”, “max”, “controller”, and three SRAMs. There are two inputs and one output for the top module to adapt the new design in IC project 2, “start”, “restart” and “max\_out”.

Diagram

Description automatically generated

Figure 2.1: The block diagram of the matrix multiplier

In IC project 2, “SRAM\_coe” and “SRAM\_input”, two SRAMs keep the coefficient matrix and input matrix respectively. Module “load\_input” will generate the address for “SRAM\_input” and then temporarily keep for a row of input matrix by four 16-bit registers. Module “load\_coeff” will generate an address for “SRAM\_coe” and send the data directly to the module “multiply”.

**2.2 Fixed ASMD charts**

To adapt the IC project 2, there are three fixed ASMD charts shown in this part, “controller”, “load\_coeff”, and “load\_input”. In this part, the fixed state machine will be introduced.

**2.2.1 Controller**

The ASMD chart of the “controller” is shown in Figure 2.2.1. There are three states in this module.

Diagram

Description automatically generated

Figure 2.2.1: The ASMD chart of the “controller”

S\_initial:

In this state, all signals are set to default values. Signal “flag1” in IC project 1 has been deleted due to all the data has been stored in SRAMs. There is no need to get the data from the outside. Thus, once the signal “start” is HIGH, the calculation will start immediately.

S\_ld\_input:

In this state, 8 inputs will be loaded from “SRAM\_input” and stored in four 16-bit registers. After that, the signal “ldinput\_done” will be set to HIGH, which indicates that the system is ready to produce a column of results.

S\_op:

The state “s\_op” is where the multiplication and accumulation take place. There is no change compared with the design in IC project 1.

**2.2.2 Load coefficient**

The ASMD chart of “load\_coeff” is shown in Figure 2.2.2. There are three states in this module.

Diagram

Description automatically generated

Figure 2.2.2: The ASMD chart of “load\_coefficient”

In the IC project 1 design, there are four states in module “load\_coeff”. There is no need to load the coefficients from outside into SRAM. All the data has been embedded by coe file. However, the other three states kept making the state machine work well.

**2.2.3 Load input**

The ASMD chart of “load\_input” is shown in Figure 2.2.3. There are five states in this module.

Diagram

Description automatically generated

Figure 2.2.3: The ASMD chart of “load\_input”

S\_initial:

In this state, all variables will be set to default values. There is no change compared to the design with it in IC project 1.

S\_ld\_input\_1:

This is the new state in IC project 2. In this state, the module will send the address to the “SRAM\_input”. After receiving one raw input data, the state machine will return to the initial state.

S\_ld\_input\_2:

This is also the new state in IC project 2. In this state, the system will receive the input data from the SRAM and store them in registers. Then the system will get back to state “S\_ld\_input\_1” to do the next execution.

S\_send2multi\_w1:

This state will wait for a clock cycle to ensure that the coefficients have been received.

S\_send2multi:

In this state, 8 inputs will be sent to the module “multiply” two by two.

**3. Verification**

The multiplication was verified in Vivado, and the simulation results are shown in Figures 3.1 and 3.2. The results are stored in SRAM in order successfully which has been shown in figure 3.3. Figure 3.4 shows the system iterated 14 times due to 14 rows in the input matrix. Figure 3.5 shows the data are the same in both the simulation and logic analyzer.

A picture containing diagram

Description automatically generated

store signal

Result from each multiplier

input

coefficient

1336

812

554

114

480

44

120

11

4

4

189

69

63

23

3

3

204

102

118

2

2

236

53

61

53

61

1

1

Figure 3.1: Simulation result from Vivado

2

10

28

60

Graphical user interface

Description automatically generated

794

794

344

88

86

22

4

4

3

3

362

63

21

39

13

260

126

88

46

44

23

2

51

75

51

75

1

2

1

Figure 3.2: Another group of simulation results from Vivado

A screenshot of a computer

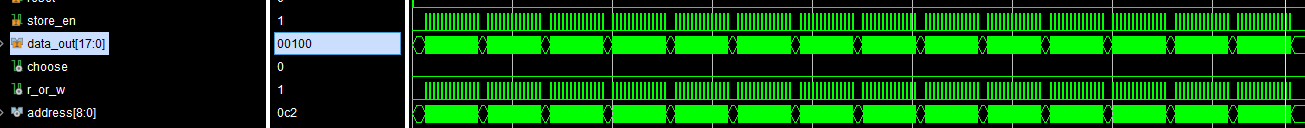
Description automatically generated with medium confidence

556

1136

384

Figure 3.3: The results in decimal stored in memory successfully.



14

13

12

11

10

9

8

7

6

5

4

3

2

1

Figure 3.4: The calculation will iterate 14 times due to 14 rows in the input matrix.

A screen shot of a computer

Description automatically generated with low confidence

0

0

0

1

1

0

1

1

1

0

0

0

A picture containing chart

Description automatically generated

Figure 3.5: Compare the data detected in the logic analyzer with the simulation result.

**4. Appendix**

**4.1 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;

start : in std\_logic;

ld\_input\_done : in std\_logic;

multi\_done : in std\_logic;

ld\_input : out std\_logic;

op\_en : out std\_logic

);

end controller;

architecture Behavioral of controller 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\_ld\_input, s\_op);

signal state\_reg, state\_nxt : state\_type;

signal counter14, counter14\_nxt : std\_logic\_vector(3 downto 0) := (others => '0');

begin

--state contrl--------------------------------

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;

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

process(state\_reg, start, ld\_input\_done, multi\_done, counter14)

begin

ld\_input <= '0';

op\_en <= '0';

counter14\_nxt <= counter14;

case state\_reg is

when s\_initial =>

counter14\_nxt <= (others => '0');

if start = '1' then

state\_nxt <= s\_ld\_input;

else

state\_nxt <= s\_initial;

end if;

when s\_ld\_input =>

ld\_input <= '1';

if ld\_input\_done = '1' then

state\_nxt <= s\_op;

else

state\_nxt <= s\_ld\_input;

end if;

when s\_op =>

if multi\_done = '1' then

if counter14 = "1101" then

counter14\_nxt <= (others => '0');

state\_nxt <= s\_initial;

op\_en <= '0';

else

counter14\_nxt <= counter14 + 1;

state\_nxt <= s\_ld\_input;

op\_en <= '0';

end if;

else

op\_en <= '1';

counter14\_nxt <= counter14;

state\_nxt <= s\_op;

end if;

end case;

end process;

counter\_14: FF

generic map(N => 4)

port map( D =>counter14\_nxt,

Q =>counter14,

clk =>clk,

reset =>reset

);

end Behavioral;

**4.2 VHDL code for load\_coeff**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity load\_coeff is

Port (

clk, reset : in std\_logic;

--signal from controller

op\_en : in std\_logic;

--control signal to multiply

multi\_en : out std\_logic;

--coeff to multiply

data\_coeff : out std\_logic\_vector(15 downto 0)

);

end load\_coeff;

architecture Behavioral of load\_coeff is

component SRAM\_coe

port (

clk : in std\_logic;

we : in std\_logic;

a : in std\_logic\_vector (6 downto 0);

d : in std\_logic\_vector (15 downto 0);

qspo : out std\_logic\_vector (15 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;

--SRAM---------------------------------------------

signal choose : std\_logic;

signal r\_or\_w : std\_logic;

signal address : std\_logic\_vector(6 downto 0);

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

type state\_type is (s\_initial, s\_op, s\_send2multi);

signal state\_reg, state\_nxt : state\_type;

signal counter\_1, counter\_1\_nxt : std\_logic\_vector(5 downto 0) := (others => '0');

signal coeff\_32 : std\_logic\_vector(15 downto 0);

signal data\_coeff\_32 : std\_logic\_vector(15 downto 0);

signal coeff : std\_logic\_vector(15 downto 0);

begin

--SRAM bits transfer----------------------

coeff\_32 <= coeff;

data\_coeff <= data\_coeff\_32(15 downto 0);

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

Ram\_coeff: SRAM\_coe

port map(

clk => clk,

we => r\_or\_w,

a => address,

d => coeff\_32,

qspo => data\_coeff\_32

);

--state contrl--------------------------------

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;

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

process(state\_reg, op\_en, counter\_1)

begin

--SRAM------------------------------

choose <= '1';

r\_or\_w <= '0';--read

address <= "0" & counter\_1;

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

counter\_1\_nxt <= (others => '0');

multi\_en <= '0';

case state\_reg is

when s\_initial =>

if op\_en = '1' then

state\_nxt <= s\_op;

else

state\_nxt <= s\_initial;

end if;

when s\_op =>

choose <= '0';

r\_or\_w <= '0'; --read

address <= "0" & counter\_1;

counter\_1\_nxt <= counter\_1;

if counter\_1 = "111000" then --counter = 56 (address = 0 -55)

state\_nxt <= s\_initial;

elsif counter\_1 = "000000" then

multi\_en <= '1';

state\_nxt <= s\_send2multi;

else

multi\_en <= '0';

state\_nxt <= s\_send2multi;

end if;

when s\_send2multi =>

choose <= '0';

r\_or\_w <= '0'; --read

address <= "0" & counter\_1;

counter\_1\_nxt <= counter\_1 + 1;

state\_nxt <= s\_op;

end case;

end process;

counter1: FF

generic map(N => 6)

port map( D =>counter\_1\_nxt,

Q =>counter\_1,

clk =>clk,

reset =>reset

);

end Behavioral;

**4.3 VHDL code for load\_input**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity load\_input is

Port (

clk, reset : in std\_logic;

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

ld\_input : in std\_logic;

ld\_input\_done : out std\_logic;--feedback to controller

--signal from controller

op\_en : in std\_logic;

start\_ld\_input : out std\_logic;

data\_input : out std\_logic\_vector(15 downto 0)

);

end load\_input;

architecture Behavioral of load\_input is

component SRAM\_input

port (

clk : in std\_logic; --Active Low

we : in std\_logic;

a : in std\_logic\_vector (6 downto 0);

d : in std\_logic\_vector (15 downto 0);

qspo : out std\_logic\_vector (15 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;

--SRAM---------------------------------------------

signal choose : std\_logic;

signal r\_or\_w : std\_logic; -- Active Low (reand & write) --write '0' --read '1'

signal address : std\_logic\_vector(6 downto 0);

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

type state\_type is (s\_initial, s\_ld\_input\_1, s\_ld\_input\_2, s\_send2multi, s\_send2multi\_w1);

signal state\_reg, state\_nxt : state\_type;

signal reg\_1, reg\_1\_nxt : std\_logic\_vector(15 downto 0);

signal reg\_2, reg\_2\_nxt : std\_logic\_vector(15 downto 0);

signal reg\_3, reg\_3\_nxt : std\_logic\_vector(15 downto 0);

signal reg\_4, reg\_4\_nxt : std\_logic\_vector(15 downto 0);

--enable write in reg

signal flag1 : std\_logic;

--enable send data

signal flag2 : std\_logic;

signal hold, hold\_nxt : std\_logic\_vector(0 downto 0) := (others => '0');

--control load

signal counter1, counter1\_nxt : std\_logic\_vector(2 downto 0) := (others => '0');

--control send

signal counter2, counter2\_nxt : std\_logic\_vector(1 downto 0) := (others => '0');

--control the loop will be executed 14 times

signal counter3, counter3\_nxt : std\_logic\_vector(3 downto 0) := (others => '0');

signal counter4, counter4\_nxt : std\_logic\_vector(3 downto 0) := (others => '0');

signal input\_32 : std\_logic\_vector(15 downto 0);

signal input : std\_logic\_vector(15 downto 0);

begin

Ram\_input: SRAM\_input

port map(

clk => clk,

we => r\_or\_w,

a => address,

d => input\_32,

qspo => input

);

--state contrl----------------------------------------------

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;

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

process(state\_reg, ld\_input, op\_en, counter1, counter2, counter3, counter4, hold)

begin

start\_ld\_input <= '0';

ld\_input\_done <= '0';

counter1\_nxt <= (others => '0');

counter2\_nxt <= (others => '0');

counter3\_nxt <= (others => '0');

counter4\_nxt <= counter4;

flag1 <= '0';

flag2 <= '0';

hold\_nxt <= (others => '0');

r\_or\_w <= '0'; --always read

address <= "000" & counter4;

case state\_reg is

when s\_initial =>

if ld\_input = '1' and op\_en = '0' then

start\_ld\_input <= '1';--give signal to outside

state\_nxt <= s\_ld\_input\_1;

elsif ld\_input = '0' and op\_en = '1' then

state\_nxt <= s\_send2multi\_w1;

else

state\_nxt <= s\_initial;

end if;

when s\_ld\_input\_1 =>

flag1 <= '1';

if counter1 > "011" then

start\_ld\_input <= '1';

ld\_input\_done <= '1';

counter1\_nxt <= (others => '0');

state\_nxt <= s\_initial;

else

start\_ld\_input <= '1';

ld\_input\_done <= '0';

counter1\_nxt <= counter1;

state\_nxt <= s\_ld\_input\_2;

end if;

when s\_ld\_input\_2 =>

flag1 <= '1';

counter4\_nxt <= counter4 + 1;

counter1\_nxt <= counter1 + 1;

state\_nxt <= s\_ld\_input\_1;

when s\_send2multi\_w1 =>

state\_nxt <= s\_send2multi;

when s\_send2multi =>

flag2 <= '1';

if hold = "0" then

hold\_nxt <= hold + 1;

state\_nxt <= s\_send2multi;

counter2\_nxt <= counter2;

counter3\_nxt <= counter3;

else

hold\_nxt <= (others => '0');

if counter3 = "1101" and counter2 = "11" then

counter2\_nxt <= (others => '0');

counter3\_nxt <= (others => '0');

state\_nxt <= s\_initial;

elsif counter3 < "1101" and counter2 = "11" then

counter3\_nxt <= counter3 + 1;

counter2\_nxt <= (others => '0');

state\_nxt <= s\_send2multi;

else

counter3\_nxt <= counter3;

counter2\_nxt <= counter2 + 1;

state\_nxt <= s\_send2multi;

end if;

end if;

end case;

end process;

reg\_1\_nxt <= input when counter1 = "000" and flag1 = '1' else reg\_1;

reg\_2\_nxt <= input when counter1 = "001" and flag1 = '1' else reg\_2;

reg\_3\_nxt <= input when counter1 = "010" and flag1 = '1' else reg\_3;

reg\_4\_nxt <= input when counter1 = "011" and flag1 = '1' else reg\_4;

--Send the data --------------------------------------------

data\_input <= reg\_1 when counter2 = "00" and flag2 = '1' else

reg\_2 when counter2 = "01" and flag2 = '1' else

reg\_3 when counter2 = "10" and flag2 = '1' else

reg\_4 when counter2 = "11" and flag2 = '1' else

(others => '0');

--Flip Flop ------------------------------------------------

reg\_01: FF

generic map(N => 16)

port map( D =>reg\_1\_nxt,

Q =>reg\_1,

clk =>clk,

reset =>reset

);

reg\_02: FF

generic map(N => 16)

port map( D =>reg\_2\_nxt,

Q =>reg\_2,

clk =>clk,

reset =>reset

);

reg\_03: FF

generic map(N => 16)

port map( D =>reg\_3\_nxt,

Q =>reg\_3,

clk =>clk,

reset =>reset

);

reg\_04: FF

generic map(N => 16)

port map( D =>reg\_4\_nxt,

Q =>reg\_4,

clk =>clk,

reset =>reset

);

counter\_01: FF

generic map(N => 3)

port map( D =>counter1\_nxt,

Q =>counter1,

clk =>clk,

reset =>reset

);

counter\_02: FF

generic map(N => 2)

port map( D =>counter2\_nxt,

Q =>counter2,

clk =>clk,

reset =>reset

);

counter\_03: FF

generic map(N => 4)

port map( D =>counter3\_nxt,

Q =>counter3,

clk =>clk,

reset =>reset

);

counter\_04: FF

generic map(N => 4)

port map( D =>counter4\_nxt,

Q =>counter4,

clk =>clk,

reset =>reset

);

hold\_time : FF

generic map(N => 1)

port map( D =>hold\_nxt,

Q =>hold,

clk =>clk,

reset =>reset

);

end Behavioral;

**4.4 VHDL code for multiply**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity multiply is

Port (

clk, reset : in std\_logic;

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

--signal from load\_coeff

multi\_en : in std\_logic;

--data in

data\_input : in std\_logic\_vector(15 downto 0);

data\_coeff : in std\_logic\_vector(15 downto 0);

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

--ctrl out

multi\_done : out std\_logic;

store\_en : out std\_logic;

max\_en : out std\_logic;

--data out

data\_out : out std\_logic\_vector(17 downto 0)

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

);

end multiply;

architecture Behavioral of multiply 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\_multi, s\_add, s\_send);--, s\_wait);

signal state\_reg, state\_nxt : state\_type;

signal input\_1, input\_2 : std\_logic\_vector(7 downto 0);

signal coeff\_1, coeff\_2 : std\_logic\_vector(6 downto 0);

signal result\_1, result\_2 : std\_logic\_vector(17 downto 0);

--the matrix is [14\*8]\*[8\*14], when counting 111 in binary means one number is done.

signal counter\_8, counter\_8\_nxt : std\_logic\_vector(1 downto 0) := (others => '0');

signal counter\_14, counter\_14\_nxt : std\_logic\_vector(3 downto 0) := (others => '0');

signal counter\_6, counter\_6\_nxt : std\_logic\_vector(2 downto 0) := (others => '0');

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

signal data, data\_nxt : std\_logic\_vector(17 downto 0);

signal store, store\_nxt : std\_logic\_vector(0 downto 0);

begin

--state contrl--------------------------------

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;

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

process(state\_reg, multi\_en, counter\_8, data, result\_1, result\_2, counter\_14, counter\_6)

begin

multi\_done <= '0';

data\_nxt <= (others => '0');

counter\_8\_nxt <= (others => '0');

counter\_14\_nxt <= (others => '0');

counter\_6\_nxt <= (others => '0');

store\_nxt <= "0";

input\_1 <= (others => '0');

input\_2 <= (others => '0');

coeff\_1 <= (others => '0');

coeff\_2 <= (others => '0');

case state\_reg is

when s\_initial =>

if multi\_en = '1' then

state\_nxt <= s\_multi;

else

state\_nxt <= s\_initial;

end if;

when s\_multi =>

input\_1 <= data\_input(7 downto 0);--the input has 8 bits

input\_2 <= data\_input(15 downto 8);

coeff\_1 <= data\_coeff(6 downto 0);--the coeff only has 7 bits

coeff\_2 <= data\_coeff(14 downto 8);

if counter\_8 = "00" then

data\_nxt <= (others => '0');

else

data\_nxt <= data;

end if;

counter\_8\_nxt <= counter\_8;

counter\_14\_nxt <= counter\_14;

state\_nxt <= s\_add;

when s\_add =>

data\_nxt <= result\_1 + result\_2 + data;

input\_1 <= data\_input(7 downto 0);--the input has 8 bits

input\_2 <= data\_input(15 downto 8);

coeff\_1 <= data\_coeff(6 downto 0);--the coeff only has 7 bits

coeff\_2 <= data\_coeff(14 downto 8);

if counter\_8 = "11" then

counter\_8\_nxt <= (others => '0');

store\_nxt <= "1";--signal to store

if counter\_14 = "1101" then

counter\_14\_nxt <= (others => '0');

state\_nxt <= s\_send;

else

counter\_14\_nxt <= counter\_14 + 1;

state\_nxt <= s\_multi;

end if;

else

store\_nxt <= "0";--signal to store

counter\_14\_nxt <= counter\_14;

counter\_8\_nxt <= counter\_8 + 1;

state\_nxt <= s\_multi;

end if;

when s\_send =>

multi\_done <= '1';--feedback to controller & average

state\_nxt <= s\_initial;

end case;

end process;

--Computing part with two multipliers-----------------------

result\_1 <= input\_1 \* coeff\_1 + "000000000000000000"; --to make they have the same digits

result\_2 <= input\_2 \* coeff\_2 + "000000000000000000";

--Send data-------------------------------------------------

store\_en <= store(0);

max\_en <= store(0);

data\_out <= data when store = "1" else (others => '0');

--Flip Flop-------------------------------------------------

counter1: FF

generic map(N => 2)

port map( D =>counter\_8\_nxt,

Q =>counter\_8,

clk =>clk,

reset =>reset

);

counter2: FF

generic map(N => 4)

port map( D =>counter\_14\_nxt,

Q =>counter\_14,

clk =>clk,

reset =>reset

);

counter3: FF

generic map(N => 3)

port map( D =>counter\_6\_nxt,

Q =>counter\_6,

clk =>clk,

reset =>reset

);

add: FF

generic map(N => 18)

port map( D =>data\_nxt,

Q =>data,

clk =>clk,

reset =>reset

);

send\_data: FF

generic map(N => 1)

port map( D =>store\_nxt,

Q =>store,

clk =>clk,

reset =>reset

);

end Behavioral;

**4.5 VHDL code for store**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity store is

Port (

clk, reset : in std\_logic;

store\_en : in std\_logic;

data\_out : in std\_logic\_vector(17 downto 0)

);

end store;

architecture Behavioral of store is

component SRAM\_store

port (

clk : in std\_logic; --Active Low

we : in std\_logic;

a : in std\_logic\_vector (8 downto 0);

d : in std\_logic\_vector (31 downto 0);

qspo : 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;

--SRAM---------------------------------------------

signal choose : std\_logic;

signal r\_or\_w : std\_logic;

signal address : std\_logic\_vector(8 downto 0) := (others => '0');

signal RY\_ram : std\_logic;

signal sram\_out : std\_logic\_vector(31 downto 0);

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

signal address\_nxt : std\_logic\_vector(8 downto 0);

signal store\_data\_32 : std\_logic\_vector(31 downto 0);

begin

--SRAM bits transfer----------------------

store\_data\_32 <= "00000000000000" & data\_out;

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

Ram\_store: SRAM\_store

port map(

clk => clk,

we => r\_or\_w,

a => address,

d => store\_data\_32,

qspo => sram\_out

);

address\_nxt <= address + 1 when store\_en = '1' else address;

r\_or\_w <= '1' when store\_en = '1' else '0';

choose <= '0';

data\_address: FF

generic map(N => 9)

port map( D =>address\_nxt,

Q =>address,

clk =>clk,

reset =>reset

);

end Behavioral;

**4.6 VHDL code for max**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity max is

Port (

clk, reset : in std\_logic;

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

--control signal from multiply

max\_en : in std\_logic;

--data from multiply

data\_out : in std\_logic\_vector(17 downto 0);

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

--find the maximum data

max\_out : out std\_logic\_vector(17 downto 0);

clk\_out : out std\_logic

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

);

end max;

architecture Behavioral of max 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;

signal count, count\_nxt : std\_logic\_vector(6 downto 0);

signal data\_max, data\_max\_nxt : std\_logic\_vector(17 downto 0) := (others => '0');

begin

max\_out <= data\_max;

clk\_out <= clk;

data\_max\_nxt <= data\_out when data\_max < data\_out else data\_max;

--Flip Flop-------------------------------------------------

compare\_data: FF

generic map(N => 18)

port map( D =>data\_max\_nxt,

Q =>data\_max,

clk =>clk,

reset =>reset

);

end Behavioral;

**4.7 VHDL code for top**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity top is

Port (

--input

clki : in std\_logic;

reseti : in std\_logic;

starti : in std\_logic;

--output

start\_ld\_inputo : out std\_logic;

start\_ld\_coeffo : out std\_logic;

max\_outo : out std\_logic\_vector(17 downto 0);

clk\_out : out std\_logic

);

end top;

architecture Behavioral of top is

component controller is

Port (

clk, reset : in std\_logic;

start : in std\_logic;

ld\_input\_done : in std\_logic;

multi\_done : in std\_logic;

ld\_input : out std\_logic;

op\_en : out std\_logic

);

end component;

component load\_coeff is

Port (

clk, reset : in std\_logic;

op\_en : in std\_logic;

multi\_en : out std\_logic;

data\_coeff : out std\_logic\_vector(15 downto 0)

);

end component;

component load\_input is

Port (

clk, reset : in std\_logic;

ld\_input : in std\_logic;

ld\_input\_done : out std\_logic;--feedback to controller

op\_en : in std\_logic;

start\_ld\_input : out std\_logic;

data\_input : out std\_logic\_vector(15 downto 0)

);

end component;

component multiply is

Port (

clk, reset : in std\_logic;

multi\_en : in std\_logic;

data\_input : in std\_logic\_vector(15 downto 0);

data\_coeff : in std\_logic\_vector(15 downto 0);

multi\_done : out std\_logic;

store\_en : out std\_logic;

max\_en : out std\_logic;

data\_out : out std\_logic\_vector(17 downto 0)

);

end component;

component store is

Port (

clk, reset : in std\_logic;

store\_en : in std\_logic;

data\_out : in std\_logic\_vector(17 downto 0)

);

end component;

component max is

Port (

clk, reset : in std\_logic;

max\_en : in std\_logic;

data\_out : in std\_logic\_vector(17 downto 0);

max\_out : out std\_logic\_vector(17 downto 0);

clk\_out : out std\_logic

);

end component;

--controller

signal ld\_input\_done : std\_logic;

signal multi\_done : std\_logic;

signal ld\_input : std\_logic;

signal op\_en : std\_logic;

--ld\_coeff

signal multi\_en : std\_logic;

signal data\_coeff : std\_logic\_vector(15 downto 0);

--ld\_input

signal data\_input : std\_logic\_vector(15 downto 0);

--multiply

signal store\_en : std\_logic;

signal data\_out : std\_logic\_vector(17 downto 0);

--store

--max

signal max\_en : std\_logic;

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

begin

--clk\_out <= clki;

controller\_part: controller

port map(

clk => clki ,

reset => reseti ,

start => starti ,

ld\_input\_done => ld\_input\_done ,

multi\_done => multi\_done ,

ld\_input => ld\_input ,

op\_en => op\_en

);

coeff\_part: load\_coeff

port map(

clk => clki ,

reset => reseti ,

op\_en => op\_en ,

multi\_en => multi\_en ,

data\_coeff => data\_coeff

);

input\_part: load\_input

port map(

clk => clki ,

reset => reseti ,

ld\_input => ld\_input ,

ld\_input\_done => ld\_input\_done ,

op\_en => op\_en ,

start\_ld\_input => start\_ld\_inputo ,

data\_input => data\_input

);

multiply\_part: multiply

port map(

clk => clki,

reset => reseti,

multi\_en => multi\_en,

data\_input => data\_input,

data\_coeff => data\_coeff,

multi\_done => multi\_done,

store\_en => store\_en,

max\_en => max\_en,

data\_out => data\_out

);

store\_part: store

port map(

clk => clki,

reset => reseti,

store\_en => store\_en,

data\_out => data\_out

);

max\_part: max

port map(

clk => clki,

reset => reseti,

max\_en => max\_en,

data\_out => data\_out,

max\_out => max\_outo,

clk\_out => clk\_out

);

end Behavioral;