VHDL Final Project

Students Exam ID: Y3852394 & Y3858168

CONTENTS

1. Script questions

Q1. (Datapath): What is the maximum width (in terms of bits) of the coefficients of C? Which generics are relevant? Consider the multiplication operator and the sequence of sums to be performed. The result should be expressed as a function of the relevant generic values.

- According to the function package provided, size (N) function is used in calculating how many bits needed to represent number N, however, $\log_2$ (N) function is used in calculating how many bits needed to have maximum N locations. So it is easy to get a relationship for size (N) = $\log_2(N + 1)$.

- There are two formulae to calculate the width of the coefficients of C:

    - To acquire the formula by calculating the maximum decimal number in matrix C: (defined in VHDL by size function from DigEng.vhd package)

    $$(\text{Size } ((2^{\text{data\_size}-1})^2 * M)) + 1$$

    - To acquire the formula by considering how many addition and multiplication takes: (defined in VHDL by $\log_2$ function from DigEng.vhd package)

    $$\text{Log}_2 (M-1) + (\text{data\_size}*2)$$

- Following the specifications of the system, there are two cases tested in the self-checking testbench. When M = 3, N = 5, H = 4 and data_size = 5, the maximum width should be 11 bits. When M = 17, N = 8, H = 4 and data_size = 4, the maximum width should be 12 bits.

- The value of coefficient of C (example when calculating c_{11}) is determined by the formula:

$$a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$

- The value of coefficient of C is globally defined by the formula:

$$C(x,y) = \sum_{m}^{M} A(x,m) * B(m,y)$$

- Relevant generics:

    - data_size: Defines how many bits are used to represent values inside matrix A and B. Due to a multiplication between two numbers, the length of product will be twice larger than the length of multipliers, which are the values stored in matrix A and B.

    - M: This represents how many times that the addition takes for calculating one item of matrix C. Because there might be an overflow when adding two binary numbers, which influences the width of modulus of C, M should be considered.

- Derivation to get the formula:

    - Derivation based on calculating the maximum value in C:

        - $2^{\text{data\_size}-1}$: In this matrix multiplication, considering about coefficients of

every matrix can be positive and negative, we use signed number to represent. There are addition and multiplication here and we consider the multiplication firstly. To calculate the maximum product, because we have set how many bits used to represent a number, we can acquire the maximum negative number is (-2^{data_size-1}). However, we need the largest number of the product, which is positive.

- $(2^{data\_size}-1)^2$: In terms of the largest number of the product, we need two negative number multiplied with each other, which gives me $(-2^{data\_size-1}) * (-2^{data\_size-1})$, the same as $(2^{data\_size}-1)^2$.

- Size $((2^{data\_size-1})^2 * M)$: Because the maximum coefficient of matrix C is required, we assume every product from two cells is $(2^{data\_size}-1)^2$. Due to M time's addition, the maximum coefficient in matrix C should be $(2^{data\_size-1})^2 * M$. To calculate how many bits to represent a number, we use size function.

- (Size $((2^{data\_size-1})^2 * M)) + 1$: To show signed number which is a 2's complement number, the result from size function should be added 1.

- Derivation based on the number of addition and multiplication it takes:

  - Multiplication: The matrix coefficients can be set positive and negative so that signed values need to be used. If we want the maximum absolute value of a single coefficient, both the A coefficient and the B coefficient could have the maximum absolute value, which are both maximum negative numbers. The product takes data_size*2 including the signed bit as the maximum width of product.

  - Addition: Adding two binary numbers causes overflow but it does not happen every time adding two numbers, which follows a rule. For instance, when there is one adder, there could be one overflow, when there is 2 or 3 adders, there could be two overflows and when there is 4, 5, 6, 7 adders, there could be three overflows. As shown in the example, $\log_2$ (how many addition we do) should be bits of overflow. Besides, $\log_2 7$ and $\log_2 4$ can be set to the same value when designing the counter in VHDL.

Q2 (Datapath): What is the size (width and depth) of the memories that store A, B, and C, as a function of M, N, H, and the size of the data? What is then the width of the address bus for each of the memories as a function of M, N, H? What are the input matrices that you will use to test the system?

- Size of memories and width of the address bus:

  - As shown in the DigEng.vhd package, it is suitable to define width of the address bus by size function and width of coefficient of C by log function or size function when considering to represent maximum value. Since width*height gives how many locations we need in C matrix, so we can use $\log_2(H*N)$ or size(H*N-1) (from DigEng.vhd package) to get width of the address bus. It is the same principle used to calculate width of the address bus for ROM A and B.

  - ROM_A: **Width:** data_size

Depth: H*M
Width of the address bus: size (H*M-1) (defined in VHDL by size function from DigEng.vhd package)

- ROM_B: **Width:** data_size
  Depth: N*M
  Width of the address bus: size (N*M-1) (defined in VHDL by size function from DigEng.vhd package)

- RAM_C: **Width:** (Size $((2^{data\_size-1})^2 * M)) + 1$
  Depth: H*N
  Width of the address bus: size (H*N-1) (defined in VHDL by size function from DigEng.vhd package)

- Input matrices used to test the system

  - For M = 3, N = 5, H = 4 and data_size = 5, the input matrix will be:

    - Considering about the range that a 5-bit signed number can represent, it is from -16 to 15. The size of matrix A will be 4*3, matrix B will be 3*5 and matrix C will be 4*5.

    - Matrix A:

-16	-16	-16
15	15	15
-6	11	10
-8	0	1

    - Matrix B:

-16	15	-1	-3	-6
-16	15	-2	0	10
-16	15	0	8	-3

    - Expected matrix C (the first item should be largest possible positive value and second item should be largest possible negative value):

768	-720	48	-80	-16
-720	675	-45	75	15
-240	225	-16	98	116
112	-105	8	32	45

  - For M = 17, N = 8, H = 4 and data_size = 4, the input matrix will be:

    - Considering about the range that a 4-bit signed number can represent, it is from -8 to 7. The size of matrix A will be 4*17, matrix B will be 17*8 and matrix C will be 4*8. As tested matrix will sue the first case so that this matrix is not

given as follows. Just declare the size of matrix and range of values.

Q3 (Control logic): How are the counters used for address generation related to each other? When should each counter reset to 0 (beyond the global reset)? What are the sizes (widths) of the counters? How are the addresses computed from the counter values?

- To design the circuits, the order of calculation is set as calculating the first row of matrix C and then transfer to the next row. It is also possible to calculate the first column and then transfer to the next column but in this designed, we use the first way.

- Operation that counters do for address generation:

    - As shown in the structure of control logic, there are three counters connecting with FSM, which gives enable signals for three counters. According to the equation to calculate the first item of matrix C, which is $a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$, it can be found that M is counted but H and N are constant until one item is calculated as well as being stored into the RAM. After calculating one item, H and N will count to transfer to the next row and column in matrix A and B respectively. In this way, the addresses as an enable signal of two ROMs can command ROMs to load data correctly.

    - M counter indicates the current position in the column of matrix A and row of matrix B. It will count for every coefficient in matrix C.

    - N counter indicates the current position in the column of matrix B. It will count every time M counter counts to the maximum, meanwhile, H counter does not count. Because we calculate the first row and then transfer the next row, N counter will firstly count and then be set back to 0 to process the next row.

    - H counter indicates the current position in the row of matrix A. It will count every time M counter counts to the maximum, meanwhile, H counter counts to the maximum.

- When each counter resets to 0

    - M counter should be reset to 0 every time one coefficient of C is calculated. At that point, M counter has counted to the maximum value of M could be.

    - H counter should never be reset to 0 because we calculate row by row. It remains at its maximum value until the circuits is globally reset.

    - N counter should be set back to 0 when all coefficients of matrix C in a row are worked out unless the H counter has also reached its maximum values, which means the matrix C is filling-in. At this moment, it reaches its maximum value.

- Sizes of the counters

    - Since M, N, H mean how many addresses should be set in counters, we use $\log_2$ function from DigEng.vhd package to set bits we use to have M, N, H locations in three counters

    - M counter : $\log_2(M)$ (example: M = 3, width of M counter = 2)

    - N counter : $\log_2(N)$ (example: N = 5, width of N counter = 3)

- H counter : $\log_2(H)$ (example: H = 4, width of H counter = 2)

- How addresses computed from the counter values

  - The two matrices must be stored in row-first order with the memories. Because this circuits is designed to write values in RAM C row by row, that H reaches its terminal value shows that all coefficients has been worked out. Multiplier from ROM A has the same row serial number as column serial number of multiplier in B for every product. In the design, we use resize function to transfer the address that H, M, N counter counts to corresponding addresses shown in the form above.

  - ROM A controlled by counter H and counter M (H = 4 and M = 3 as an example)

Memory locations in ROM A	Coefficient	Counter H	Counter M
0 (0000)	A_{11}	0 (00)	0 (00)
1 (0001)	A_{12}	0	1 (01)
2 (0010)	A_{13}	0	2 (10)
3 (0011)	A_{21}	1 (01)	0
4 (0100)	A_{22}	1	1
5 (0101)	A_{23}	1	2
6 (0110)	A_{31}	2 (10)	0
7 (0111)	A_{32}	2	1
8 (1000)	A_{33}	2	2
9 (1001)	A_{41}	3 (11)	0
10 (1010)	A_{42}	3	1
11 (1011)	A_{43}	3	2

  - ROM B controlled by counter N and counter M (N = 5 and M = 3 as an example)

Memory locations in ROM B	Coefficient	Counter N	Counter M
0 (0000)	B_{11}	0 (000)	0 (00)
1 (0001)	B_{12}	1 (001)	0
2 (0010)	B_{13}	2 (010)	0
3 (0011)	B_{14}	3 (011)	0
4 (0100)	B_{15}	4 (100)	0
5 (0101)	B_{21}	0	1(01)
6 (0110)	B_{22}	1	1
7 (0111)	B_{23}	2	1
8 (1000)	B_{24}	3	1
9(1001)	B_{25}	4	1
10 (1010)	B_{31}	0	2(10)
11 (1011)	B_{32}	1	2
12 (1100)	B_{33}	2	2
13 (1101)	B_{34}	3	2
14 (1110)	B_{35}	4	2

- RAM C controlled by counter N and counter H (N = 5 and H = 4 as an example)

Memory locations in RAM C	Coefficient	Counter N	Counter H
0 (00000)	C_{11}	0 (000)	0 (00)
1 (00001)	C_{12}	1 (001)	0
2 (00010)	C_{13}	2 (010)	0
3 (00011)	C_{14}	3 (011)	0
4 (00100)	C_{15}	4 (100)	0
5 (00101)	C_{21}	0	1 (01)
6 (00110)	C_{22}	1	1
7 (00111)	C_{23}	2	1
8 (01000)	C_{24}	3	1
9 (01001)	C_{25}	4	1
10 (01010)	C_{31}	0	2 (10)
11 (01011)	C_{32}	1	2
12 (01100)	C_{33}	2	2
13 (01101)	C_{34}	3	2
14 (01110)	C_{35}	4	2
15 (01111)	C_{41}	0	3 (11)
16 (00001)	C_{42}	1	3
17 (00010)	C_{43}	2	3
18 (00011)	C_{44}	3	3
19 (00100)	C_{45}	4	3

Q4 (Control logic / FSM): What is the behaviour of the state machine? How many states do you need? What are the signals to be generated to control the memories, the MACC unit, and the counters as a function of each state? Draw the state graph of the state machine, with a description of each state and a list of all the control signals generated in each state

- Behaviour of the state machine

  - The FSM stays at IDLE, which is the initial state of FSM, waiting until NXT input signal goes high.

  - MACC gets the control signal from FSM and it starts to do calculation of the first coefficient of matrix C. It does multiplication of two numbers stored in address 0 of two ROMs and then store product in ACC. At the same time, M counter should count to ask combinational logic to give next address to let ROM output the next value multiplied in MACC. This product will be added up with previous one (for the first time of sum, the previous value will be set to 0), which is output from ACC. This process could be ended when M counter increment to its terminal values.

  - After calculating the first coefficient of matrix C, MACC gets reset to ensure adding 0 when doing the first time of sum for the next coefficient. The write_en is enabled to store the coefficient in RAM at corresponding address. However, when the final coefficient of matrix C is calculated, the data_out goes high, which makes the whole

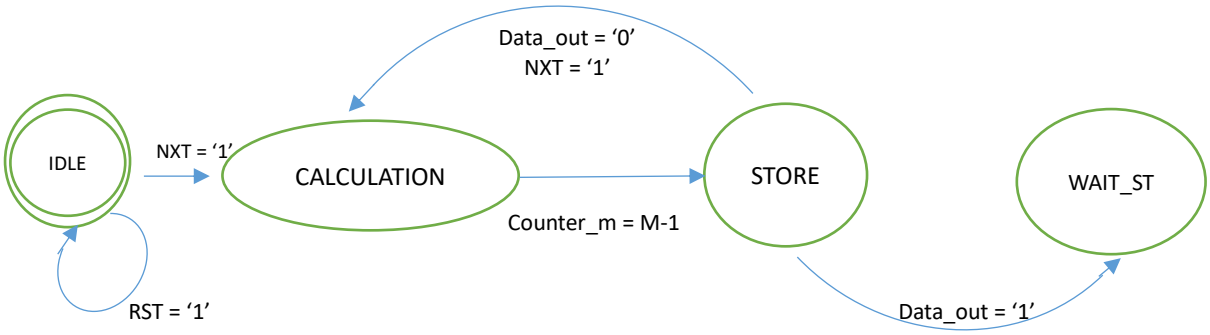
system to wait forever.

- When there is another NXT button pressed and the coefficients in C haven't been all figured out, update the address of ROM A and B, loading new coefficients to let MACC calculate the next coefficient of matrix C. It will repeat from calculation state.

- When all coefficients in matrix C has been computed, there is no necessity to go back to calculation and so it will freeze at wait state. This happens when both H and N reaches the maximum value, specifically, for our design, when H reaches the maximum value.

- To give appropriate control signals to the circuits, we take advantage of a 4-state mealy machine.

- State diagram



- Description of each state

States name	States description
Idle	It is the reset state. In this state, memory inputs are address 0. MACC isn't enabled so that no calculation is processing and no input is loaded. Every time the reset button is pressed, which means the reset signal goes high, the next state will be IDLE state. It is the initial state for the whole finite state machine and all values including the address in ROM A, B, C and counter M, N, H are set to 0, waiting to count or change. Once the user presses NXT, the MACC starts load the value at first address of ROMs and the state goes to calculation.
Calculation	This state is to load values from ROM A and B, meanwhile, calculate the coefficient of matrix C according to the values in current address of ROM A and B. It repeats addition and multiplication in MACC until one coefficient has been worked out, where M counter counts to the terminal value, which is M-1.
Store	Store state is used to store the coefficient figured out by calculation into RAM C and give an output of computed coefficient. At this stage, values can be written into RAM C and the addresses of numbers to be loaded into MACC should be updated. Meanwhile, MACC should be reset, waiting for computing a new coefficient. In terms of the time when it goes to the next state, it is controlled by datat_out and nxt button. Data_out = '1' is defined by that H counter counts to the maximum value. When nxt button is pressed, meanwhile, H counter doesn't count to maximum, it goes back to calculation state again, loading values and computing another coefficient of matrix C. However, when data_out is equal to 1, which means all coefficients in C has been calculated, it will go to wait state, which means waiting forever.
Wait_ST	In this state, the system should freeze. After displaying the last value computed, this state will ignore any further NXT input until reset to start a new computation from scratch.

- Table of the next state and condition

Current_state	*Next_state*
Idle	Idle: else Calculation: when NXT = '1'
Calculation	Idle: when rst = '1' Store: when counter_m = M-1 Calculation: else
Store	Idle: when rst = '1' Wait_ST: when data_out = '1' Calculation: when data_out = '0' and NXT = '1' Calculation: else
Wait_ST	Idle: when rst = '1' Wait_ST: else

- Table of FSM output values

Current_State	*rst_macc*	*en_macc*	*en_M*	*en_H*	*en_N*	*write_en_ram*
Idle	1	0	0	0	0	0
Calculation	0	1	1	0	0	0
Store	1 (NXT='1')	0	0	1 (NXT='1')	1 (NXT='1' and counter_n = (N-1))	1
Wait_ST	1	0	0	0	0	0

2. Commented VHDL code

2.1 Top-level

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity Top_level is

--------------------set generic value for N,M,H,data_size------------------

-- Default values 2*3 matrix A and 3*2 matrix B
generic( M: natural := 3; -- the number of columns of A and rows of B
         N: natural := 5; -- the number of columns of B
         H: natural := 4; -- the number of rows of A
         data_size: natural := 5); -- how many bits of binary number
                                   -- representing the data stored in ROM

---------------set inputs and outputs for the matrix multiplication--------

-- Define ports in the circuits
Port (CLK    : in STD_LOGIC; -- time sequence
      RST    : in STD_LOGIC; -- global reset
      NXT    : in STD_LOGIC; -- enable for the whole circuits

      OUTPUT: out STD_LOGIC_VECTOR ((size(((2**(data_size - 1))**2)*M))
downto 0)); -- equation to calculate how many bits to represent
            -- coefficients in matrix C

end Top_level;

architecture Behavioral of Top_level is

----------------------------Internal signals-------------------------------

  -- Internal signals in the circuits
  signal inv_rst : STD_LOGIC; -- inverted reset signal (compensates
                              -- active low)

  -- internal signals related to debouncer
  signal deb_rst, deb_nxt : STD_LOGIC; -- debounced reset and "next"
signals

  -- internal signals related to ROM address
  signal address_A: UNSIGNED ((size(H*M-1)-1) downto 0); -- to tell the
ROM A from which address to read contents
  signal address_B: UNSIGNED ((size(N*M-1)-1) downto 0); -- to tell the ROM
B from which address to read contents

  -- internal signals related to single-port RAM
  signal address_C : UNSIGNED ((size(H*N-1)-1) downto 0); -- to tell the
RAM C which address to wrtie contents in and output contents from
  signal ram_write_en : STD_LOGIC; -- to enable RAM to write value in the
corresponding location

  -- internal signals related to MACC
  signal rst_macc, en_macc: STD_LOGIC; -- reset and enable to control MACC

begin
```

```
  -- Inversion of reset signal to compensate for active-low button
  inv_rst <= not RST;

--------------------Debouncer for (inverted) reset signal-----------------
  Rst_Debouncer: entity work.Debouncer
  PORT MAP( clk     => CLK,
            Sig     => inv_rst,
            Deb_Sig => deb_rst);

--------------------Debouncer for for "NXT" signal------------------------
  NXT_Debouncer: entity work.Debouncer
  PORT MAP( clk     => CLK,
            Sig     => NXT,
            Deb_Sig => deb_nxt);

----------------------Datapath inside the circuits-------------------------
  Datapath_tp: entity work.Datapath
  PORT MAP( clk             => clk,
            RAM_write_en    => ram_write_en,
            MACC_enable     => en_macc,
            MACC_rst        => rst_macc,
            ROM_A_address   => address_A,
            ROM_B_address   => address_B,
            RAM_C_address   => address_C,
            output_datapath => OUTPUT);

---------------Control logic providing control signal to datapath----------
  Control: entity work.Control_logic
  PORT MAP( rst          => deb_rst,
            nxt          => deb_nxt,
            clk          => CLK,
            rst_macc     => rst_macc,
            en_macc      => en_macc,
            address_romA => address_A,
            address_romB => address_B,
            address_ram  => address_C,
            write_en_ram => ram_write_en);

end Behavioral;
```

2.2.1 Datapath

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity Datapath is

--------------------set generic value for N,M,H,data_size-----------------

-- Default values for generic M, H, N and data_size
generic( M: natural := 3; -- the number of columns of A and rows of B
         H: natural := 4; -- the number of rows of A
         N: natural := 5; -- the number of columns of B
         data_size: natural := 5); -- width of data stored in ROM A

---------------set inputs and outputs for datapath-------------------------

-- Define ports in the datapath
Port ( clk            : in STD_LOGIC; -- time sequence
       RAM_write_en   : in STD_LOGIC; -- write enable for RAM
       MACC_enable    : in STD_LOGIC; -- enable for MACC
       MACC_rst       : in STD_LOGIC; -- reset for MACC

       -- Width of ROM A and B is defined by H, M and N,
       -- deduced by the maximum locations needed in the matrix A and B
       -- ROM A address
       ROM_A_address : in UNSIGNED ((size(H*M-1)-1) downto 0);
       -- ROM B address
       ROM_B_address : in UNSIGNED ((size(N*M-1)-1) downto 0);

       -- Width of RAM C is defined by H and N,
       -- derived by considering about the maximum locations needed
       RAM_C_address : in UNSIGNED ((size(H*N-1)-1) downto 0);

       -- In terms of how many bits we use to represent output,
       -- use size function to calculate how many bits needed to represent
       -- maximum value from matrix multiplication
       -- The derivation takes advantage of assuming all products acquired
       -- by multiplying two negative values and then sum up
       output_datapath: out STD_LOGIC_VECTOR ((size(((2**(data_size -
1))**2)*M)) downto 0));

end Datapath;

architecture Behavioral of Datapath is

----------------------------Internal signals--------------------------------

     -- Internal signals in datapath
     -- internal signals related to ROM A and ROM B
     signal data_A, data_B : STD_LOGIC_VECTOR ((data_size-1) downto 0);
-- values at corresponding address for ROM A and ROM B

     -- internal signals related to single-port RAM
     signal Data_in_ram : STD_LOGIC_VECTOR ((size(((2**(data_size -
1))**2)*M)) downto 0); -- Output from MACC and input to RAM, which holds
                       -- the coefficient of matrix C

begin
```

```
-------------------ROM to store values in matrix A-------------------------
  ROM_A: entity work.asynchronous_read_ROM
  generic map( M         => M,
               H         => H,
               data_size => data_size)

  PORT MAP( address => ROM_A_address,
            Dataout => data_A);

--------------------ROM to store values in matrix B-------------------------
  ROM_B: entity work.asynchronous_read_ROMB
  generic map( M         => M,
               N         => N,
               data_size => data_size)

  PORT MAP( address => ROM_B_address,
            Dataout => data_B);

---------RAM to store results of matrix C and output from the system-------
  RAM_C: entity work.RAM
  generic map( M         => M,
               N         => N,
               H         => H,
               data_size => data_size)

  PORT MAP( clk      => clk,
            write_en => RAM_write_en,
            Address  => RAM_C_address,
            Data_In  => Data_in_ram,
            Data_out => output_datapath);

------------------MACC to calculate coefficients in matrix C--------------
  MACC: entity work.MACC
  generic map( M     => M,
               data_size => data_size)

  PORT MAP( clk         => clk,
            rst         => MACC_rst ,
            en          => MACC_enable,
            A           => data_A,
            B           => data_B,
            macc_output => Data_in_ram);

end Behavioral;
```

2.2.2 ROM A

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity asynchronous_read_ROM is

--------------------set generic value for M,H,data_size--------------------

-- Default values for generic M and H and data_size
generic( M:         natural := 3; -- the number of columns of A
         H:         natural := 4; -- the number of rows of A
         data_size: natural := 5); -- width of data stored in ROM A

-------------------Define ports in the ROM A-------------------------------

Port ( address: in UNSIGNED ((size(H*M-1)-1) downto 0); -- width of the
address bus is defined by this equation. details are in report

       DataOut: out STD_LOGIC_VECTOR ((data_size-1) downto 0)); -- width of
data in ROM A is data_size

end asynchronous_read_ROM;

architecture Behavioral of asynchronous_read_ROM is

-- As matrix A is defined as 4*3, there should at least have 12 locations
-- As calculated, to represent 12 locations, we could use 5 bits
-- implement matrix in row-first order with the memory
type ROM_Array is array (0 to (2**log2(H*M)-1)) of
STD_LOGIC_VECTOR((data_size-1) downto 0);
  constant Content: ROM_Array := ( -- implement values into ROM A in binary
                      --and only implement values in locations needed
    0      => B"10000", -- -16
    1      => B"10000", -- -16
    2      => B"10000", -- -16
    3      => B"01111", -- 15
    4      => B"01111", -- 15
    5      => B"01111", -- 15
    6      => B"11010", -- -6
    7      => B"01011", -- 11
    8      => B"01010", -- 10
    9      => B"11000", -- -8
    10     => B"00000", -- 0
    11     => B"00001", -- 1
    others => B"00000"); -- for other addresses which are redundant

begin
     DataOut <= Content(to_integer(address)); -- Asynchronous read

end Behavioral;
```

2.2.3 ROM B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity asynchronous_read_ROMB is

--------------------set generic value for N,M,data_size-----------------

-- Default values for generic M and N and data_size
generic( M:         natural := 3; -- the number of rows of B
         N:         natural := 5; -- the number of columns of B
         data_size: natural := 5); -- width of data stored in ROM B
                                   -- which must be the same as ROM A

-------------------Define ports in the ROM B-------------------------------

Port ( address     : in UNSIGNED ((size(N*M-1)-1) downto 0); -- width of
the address bus is defined by this equation. details are in report

       DataOut     : out STD_LOGIC_VECTOR ((data_size-1) downto 0));
            -- width of data in ROM B is data_size

end asynchronous_read_ROMB;

architecture Behavioral of asynchronous_read_ROMB is

-- As matrix B is defined as 3*5, there should at least have 15 locations
-- As calculated, to represent 15 locations, we could use 5 bits
-- implement matrix in row-first order with the memory
type ROM_Array is array (0 to (2**log2(N*M)-1)) of
STD_LOGIC_VECTOR((data_size-1) downto 0);
  constant Content: ROM_Array := ( -- implement values into ROM A in binary
                        -- and only imlement calues in locations needed
    0      => B"10000", -- -16
    1      => B"01111", -- 15
    2      => B"11111", -- -1
    3      => B"11101", -- -3
    4      => B"11010", -- -6
    5      => B"10000", -- -16
    6      => B"01111", -- 15
    7      => B"11110", -- -2
    8      => B"00000", -- 0
    9      => B"01010", -- 10
    10     => B"10000", -- -16
    11     => B"01111", -- 15
    12     => B"00000", -- 0
    13     => B"01000", -- 8
    14     => B"11101", -- -3
    others => B"00000"); -- for other addresses which are redundant

begin
     DataOut <= Content(to_integer(address)); -- Asynchronous read

end Behavioral;
```

2.2.4 RAM C

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

-- Synchronous write / asynchronous read single-port RAM
entity RAM is

--------------------set generic value for N,M,H,data_size-----------------

-- Default values for generic M, H, N and data_size
generic( M:         natural:= 3; -- the number of columns of A and rows of B
         H:         natural:= 4; -- the number of rows of A
         N:         natural:= 5; -- the number of columns of B
         data_size:natural:= 5); -- width of data stored in ROM A

-------------------Define ports in the RAM C-------------------------------
Port ( clk     : in  STD_LOGIC;
       write_en: in  STD_LOGIC; -- Write enable which allows data written
                                -- in this RAM
       Data_In : in  STD_LOGIC_VECTOR ((size(((2**(data_size - 1))**2)*M))
downto 0); -- use function discovered in report to define the width of
           -- coefficients in matrix C
       Address : in  UNSIGNED ((size(H*N-1)-1) downto 0); -- width of the
          -- address bus is defined by this equation. details are in report

       Data_Out: out STD_LOGIC_VECTOR ((size(((2**(data_size - 1))**2)*M))
downto 0)); -- use function discovered in report to define the width of
            -- coefficients in matrix C

end RAM;

architecture Behavioral of RAM is

type ram_type is array (0 to (2**log2(H*N)-1)) of
STD_LOGIC_VECTOR((size(((2**(data_size - 1))**2)*M)) downto 0);
signal ram_inst: ram_type;

begin

-------------------Synchronous write (write enable signal)-----------------
  process (clk)
  begin
    if (rising_edge(clk)) then
      if (write_en='1') then
        ram_inst(to_integer(Address)) <= Data_In;
      end if;
    end if;
  end process;

----------------------------Asynchronous read-------------------------------
  Data_Out <= ram_inst(to_integer(Address));

end Behavioral;
```

2.2.5.1 MACC

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity MACC is

--------------------set generic value for M,data_size----------------------

  generic(data_size: natural := 5;
          M:         natural := 3);

-------------------------Define ports in MACC-----------------------------

  Port ( clk         : in STD_LOGIC;
         rst         : in STD_LOGIC;
         en          : in STD_LOGIC;
         A           : in STD_LOGIC_VECTOR ((data_size-1) downto 0);
         -- output from one of the ROM, sending data at current address
         B           : in STD_LOGIC_VECTOR ((data_size-1) downto 0);
         -- output from another ROM, sending data at current address

         macc_output : out STD_LOGIC_VECTOR ((size(((2**(data_size -
1))**2)*M)+1)-1 downto 0)); -- use function discovered in report to define
                             -- the width of coefficients in matrix C

end MACC;

architecture Behavioral of MACC is

---------------------------Internal signals-------------------------------

  signal result_multi: SIGNED (((data_size)*2)-1 downto 0);
  -- the result of the multiplication. The width to represent the maximum
  -- product is described as two times of width of multipliers

  signal result_add  : SIGNED (size(((2**(data_size - 1))**2)*M) downto 0);
  -- same maximum width as macc_output and acc_input and acc_output

  signal acc_add: STD_LOGIC_VECTOR (size(((2**(data_size - 1))**2)*M)
downto 0); -- take the output from ACC as an addend of adder

begin

------Design port map and generic map to connect ACC with ports in MACC----
  MACC: entity work.ACC -- generate MACC based on ACC
  generic map (data_size => data_size,
               M         => M)
  Port map (input_acc  => STD_LOGIC_VECTOR(result_add),
            en         => en,
            rst        => rst,
            clk        => clk,
            output_acc => acc_add);

-----------Define formula of values for internal signals inside MACC-------
  -- result_multi is given by values from ROM multiplied by each other
  result_multi <= (signed(A) * signed(B));

  -- result_add is given by adding product of current values at current
```

```
  -- address of ROM A and B and value sent from ACC
  -- initial value sent from ACC is set as 0 by setting output_acc 0
  result_add <= (result_multi + SIGNED(acc_add));

  -- set the result_add also output of macc
  macc_output <= STD_LOGIC_VECTOR(acc_add);

end Behavioral;
```

2.2.5.2 ACC

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity ACC is

--------------------set generic value for M,data_size----------------------
generic(data_size: natural := 5;
        M:         natural := 3);

---------------------------Define ports in ACC----------------------------
Port ( clk       : in STD_LOGIC;
       rst       : in STD_LOGIC;
       input_acc : in STD_LOGIC_VECTOR(size(((2**(data_size - 1))**2)*M)
downto 0); -- widths of input and output are same
       en        : in STD_LOGIC; -- enable to tell macc when to store the
                                 -- sum from addition, generated by FSM

       output_acc: out STD_LOGIC_VECTOR (size(((2**(data_size - 1))**2)*M)
downto 0)); -- equation derivation is in report:
            -- (size(((2**(data_size - 1))**2)*M)+1)-1

end ACC;

architecture Behavioral of ACC is

begin

--------------------------Define a process of ACC-------------------------
  -- ACC is like a register to take the input from adder and give the
  -- output back to one input of adder
  -- Once one coefficient of matrix C has been calculated,
  -- there should be an enable signal to enable ACC to give output to RAM
  -- instead of adder
  -- Initially, the output and the input are set to 0 for ACC
  -- rst signal for ACC will set all values back to 0
  acc: process(clk, en)
  begin
    if(rising_edge(clk)) then
      if(rst = '1') then -- with synchronous reset
        output_acc <= (others => '0'); -- When reset is pressed
      elsif(en = '1') then -- with wynchronous enable
        output_acc <= input_acc;
      end if;
    end if;
  end process acc;

end Behavioral;
```

2.3 Control logic

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity Control_logic is

----------------Default values for generic M, H, N-------------------------
generic( M: natural := 3; -- the number of columns of A and rows of B
         H: natural := 4; -- the number of rows of A
         N: natural := 5); -- the number of columns of B

----------------------Define ports in Control logic-----------------------
Port ( rst          : in STD_LOGIC;
       nxt          : in STD_LOGIC;
       clk          : in STD_LOGIC;

       rst_macc     : out STD_LOGIC; -- reset of macc
       en_macc      : out STD_LOGIC; -- enable of macc
       address_ram  : out UNSIGNED ((size(H*N-1)-1) downto 0);
       -- address to RAM to tell which number to output
       address_romA : out UNSIGNED ((size(H*M-1)-1) downto 0);
       -- address to ROM1 to tell which number should be sent to MACC A
       address_romB : out UNSIGNED ((size(N*M-1)-1) downto 0);
       -- address to ROM2 to tell which number should be sent to MACC B
       write_en_ram : out STD_LOGIC); -- writing enable for RAM

end Control_logic;

architecture Behavioral of Control_logic is

------------------Declare the states as an enumerated type---------------
  -- There are five states
  type fsm_states is (IDLE, CALCULATION, STORE, WAIT_ST);

----------------------------Internal signals-------------------------------
  signal state, next_state: fsm_states; -- of this type above

  -- Define as 1 when N and H both reaches terminal values but in other
  -- cases, keep 0 to let the next coefficient calculated
  signal data_out: STD_LOGIC;    signal en_M, en_N, en_H: STD_LOGIC;

  -- Define enable to let M N H counter count to update values out of ROM
  signal counter_m: UNSIGNED (log2(M)-1 downto 0); -- width of M counter
  signal counter_n: UNSIGNED (log2(N)-1 downto 0); -- width of N counter
  signal counter_h: UNSIGNED (log2(H)-1 downto 0); -- width of H counter
  -- as a signal to tell when data_out should be set to 1. It also
  -- indicates current location of ram
  signal ram: UNSIGNED ((size(H*N-1)-1) downto 0);

begin

-------------------------Define M counter process-------------------------
  M_counter: entity work.Param_Counter
  generic map (LIMIT => M)
  PORT MAP( clk => clk,
            rst => rst,
            en => en_M,
            count_out => counter_m);
```

```
-------------------------Define N counter process-------------------------
  N_counter: entity work.Param_Counter
  generic map (LIMIT => N)
  PORT MAP( clk => clk,
            rst => rst,
            en => en_N,
            count_out => counter_n);

-------------------------Define H counter process-------------------------
  H_counter: entity work.Param_Counter
  generic map (LIMIT => H)
  PORT MAP( clk => clk,
            rst => rst,
            en => en_H,
            count_out => counter_h);

-------------------------Define the state register----------------------
  -- synchronous process sensitive only to clock
  state_assignment: process (clk) is
  begin
    if (rising_edge(clk)) then
      if(rst = '1') then -- Define a reset state
        state <= IDLE;
      else
        state <= next_state;
      end if;
    end if;
  end process state_assignment;

------Define the fsm transition rules which is a mealy machine-------------
  FSM_transitions: process (state, rst, nxt, data_out, counter_m) is
  begin
    case state is
      when IDLE => -- stage 1 is a reset state which all addresses in
                   -- memory are set to 0 and MACC is disabled,
                   -- as well as counters are set to 0
                   if(nxt = '1') then next_state <= CALCULATION;
                   else next_state <= IDLE;
                   end if;
      when CALCULATION => -- stage 2 is to load values from ROM A and B and
                          -- do multiplication and addition of two numbers
                          -- It repeats until the last coefficient of C has
                          -- been worked out
                   if(counter_m = M-1) then next_state <= STORE;
                   -- When reset button is pressed, it goes back to IDLE
                   elsif(rst = '1') then next_state <= IDLE;
                   else next_state <= CALCULATION;
                   end if;
      when STORE => -- stage 3 is to store coefficients calculated by
                    -- datapath into RAM C and data_out as a signal to
                    -- check which is the next state
                    -- The condition starting to calculate the next
                    -- coefficient is when nxt button is pressed and
                    -- there remains coefficients needed calculating
                    -- (data_out = '0')
                    -- The condition waiting forever is the last
                    -- coefficient has been computed (data_out = '1')

                    if(data_out = '1') then next_state <= WAIT_ST;
                    -- When reset button is pressed, it goes back to IDLE
                    elsif(rst = '1') then next_state <= IDLE;
```

```
                    elsif(data_out = '0' and nxt = '1') then next_state <=
CALCULATION;
                    else next_state <= STORE;
                    end if;
      when WAIT_ST => -- stage 4 is a freeze state, which will wait
                      -- forever, ignoring nxt button pressed but wait for
                      -- the reset button to make it go back to IDLE
                      -- It only happens all coefficients in matrix C has
                      -- been computed
                      -- When reset button is pressed, it goes back to IDLE
                      if(rst = '1') then next_state <= IDLE;
                      else next_state <= WAIT_ST;
                      end if;
      end case;
  end process FSM_transitions;

---------------------Define combinational logic signals--------------------
  -- By means of resize function, convert size of numbers counted by M,N,H
  -- counters to size of addresses required of ROM A and B as well as RAM C
  -- Because width in counter and width of memory are different, we convert
  -- width in counter to width in memory
  -- Because M is a non-negative number, we use to_unsigned function to
  -- convert it to an unsigned vector with the specified size, which is
log2(M)
  -- Because N is a non-negative number, we use to_unsigned function to
  -- convert it to an unsigned vector with the specified size, which is
log2(N)
  -- * performs the multiplication operation on two unsigned vectors which
  -- possibly be of different lengths
  -- + performs the addition on two unsigned vectors which possibly be of
  -- different lengths
  -- according to sheets written in report, we can find corresponding
  -- address as given values from counter_m and counter_h
  -- because values are stored following row-first order, it should the
  -- first unsigned parameter should be set current column + current row *
  -- unsigned number showing how many columns there are.
  -- resize function resizes an unsigned vector which is
  -- (counter_m + counter_h * to_unsigned(M, log2(M)) to the specified size
  -- size(M*H-1), which is recognized by ROM A
  address_romA <= resize(counter_m + counter_h * to_unsigned(M, log2(M)),
size(M*H-1));

  -- resize function resizes an unsigned vector which is
  -- counter_n + counter_m * to_unsigned(N, log2(N)) to the specified size
  -- size(M*N-1), which is recognized by ROM B
  address_romB <= resize(counter_n + counter_m * to_unsigned(N, log2(N)),
size(M*N-1));

  -- resize function resizes an unsigned vector which is
  -- (counter_n + counter_h * to_unsigned(N, log2(N)) to the specified size
  -- size(N*H-1), which is recognized by RAM C
  ram <= resize(counter_n + counter_h * to_unsigned(N, log2(N)), size(N*H-
1));
  address_ram <= ram;
  -- Define output from FSM
  -- reset of macc is at rising edge in Idle state and freeze state as well
  -- as store state when nxt button is pressed
  rst_macc <= '1' when ((state = IDLE) or (state = STORE and NXT = '1') or
(state = WAIT_ST)) else
              '0';
```

```
  -- enable of macc only when MACC is calculating coefficients of C
  en_macc <= '1' when (state = CALCULATION) else
             '0';

  -- M counter enable is set to high when MACC repeats multiplying two   --
  -- numbers because M counter needs to count column of matrix A and row of
  -- matrix B
  -- Only in Calculation state, MACC is working and column in A together
  -- with row in B of values going into multiplier needs changing
  en_M <= '1' when (state = CALCULATION) else
          '0';

  -- N counter enable is set to high only after working out one coefficient
  -- Because coefficients in matrix C are calculated row by row, every
  -- time, after working out one coefficient and nxt button is pressed,
  -- address for column of B must be changed
  -- It is controlled by N counter
  en_N <= '1' when (state = STORE and NXT = '1') else
          '0';

  -- H counter enable is set to high after working gout one coefficient and
  -- when N counter counts to the maximum
  -- Because at this point, matrix A should switch to the next row
  -- Current state should be stored and also counter_n should be terminal
  -- values as well as nxt button is pressed
  en_H <= '1' when (state = STORE and counter_n = (N-1) and NXT = '1') else
          '0';

  -- written enable of RAM C only when storing values into RAM C
  write_en_ram <= '1' when (state = STORE) else
                  '0';

  -- Define a signal called data_out to tell FSM if all coefficients of
  -- matrix C have been computed
  -- data_out is high when all coefficients of matrix C have been computed
  -- It happens when ram addresses count to maximum
  data_out <= '1' when (state = STORE and ram = H*N-1) else
              '0';

end Behavioral;
```

3. Commented self-checking testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL; -- contains size and log2 functions

entity top_level_tb is

end top_level_tb;

architecture Behavioral of top_level_tb is

--------Set constant as values for changing M,N,H and data_size-----------
constant M_tb:         natural := 3;
constant N_tb:         natural := 5;
constant H_tb:         natural := 4;
constant data_size_tb: natural := 5;

----------------------------clock period definitions----------------------
constant clk_period: time := 10ns;

----------------------------Internal signals-------------------------------
-- Input declaration
signal CLK:    STD_LOGIC;
signal RST:    STD_LOGIC;
signal NXT:    STD_LOGIC;

-- Output declaration
signal OUTPUT: STD_LOGIC_VECTOR ((size(((2**(data_size_tb - 1))**2)*M_tb))
downto 0);

-----Self-checking testbench to record as a table of test vectors----------
-- Self test vector type declaration
-- Only check if coefficients are computed right
type test_vector is record

  OUTPUT: STD_LOGIC_VECTOR ((size(((2**(data_size_tb - 1))**2)*M_tb))
downto 0);

end record;

------------Declare an array of records of the records type---------------
  -- This array holds coefficients existing in matrix C, which are
calculated manually
  -- These coefficients are saved as decimal instead of binary numbers
  -- size of this array is defined by calculating how many numbers are in
matrix C, which is N_tb*H_tb
  type test_vector_array is array
    (0 to (N_tb*H_tb-1)) of integer;

  -- Declare a constant of the array of records type
  constant test_vectors : test_vector_array := (
  -- generate sample values for OUTPUT
  -- these values will be compared with the calculated values from circuits
  -- As follows, the array is matrix C and the sheet is shown in report
  -- wrong numbers should be detected by self-checking testbench after test
  -- process in TCL console
  -- Right numbers will leave a note message in TCL console
  768, -720, 48, -80, -16,
  -720, 675, -45, 75, 15,
```

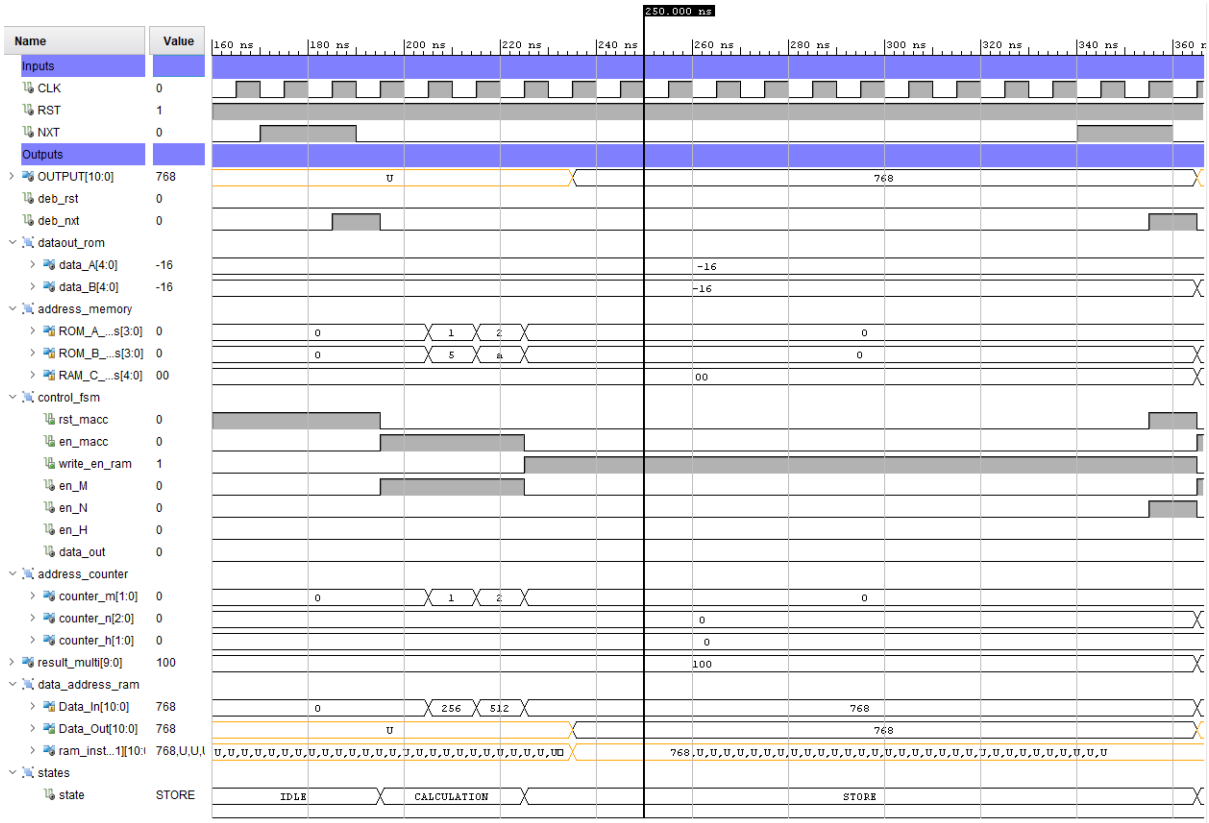
```
  -240, 225, -16, 98, 116,
  112, -105, 8, 32, 45);

begin
---------------Instantiate UUT as a component in testbench-----------------
  UUT: entity work.Top_level
  -- generic map in order to access all files which use variables below
  generic map ( M         => M_tb,
                N         => N_tb,
                H         => H_tb,
                data_size => data_size_tb)
  Port map (CLK    => CLK,
            RST    => RST,
            NXT    => NXT,
            OUTPUT => OUTPUT);

--------------------------------Clock process-------------------------------
  -- This process generates a clock signal to control sequential elements
  clk_process: process
    begin
      CLK <= '0';
      wait for clk_period/2;
      CLK <= '1';
      wait for clk_period/2;
  end process;

------Test process used for testbench to test I/O ports in entity----------
  TEST : process
    begin
      -- wait 100ns for global reset to finish
      wait for 100 ns;

      -- Input values to change the falling edge of the clock
      wait until falling_edge(clk);

      -- Reset the circuits to define initial value not U
      RST <= '1';
      NXT <= '0';
      wait for clk_period*2;
      RST <= '0';
      wait for clk_period*3;

      -- Set reset back to 0
      RST <= '1';
      wait for clk_period*2;

      -- Start to calculate coefficients in matrix C
      -- generate a for loop to repeat 22 times to calculate and let
      -- circuits give the outputs
      -- Generate a for loop to assert values one by one following order in
      -- test_vectors array
      pulse_1x22: for i in test_vectors' range loop
        NXT <= '1';
        wait for clk_period*2; -- 2-period pulse
        NXT <= '0';
        wait for clk_period*5; -- 5-period pulse

        -- Error message
      -- Insert values for OUTPUT to compare computation from circuits with
        -- manual computation
        -- Convert signed output to integer and compare it with value in
```

```
        -- test_vector array
        assert((to_integer(SIGNED(OUTPUT))) = test_vectors(i))

        -- The error message printing out what expected output is and what
        -- output from the circuits is
        report "TEST FAILED: " &
               "The expected output value should be " &
               integer'image(test_vectors(i)) &
               ". The real output value is " &
               integer'image(to_integer(SIGNED(OUTPUT)))
        severity error;

        -- Note message
      -- Insert values for not OUTPUT to compare computation form circuits
        -- with manual computation
        assert((to_integer(SIGNED(OUTPUT))) /= test_vectors(i))

    -- The note message printing expected output corresponds to real output
        report "TEST SUCCEED: " &
               "The expected output value and the real output value are
both " & integer'image(test_vectors(i))
        severity note;

      end loop pulse_1x22;

    -- test reset and restart after 20 clock periods
    rst <= '1';
    wait for clk_period*2;
    rst <= '0';
    wait for clk_period*2;

    pulse_1x10: for k in test_vectors' range loop
      NXT <= '1';
      wait for clk_period*2; -- 2-period pulse
      NXT <= '0';
      wait for clk_period*5; -- 5-period pulse

      -- Error message
      -- Insert values for OUTPUT to compare computation from circuits with
      -- manual computation
      -- Convert signed output to integer and compare it with value in
      -- test_vector array
      assert((to_integer(SIGNED(OUTPUT))) = test_vectors(k))

      -- The error message printing out what expected output is and what
      -- output from the circuits is
      report "TEST FAILED: " &
             "The expected output value should be " &
integer'image(test_vectors(k)) &
             ". The real output value is " &
integer'image(to_integer(SIGNED(OUTPUT)))
      severity error;

      -- Note message
      -- Insert values for not OUTPUT to compare computation form circuits
      -- with manual computation
      assert((to_integer(SIGNED(OUTPUT))) /= test_vectors(k))

    -- The note message printing expected output corresponds to real output
      report "TEST SUCCEED: " &
```

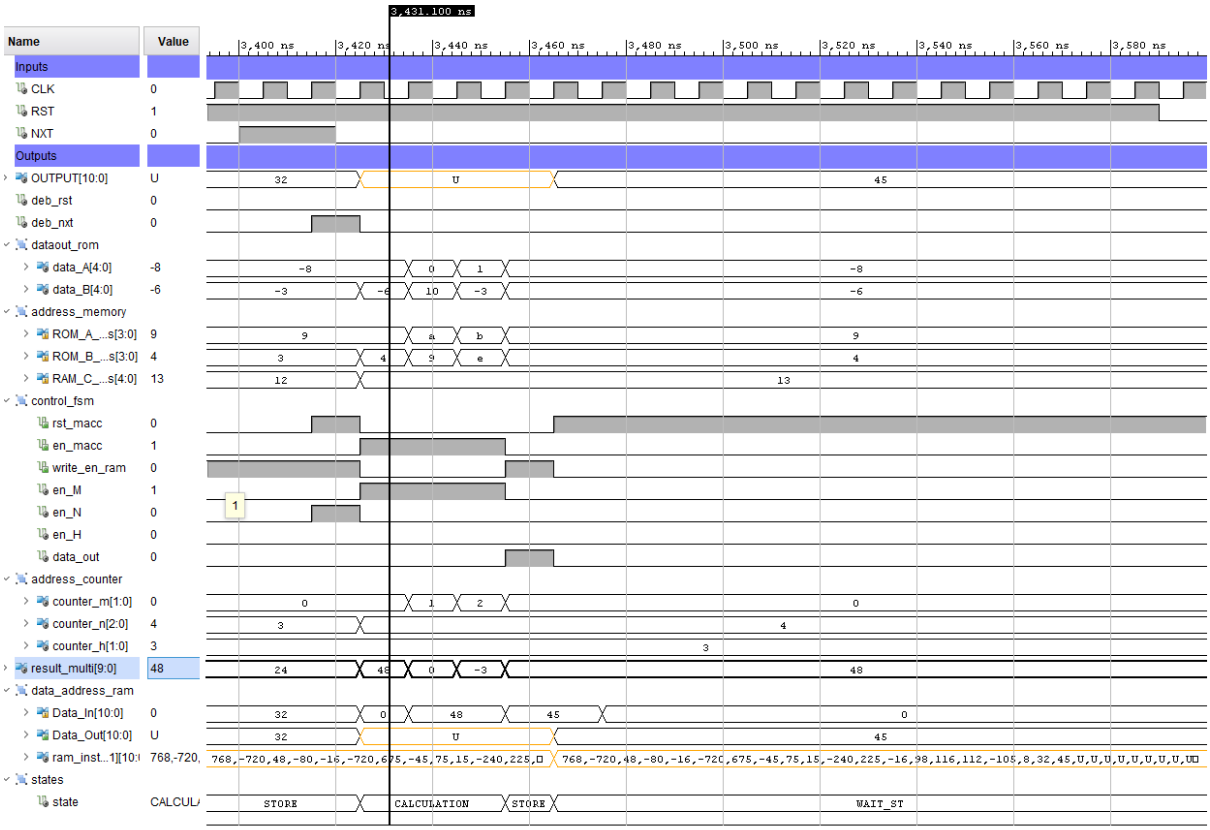
```
            "The expected output value and the real output value are both
" & integer'image(test_vectors(k))
        severity note;
      end loop pulse_1x10;

    wait; -- will wait forever
  end process;

end Behavioral;
```

4. Simulations

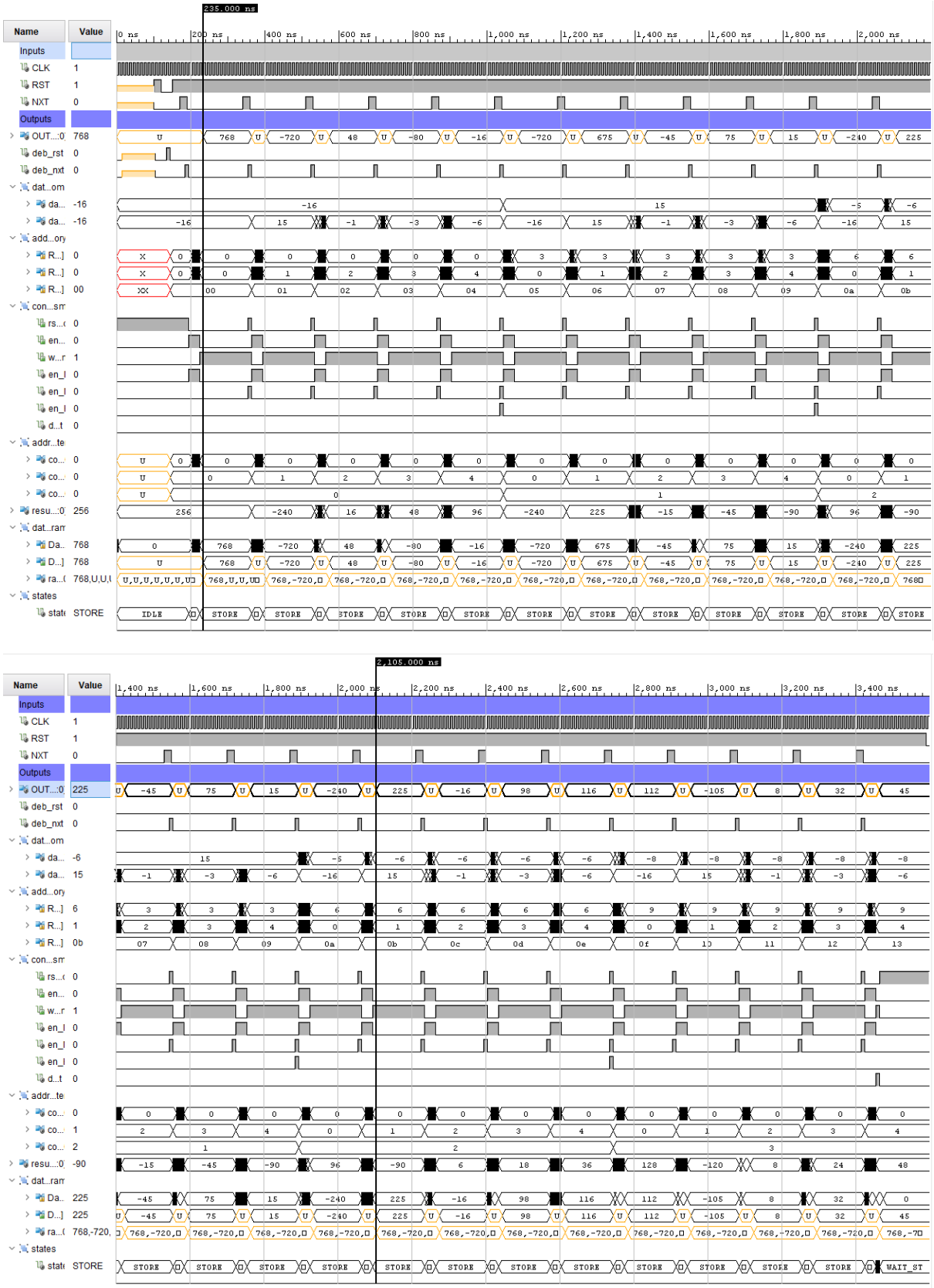
4.1 Complete computation of the first product matrix coefficient



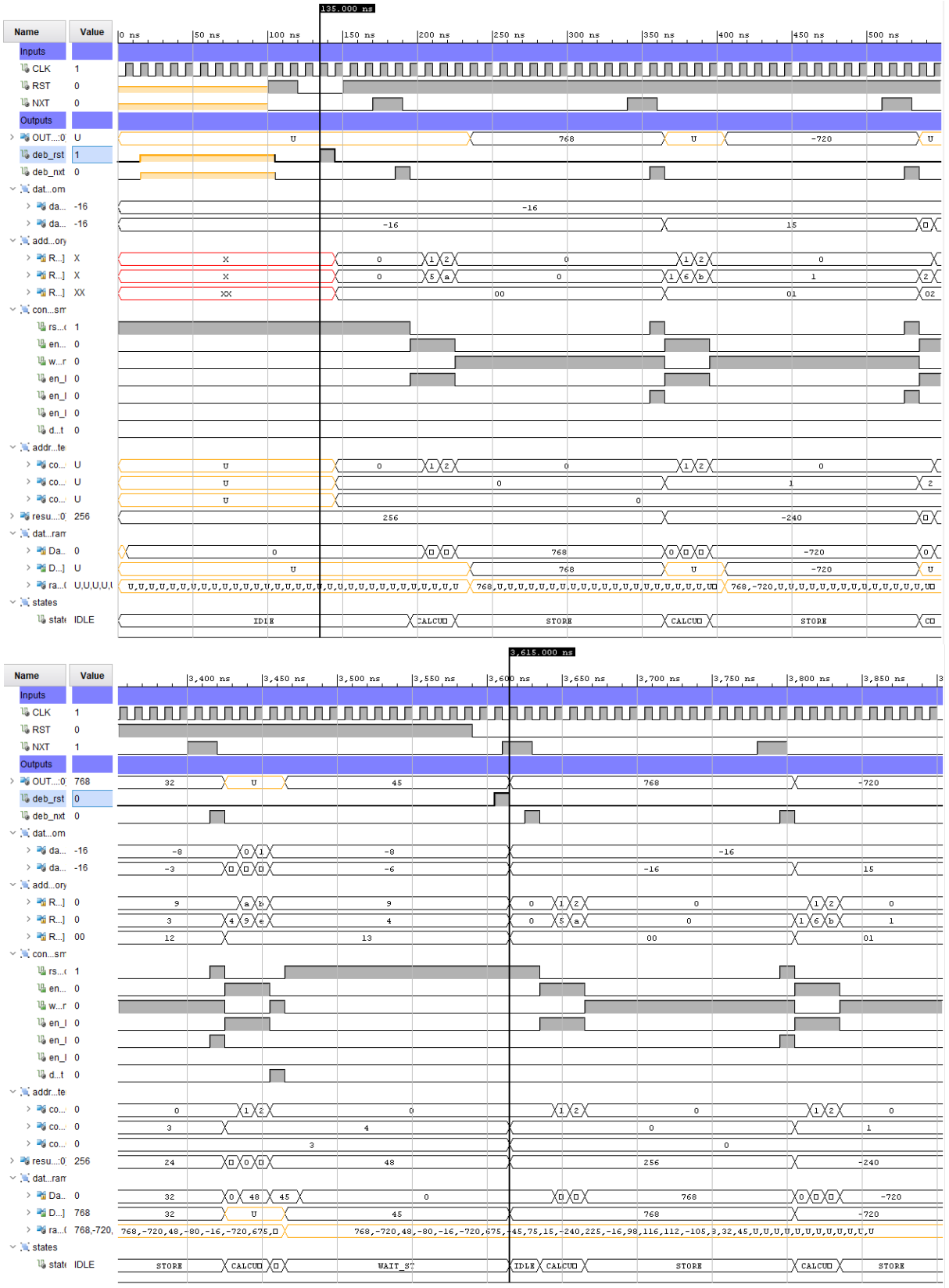
4.2 Complete computation of the last product matrix coefficient



4.3 Snapshots showing the final value of every coefficient on output



4.4 Significant event



As RAM has stored values at corresponding addresses and reset has set RAM back to the initial address, there is no undefined due to calculation state between two numbers stored in RAM. Although there are previous values in RAM, the output from RAM, MACC and ROM can show circuits still do matrix multiplication.

4.5 Tcl console

```
run 10 us
Note: TEST SUCCEED: The expected output value and the real output value are both 768
Time: 340 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -720
Time: 510 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 48
Time: 680 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -80
Time: 850 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -16
Time: 1020 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -720
Time: 1190 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 675
Time: 1360 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -45
Time: 1530 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 75
Time: 1700 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 15
Time: 1870 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -240
Time: 2040 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 225
Time: 2210 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -16
Time: 2380 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 98
Time: 2550 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 116
Time: 2720 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 112
Time: 2890 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -105
Time: 3060 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 8
Time: 3230 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 32
Time: 3400 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 45
Time: 3570 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 768
Time: 3780 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -720
Time: 3950 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 48
Time: 4120 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -80
Time: 4290 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -16
Time: 4460 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both -720
Time: 4630 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
Note: TEST SUCCEED: The expected output value and the real output value are both 675
Time: 4800 ns  Iteration: 0  Process: /top_level_tb/TEST  File: H:/VHDL/VivadoProjects/DDVHD_Final_project/Final_project.srcs/sim_1/new/top_level_tb.vhd
```

5. RTL component statistics

```
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input      3 Bit       Adders := 1
       2 Input      2 Bit       Adders := 2
+---Registers :
                   11 Bit    Registers := 1
                    3 Bit    Registers := 1
                    2 Bit    Registers := 3
                    1 Bit    Registers := 6
+---Muxes :
      16 Input      5 Bit        Muxes := 1
       2 Input      3 Bit        Muxes := 1
       2 Input      2 Bit        Muxes := 3
       8 Input      2 Bit        Muxes := 1
       2 Input      1 Bit        Muxes := 2
---------------------------------------------------------------------------------
Finished RTL Component Statistics
```