UNIVERSITY of York

Department of Electronic Engineering

Assessments 2018/19

ELE00030I

Digital Design with HDL

# Final Project

Implementation of a Parameterizable Matrix Multiplication System

VHDL_Project_Y3843738-Y3843317.pdf

## Contents

## Question Answers

### Question 1 (Datapath)

#### What is the maximum width (in terms of bits) of the coefficients of C and which generics are relevant?

The maximum size of the output bus (OutputSize) and width of the RAM will be determined by the largest possible negative number that can be generated from the signed binary numbers within the ROMs. This would come from the maximum negative number being multiplied by the maximum positive number M times.

For example, in with a data width of **4-bit Signed** using matrices of dimensions A = 2x3 and B = 3x2:

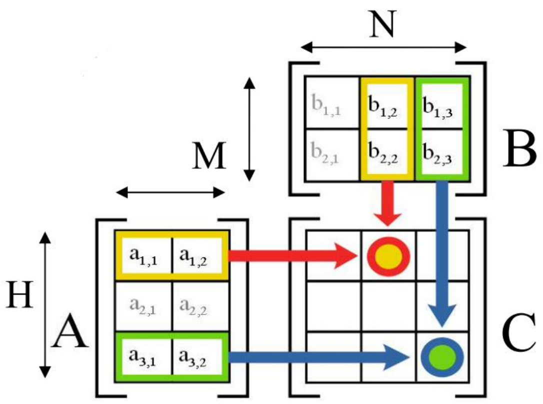


*Figure 1 - Multiplying two matrices (Tempesti, 2018)*

If A(1,1) and A(1,2) = -8

Along with B(1,2) and B(2,2) = 7

C (Red Circle) would be the largest negative number: -112

If A(3,1), A(3,2), B(1,3) and B(2,3) = -8

C (Blue Circle) would be the largest positive number: 128

The largest number is therefore calculated using this equation:

$$(M \times 2^{2\times(DataSize-1)})$$

The size function is used. Provided in the DigEng package, this returns the number of bits required to express exactly that value. E.g. To express the number 16, 5-bits is necessary. So the equation becomes:

$$Size(M \times 2^{2\times(DataSize-1)})$$

Then since the output is a signed value an extra bit must be added to account for this:

$$OutputSize = Size\left(M \times 2^{2\times(DataSize-1)}\right) + 1$$

### Question 2 (Datapath)

#### What is the width and depth of the memories that store A, B, and C, as a function of M, N, H?

From Figure 1 we can also work out the depth of the memories.

The depth of A will be: $M \times H$

The depth of B will be: $M \times N$

The depth of C will be: $N \times H$

Given the previous example in Figure 1, A will be: $2 \times 3 = 6$, B will be: $2 \times 3 = 6$ and C will be: $3 \times 3 = 9$.

The width of both A and B will be pre-determined based on the ROMs being used, and the value assigned to dataSize. Output buses from the ROMs will therefore be: `(data_size - 1 downto 0);`

The width of C (the RAM) will be: `(output_size - 1 downto 0);`

## What is then the width of the address bus for each of the memories as a function of M, N, H?

The width of the address bus for each memory will be the ceiling of the $\log_2$ function of their depths.

The address bus for ROM_A: $\log_2(M \times H)$

The address bus for ROM_B: $\log_2(M \times N)$

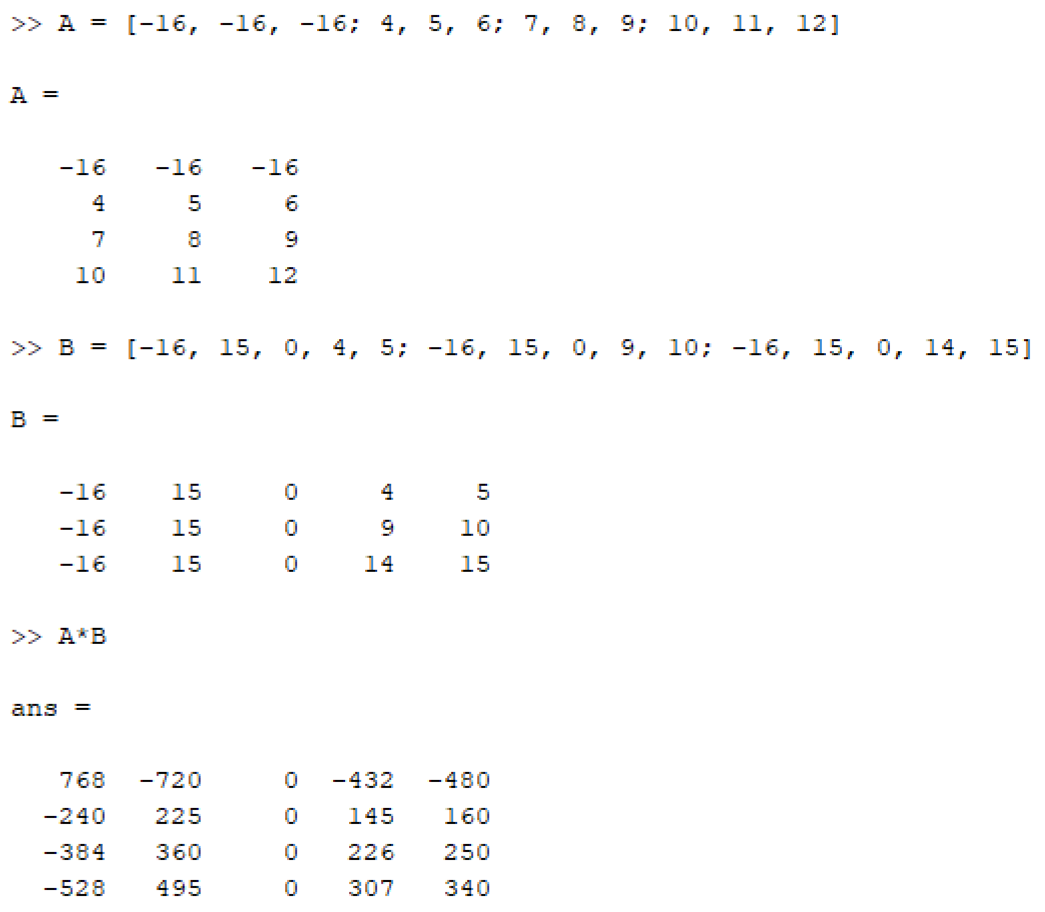
The address bus for RAM will be: $\log_2(H \times N)$

## What are the input matrices that you will use to test the system?

The testing of the Matrix Multiplier was done using the following generics M = 3, N = 5, and H = 4 and dataSize = 5.

This tested the maximum positive and negative numbers possible, along with zero output and further values to confirm correct functionality.

The below matrices were generated using MATLAB.



```
>> A = [-16, -16, -16; 4, 5, 6; 7, 8, 9; 10, 11, 12]

A =

   -16   -16   -16
     4     5     6
     7     8     9
    10    11    12

>> B = [-16, 15, 0, 4, 5; -16, 15, 0, 9, 10; -16, 15, 0, 14, 15]

B =

   -16    15     0     4     5
   -16    15     0     9    10
   -16    15     0    14    15

>> A*B

ans =

   768  -720     0  -432  -480
  -240   225     0   145   160
  -384   360     0   226   250
  -528   495     0   307   340
```

*Figure 2 - MATLAB generated input and output matrices.*

*First, second and third output are largest positive, largest negative and zero respectively.*

## Question 3 (Control Logic)

### How are the counters used for address generation related to each other?

The M counter will increment up to its maximum value. At this point the N counter will increment and the M counter will count over back to zero. The N counter will continue to do this until it reaches its maximum value. The H counter will then be incremented when both M and N counters reach their maximum value. Both M and N will count over back to zero.

This provides the coordinates for all matrix multiplications. However, this is not how the coefficients are stored in the ROM and combinational logic is needed to address them.

### When should each counter reset to 0 (beyond the global reset)?

The counters will reset to zero as they roll over their maximum value.

### What are the sizes (widths) of the counters?

The M counter will be a function of M it will count from 0 to M-1

The N counter will be a function of N it will count from 0 to N-1

The H counter will be a function of H it will count from 0 to H-1

### How are the addresses computed from the counter values?

Due to storing all the arrays row first in the ROMs, addressing the ROM using only the counters would not work. Combinational logic is therefore used as follows.

Address for ROM A: $Generic\_M \times Count\_H + Count\_M$

This is incrementing through the columns in the Matrix A shown in Figure 1 based on the M Counter and working down the row depending on the H counters value. The H counter value is multiplied by the size of the generic M which increments down H number of rows, or H rows worth of memory locations within the ROM.

Address for ROM B: $Generic\_N \times Count\_M + Count\_N$

This is incrementing through the columns in the Matrix B shown in Figure 1 based on the N Counter and working along the rows depending on the M counters value. The M counter value is multiplied by the size of the generic N which increments down M rows, or M rows worth of memory locations within the ROM.

Address for RAM: $Generic\_N \times Count\_H + Count\_N$

This increments down through the RAM in sequential order but based on the values of the counters H and N. N is only incremented as a full coefficient is calculated and cycles through the columns of the output matrix C in Figure 1. H * N increments the memory location over an H number of rows of the output matrix C.

## Question 4 (Control Logic / FSM)

### What is the behaviour of the state machine?

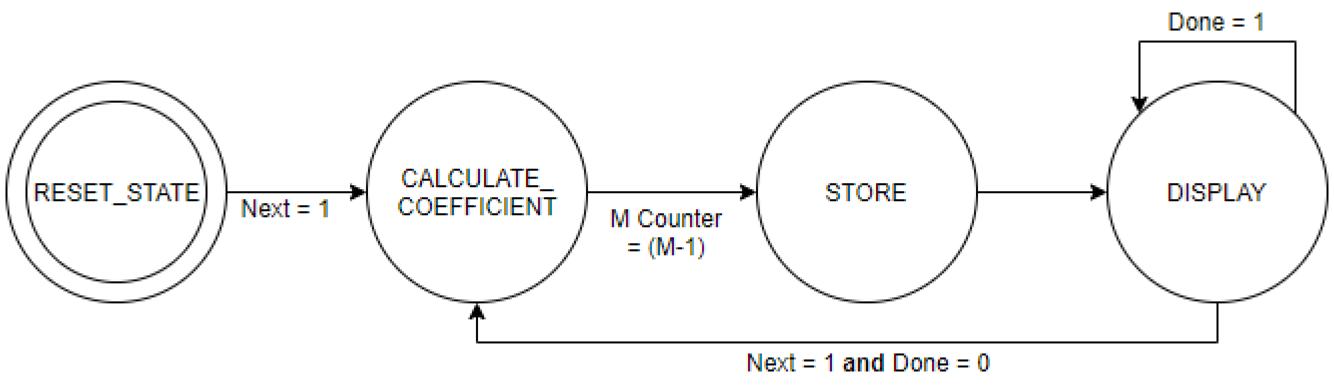


*Figure 3 - Mealy state transition diagram*

| | |
|---|---|
| RESET_STATE | In this state everything is reset, the MACC and counters are disabled. MACC_RST is held high removing any previous value. When the user presses NXT, the state goes to CALCULATE_COEFFICIENT. |
| CALCULATE_COEFFICIENT | In this state the M Counter is enabled. This updates the ROM addresses via combinational logic. MACC is enabled. Keeps incrementing counters and generating addresses for ROM's until count_M = Matrix_Common – 1. Then moves to STORE state. |
| STORE | In this state only write_EN is high. All other control signals are low. The address of the RAM is calculated via combinational logic. The value of the MACC is stored into this location. Always moves to DISPLAY state. |
| DISPLAY | In this state the output is displayed and the circuit waits for an input from the user. The N Counter is enabled when the user presses NXT if the done signal is low. The H counter is enabled based on the user pressing NXT **and** the done signal being low **and** the values of N **and** M counters. Write_EN and MACC_EN are both low. MACC_RST is high to remove the previous value. If done = 0, returns to CALCULATE_COEFFICIENT state, else remains in DISPLAY until RST. |

## What are the control signals for the memories, the MACC unit, and the counters as a function of each state?

| **State** | **CNT_EN_M** | **CNT_EN_N** | **CNT_EN_H** | **MACC_RST** | **MACC_EN** | **Write_EN** |
|---|---|---|---|---|---|---|
| **RESET_STATE** | 0 | 0 | 0 | 1 | 0 | 0 |
| **CALCULATE_COEFFICIENT** | Count_M < Matrix_Common-1 | 0 | 0 | 0 | 1 | 0 |
| **STORE** | 0 | 0 | 0 | 0 | 0 | 1 |
| **DISPLAY** | Done = 0 **and** NXT = 1 | Done = 0 **and** NXT = 1 | Done = 0 **and** NXT = 1 **and** Count_M = Matrix_ Common – 1 **and** Count_N = Matrix_B_Width - 1 | 1 | 0 | 0 |

# VHDL Code

## Top Level Matrix_Multiplier

```vhdl
----------------------------------------------------------------------------------
-- Company: University of York
-- Engineer: James Gardner
--
-- Create Date:    21/11/2018
-- Design Name:    Parameterizable Matrix-Multiplier
-- Module Name:    Matrix-Multiplier
-- Project Name:   Final Project
-- Tool versions:  Any (tested on ISE 2017.4)
-- Description:
--  A fully parameterizable matrix-multiplier, multiplies two pre-defined
--  ROMs and stores the output in a parameterizable RAM.
-- Dependencies:
--   Requires DigEng.vhd package
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;
-- This is the top-level component of the parameterizable matrix-multiplier.
-- It generates to debouncers for user inputs and connects the control-logic to
-- the datapath.
entity Matrix_Multiplier is
    -- The generics state the size of each matrix to be multiplied.
    Generic ( data_size : integer := 5; -- Number of bits per coefficient of the
                                        -- input matrices
          Matrix_A_Height : integer := 4; -- {H} The number of rows within input
                                          -- matrix A
          Matrix_B_Width : integer := 5; -- {N} The number of columns within input
                                         -- matrix B
          Matrix_Common : integer := 3); -- {M} The number of columns within input
                                         -- matrix A and number of rows within
                                         -- input matrix B
    Port ( CLK : in STD_LOGIC; -- Global clock input
           RST : in STD_LOGIC; -- User input debounced Reset
           NXT : in STD_LOGIC; -- User input debounced Next
           -- The size of the output buss is determined by the maximum value
           -- possible to represent using a binary number of data_size wide
           -- multiplied by itself Matrix_Common times.
           Output : out SIGNED ((size(Matrix_Common *
                               ((2**(data_size - 1))**2))) downto 0));
end Matrix_Multiplier;

architecture Behavioral of Matrix_Multiplier is

signal deb_rst, deb_nxt : STD_LOGIC;  -- debounced reset and "next" signals
-- Address sizes of both ROMs determined by the log2 function of generic variables
signal Address_ROM_A : UNSIGNED (log2(Matrix_A_Height*Matrix_Common)-1 downto 0);
signal Address_ROM_B : UNSIGNED (log2(Matrix_B_Width*Matrix_Common)-1 downto 0);
signal MACC_RST : STD_LOGIC; -- Reset output of multiply-accumulate unit to zero
signal MACC_EN : STD_LOGIC; -- Enable of multiply-accumulate unit
-- Address size of RAM determined by the log2 function of generic variables
signal Address_RAM : UNSIGNED(log2(Matrix_A_Height*Matrix_B_Width) -1 downto 0);
signal Write_EN : STD_LOGIC; -- Enabling writing to the RAM
```

```
-- The size of the output bus is determined by the maximum value possible to
-- represent using a binary number of data_size wide multiplied by itself
-- Matrix_Common times. Here it is declared as a constant to be used throughout
-- other components.
constant output_size : integer := size(Matrix_Common * ((2**(data_size - 1))**2))+1;

begin

-- Debouncer for "RST" signal
Rst_Debouncer: entity work.Debouncer
    PORT MAP( CLK => CLK,
              Sig => RST,
              Deb_Sig => deb_rst);

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

-- Connecting the generics and ports for the control and the datapath.
Control: entity work.Control
    GENERIC MAP ( data_size => data_size,
                  Matrix_A_Height => Matrix_A_Height,
                  Matrix_B_Width => Matrix_B_Width,
                  Matrix_Common => Matrix_Common)

    PORT MAP ( CLK => CLK,
               RST => deb_rst,
               NXT => deb_nxt,
               Address_ROM_A => Address_ROM_A,
               Address_ROM_B => Address_ROM_B,
               MACC_RST => MACC_RST,
               MACC_EN => MACC_EN,
               Address_RAM => Address_RAM,
               Write_EN => Write_EN);


Datapath: entity work.Datapath
    GENERIC MAP ( data_size => data_size,
                  Matrix_A_Height => Matrix_A_Height,
                  Matrix_B_Width => Matrix_B_Width,
                  Matrix_Common => Matrix_Common,
                  output_size => output_size)

    PORT MAP ( CLK => CLK,
               Address_ROM_A => Address_ROM_A,
               Address_ROM_B => Address_ROM_B,
               MACC_RST => MACC_RST,
               MACC_EN => MACC_EN,
               Address_RAM => Address_RAM,
               Write_EN => Write_EN,
               Output => Output);

end Behavioral;
```

## Datapath

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;
-- This is datapath of the parameterizable matrix-multiplier.
-- It connects the ROMs, MACC and RAM together.
-- It contains the combinational logic of the MACC containing:
-- the multiplication, addition and registers.
entity Datapath is -- The generics state the size of each matrix to be multiplied.
    Generic ( data_size : integer := 5; -- Number of bits per coefficient of the
                                         -- input matrices
              Matrix_A_Height : integer := 4; -- {H} The number of rows within input
                                               -- matrix A
              Matrix_B_Width : integer := 5; -- {N} The number of columns within
                                              -- input matrix B
              Matrix_Common : integer := 3; -- {M} The number of columns within
                                             -- input matrix A and number of rows
                                             -- within input matrix B
              output_size : integer := 7); -- The number of bits per coefficient of
                                            -- the output matrix

    Port ( CLK : in STD_LOGIC; -- Global clock input
           -- Address sizes of both ROMs determined by the log2 function of generic
           -- variables
           Address_ROM_A : in UNSIGNED (log2(Matrix_A_Height*Matrix_Common)-1
                                       downto 0);
           Address_ROM_B : in UNSIGNED (log2(Matrix_B_Width*Matrix_Common)-1
                                       downto 0);
           MACC_RST : in STD_LOGIC; -- Reset output of multiply-accumulate unit to
                                     -- zero
           MACC_EN : in STD_LOGIC; -- Enable of multiply-accumulate unit
           -- Address size of RAM determined by the log2 function of generic
           -- variables
           Address_RAM : in UNSIGNED (log2(Matrix_A_Height*Matrix_B_Width) -1
                                     downto 0);
           Write_EN : in STD_LOGIC; -- Enabling writing to the RAM
           Output : out SIGNED (output_size -1 downto 0)); -- The output of the RAM
end Datapath;

architecture Behavioral of Datapath is

-- Internal ROM outputs (data_size)-Bit data buses
signal ROM_A_Data_out, ROM_B_Data_out : SIGNED (data_size - 1 downto 0);
-- Internal output of MACC, size of bus is determined by largest
-- possible output of the two signed inputs.
signal MACC_Data_out : SIGNED (output_size - 1 downto 0);



begin

-- Multiply-accumulate unit (MACC) consists of a multiplier coupled
-- with an adder that accumulates the results by adding together the results of
-- the multiplication. Calculates data to be written to RAM.
MACC: process (CLK)
begin
   if (rising_edge(CLK)) then
      if (MACC_RST = '1') then -- synchronous reset
         MACC_Data_out <= (others => '0'); -- Sets output to 0
      elsif (MACC_EN = '1') then -- Enable signal
        -- Combinational logic for multiply-accumulate unit
        MACC_Data_out <= (ROM_A_Data_out * ROM_B_Data_out) + MACC_Data_out;
      end if;
   end if;
end process MACC;

-- Declaring the two ROMs and RAM
ROM_A: entity work.Async_ROM_12x5
    PORT MAP( Address => Address_ROM_A,
              DataOut => ROM_A_Data_out);

ROM_B: entity work.Async_ROM_15x5
    PORT MAP( Address => Address_ROM_B,
              DataOut => ROM_B_Data_out);

RAM: entity work.Param_RAM
    GENERIC MAP( Width => output_size,
                 -- The size of the RAM is determined by the generics
                 Depth => Matrix_A_Height*Matrix_B_Width)
    PORT MAP( CLK => CLK,
              Write_EN => Write_EN,
              Data_In => MACC_Data_out,
              Address => Address_RAM,
              Data_Out => Output);



end Behavioral;
```

## Control

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;
-- This is the control component for a fully parameterizable Matrix-Multiplier.
-- It contains a Mealy finite state machine,
-- combinational logic for the ROM/RAM addresses and counters.
entity Control is -- The generics state the size of each matrix to be multiplied.
    Generic ( data_size : integer := 5; -- Number of bits per coefficient of the
                                        -- input matrices
          Matrix_A_Height : integer := 4; -- {H} The number of rows within input
                                          -- matrix A
          Matrix_B_Width : integer := 5; -- {N} The number of columns within input
                                         -- matrix B
          Matrix_Common : integer := 3); -- {M} The number of columns within input
                                         -- matrix A and number of rows within
                                         -- input matrix B

    Port ( CLK : in STD_LOGIC; -- Global clock input
           RST : in STD_LOGIC; -- User input debounced Reset
           NXT : in STD_LOGIC; -- User input debounced Next
           -- Address sizes of both ROMs determined by the log2 function of generic
           -- variables
           Address_ROM_A : out UNSIGNED (log2(Matrix_A_Height*Matrix_Common)-1
                                         downto 0);
           Address_ROM_B : out UNSIGNED (log2(Matrix_B_Width*Matrix_Common)-1
                                         downto 0);
           MACC_RST : out STD_LOGIC; -- Reset output of multiply-accumulate unit to
                                     -- zero
           MACC_EN : out STD_LOGIC; -- Enable of multiply-accumulate unit
           -- Address size of RAM determined by the log2 function of generic
           -- variables
           Address_RAM : out UNSIGNED(log2(Matrix_A_Height*Matrix_B_Width) -1
                                      downto 0);
           Write_EN : out STD_LOGIC); -- Enabling writing to the RAM
end Control;

architecture Behavioral of Control is
    -- Defining a new type that contains all possible states of the FSM.
    type fsm_states is (RESET_STATE, CALCULATE_COEFFICIENT, STORE, DISPLAY);
    -- Creating two new signals of newly created 'fsm_states' type.
    signal state, next_state: fsm_states;
    -- Done is set high after final coefficient calculated.
    -- CNT_EM_M/N/H are the enables of each counter.
    signal done, CNT_EN_M, CNT_EN_N, CNT_EN_H : STD_LOGIC;
    -- Internal signals of clocks
    signal count_M_int : UNSIGNED (log2(Matrix_Common)-1 downto 0);
    signal count_N_int : UNSIGNED (log2(Matrix_B_Width)-1 downto 0);
    signal count_H_int : UNSIGNED (log2(Matrix_A_Height)-1 downto 0);


begin
    -- Declaring the three counters of variable sizes used for addressing
    -- RAM and ROM
    M_Counter: entity work.Param_Counter
    -- M counters size depends on generic Matrix_Common
    GENERIC MAP (LIMIT => Matrix_Common)
    PORT MAP( CLK => CLK,
              RST => RST,
              EN => CNT_EN_M,
              Count_Out => count_M_int);
```

```vhdl
    N_Counter: entity work.Param_Counter
    -- N counters size depends on generic Matrix_B_Width
    GENERIC MAP (LIMIT => Matrix_B_Width)
    PORT MAP( CLK => CLK,
              RST => RST,
              EN => CNT_EN_N,
              Count_Out => count_N_int);

    H_Counter: entity work.Param_Counter
    -- H counters size depends on generic Matrix_A_Height
    GENERIC MAP (LIMIT => Matrix_A_Height)
    PORT MAP( CLK => CLK,
              RST => RST,
              EN => CNT_EN_H,
              Count_Out => count_H_int);


    -- This is the process for the state register
    state_assignment: process (CLK) is
    begin
     if rising_edge(CLK) then
        if (RST = '1') then
            -- RESET_STATE is the reset state
            state <= RESET_STATE;
        else
            -- If reset is not pressed state gets next state
            state <= next_state;
        end if;
    end if;
    end process state_assignment;

    -- This is the process for the state transitions
    transitions: process (state, NXT, done, count_M_int) is
    begin
        case state is
           when RESET_STATE =>
           -- STATE DESCRIPTION
           -- Remains idle in this state until NXT is pressed
           -- Everything initially set to 0.
               if NXT = '1' then
                   next_state <= CALCULATE_COEFFICIENT;
               else
                   next_state <= state;
               end if;

           when CALCULATE_COEFFICIENT =>
           -- STATE DESCRIPTION
           -- Cycles through ROMs, calculating one full
           -- coefficient of the output matrix.
               if count_M_int = Matrix_Common-1 then
                   next_state <= STORE;
               else
                   next_state <= state;
               end if;

           when STORE =>
           -- STATE DESCRIPTION
           -- Writes the calculated coefficient to the RAM
               next_state <= DISPLAY;

           when DISPLAY =>
           -- STATE DESCRIPTION
           -- Displays the output of current coefficient calculated.
           -- Remains in this state until user presses NXT if not
           -- done. Else remains until RST.
               if NXT = '1' and done = '0' then
```

```
                   next_state <= CALCULATE_COEFFICIENT;
               else
                   next_state <= state;
               end if;
       end case;
   end process transitions;


-- Below is all the combinational logic determining the control signals.
-- They are based on current states, counter outputs and user inputs.

-- Done is the flag for the final coefficient being calculated.
-- Based on when all counters have reached their maximum values.
done <= '1' when count_M_int = Matrix_Common-1
            and count_N_int = Matrix_B_Width-1
            and count_H_int = Matrix_A_Height-1
            else '0';

-- Causes M_Counter to count to its maximum value every time the state is
-- CALCULATE_COEFFICIENT Is then incremented back to start value when NXT is pressed
-- and state is DISPLAY.
CNT_EN_M <= '1' when state = CALCULATE_COEFFICIENT and count_M_int < Matrix_Common-1
                else '1' when state = DISPLAY and done = '0' and NXT = '1'
                else '0';

-- Controlled by the user pressing NXT when in DISPLAY state.
CNT_EN_N <= '1' when state = DISPLAY and done = '0' and NXT = '1'
                else '0';

-- Controlled by the user pressing NXT when in DISPLAY state and
-- when other two counters are at maximum value.
CNT_EN_H <= '1' when state = DISPLAY and done = '0' and NXT = '1'
                and count_M_int = Matrix_Common-1
                and count_N_int = Matrix_B_Width-1
                else '0';

-- Used to remove previous data from the MACC output.
-- Needed prior to each new coefficient calculated or in RST.
MACC_RST <= '1' when state = RESET_STATE
                else '1' when state = DISPLAY
                else '0';

-- Enabling MACC to calculate values. Only in state CALCULATE_COEFFICIENT
MACC_EN <= '1' when state = CALCULATE_COEFFICIENT
               else '0';

-- Enabling writing of data to RAM, only in STORE state.
Write_EN <= '1' when state = STORE
                else '0';

-- All below address buses are calculated through combinational logic.
-- As a function of counters and generic values.
-- Combinational logic of RAM
Address_RAM <= RESIZE(((TO_UNSIGNED(Matrix_B_Width,
log2(Matrix_B_Width)))*count_H_int) + count_N_int,
log2(Matrix_A_Height*Matrix_B_Width));

-- Combinational logic for addresses of A and B ROM
Address_ROM_A <= RESIZE(((TO_UNSIGNED(Matrix_Common,
log2(Matrix_Common)))*count_H_int) + count_M_int,
log2(Matrix_A_Height*Matrix_Common));
Address_ROM_B <= RESIZE(((TO_UNSIGNED(Matrix_B_Width,
log2(Matrix_B_Width)))*count_M_int) + count_N_int,
log2(Matrix_B_Width*Matrix_Common));


end Behavioral;
```

## Asynchronous ROM A – Async_ROM_12x5

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
-- This is a ROM component to be used within the Parameterizable Matrix-Multiplier.
-- It is made up of an array of integers for ease of reading. This is converted
-- to signed binary numbers in synthesis.
entity Async_ROM_12x5 is
    Port ( Address : in UNSIGNED (3 downto 0);
           DataOut : out SIGNED (4 downto 0));
end Async_ROM_12x5;

architecture Behavioral of Async_ROM_12x5 is

type ROM_Array is array (0 to 11) of integer;
    constant Content: ROM_Array := (
        -- This is used as Matrix A. So to test functionality
        -- of the Matrix-Multiplier the first three numbers
        -- are set to -16. This will correspond to values chosen
        -- in the other ROM to test the limits of the design.
        0 => -16,
        1 => -16,
        2 => -16,
        3 => 4,
        4 => 5,
        5 => 6,
        6 => 7,
        7 => 8,
        8 => 9,
        9 => 10,
        10 => 11,
        11 => 12,
        others => 0);
begin
        -- Converting the integers used above to signed binary values.
        DataOut <= TO_SIGNED(Content(TO_INTEGER(Address)),5);

end Behavioral;
```

## Asynchronous ROM B – Async_ROM_15x5

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
-- This is a ROM component to be used within the Parameterizable Matrix-Multiplier.
-- It is made up of an array of integers for ease of reading. This is converted
-- to signed binary numbers in synthesis.
entity Async_ROM_15x5 is
    Port ( Address : in UNSIGNED (3 downto 0);
           DataOut : out SIGNED (4 downto 0));
end Async_ROM_15x5;

architecture Behavioral of Async_ROM_15x5 is

type ROM_Array is array (0 to 14) of integer;
    constant Content: ROM_Array := (
        -- This is used as Matrix B. So to test functionality
        -- of the Matrix-Multiplier locations 0, 5 and 10 contain the value
        -- -16. This will correspond to values chosen  in the other ROM to
        -- produce the largest positive number possible. Locations 1, 6 and 11
        -- contain 15 to be multiplied with -16 within the other ROM to produce the
        -- largest negative number.
        0 => -16,
        1 => 15,
        2 => 0,
        3 => 4,
        4 => 5,
        5 => -16,
        6 => 15,
        7 => 0,
        8 => 9,
        9 => 10,
        10 => -16,
        11 => 15,
        12 => 0,
        13 => 14,
        14 => 15,
        others => 0);
begin
        -- Converting the integers used above to signed binary values.
        DataOut <= TO_SIGNED(Content(TO_INTEGER(Address)),5);

end Behavioral;
```

## Parameterizable RAM

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;
-- This is a Synchronous write / asynchronous read parameterizable single-port RAM
entity Param_RAM is
    Generic ( Width : integer := 4; -- Datasize in bits
              Depth : integer := 5); -- Matrix_A_Height * Matrix_B_Width
    Port ( CLK : in  STD_LOGIC;
           Write_EN : in  STD_LOGIC;
           Data_In : in  SIGNED (Width -1 downto 0);
           Address : in  UNSIGNED (log2(Depth)-1 downto 0);
           Data_Out : out  SIGNED (Width -1 downto 0));
end Param_RAM;

architecture Behavioral of Param_RAM is

type ram_type is array (0 to Depth -1) of SIGNED(Width -1 downto 0);

signal ram_inst: ram_type;

begin

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

  -- 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;

end Behavioral;
```

## Self-checking VHDL Testbench

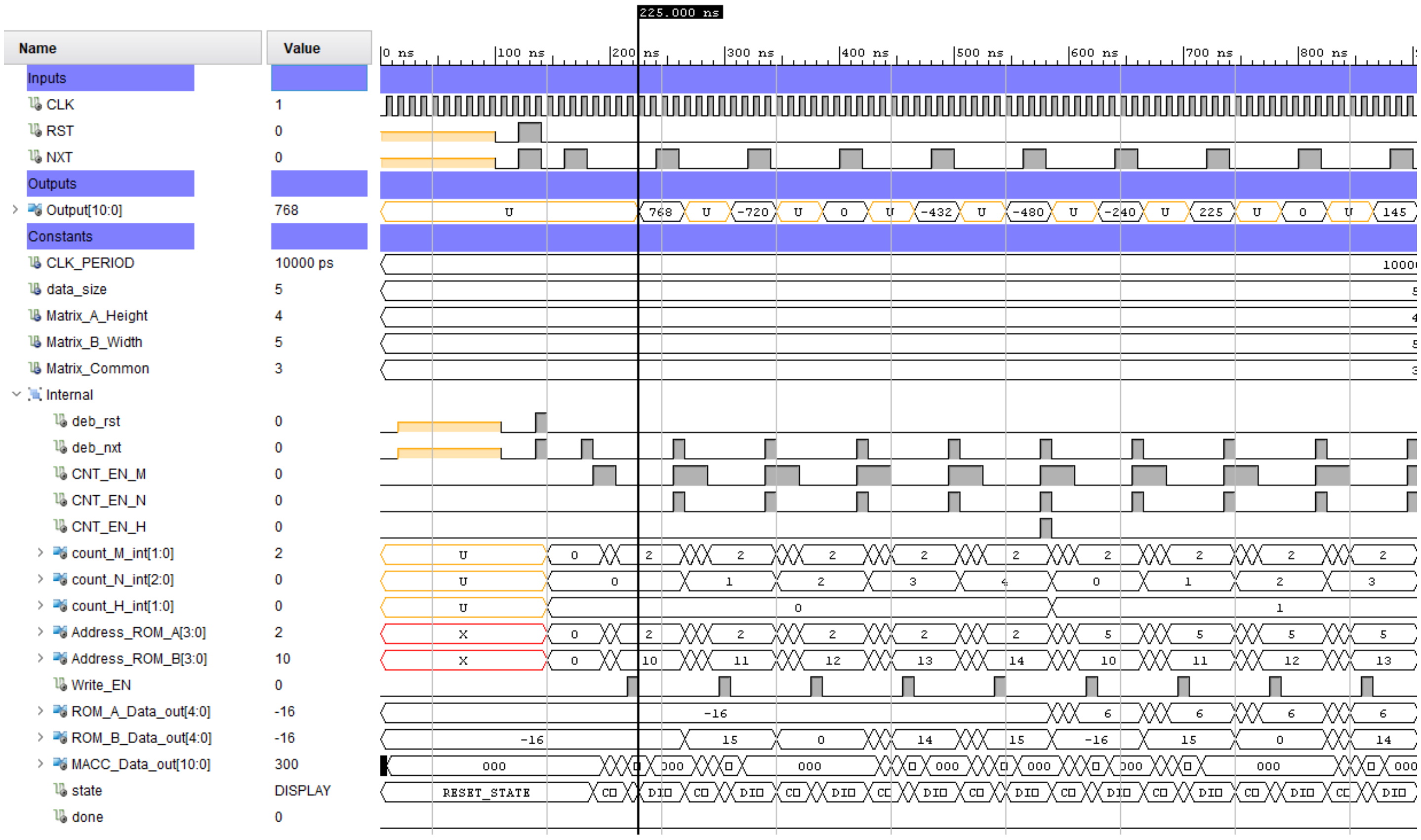
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;
-- This is the test-bench of the Matrix_Multiplier. It is a parameterizable,
-- self checking test-bench. It uses a record containing pre-calculated expected
-- outputs and their corresponding output-matrix locations. The test-bench will
-- report when there are errors and also when the circuit works as expected.

entity Matrix_Multiplier_tb is

end Matrix_Multiplier_tb;

architecture Behavioral of Matrix_Multiplier_tb is

constant CLK_PERIOD : time := 10 ns; -- Defines the standard clock time
-- Constants that will be remapped to Matrix_Multiplier generics
constant data_size : integer := 5; -- Number of bits per coefficient of the input
matrices
constant Matrix_A_Height : integer := 4; -- {H} The number of rows within input
matrix A
constant Matrix_B_Width : integer := 5; -- {N} The number of columns within input
matrix B
constant Matrix_Common : integer := 3; -- {M} The number of columns within input
matrix A
                                       -- and number of rows withing input matrix B

-- The size of the output buss is determined by the maximum value possible to
represent using a
-- binary number of data_size wide multiplied by itself Matrix_Common times. Here it
is
-- declared as a constant to be used throughout other components.
constant output_size : integer := (size(Matrix_Common * ((2**(data_size - 1))**2)))
+1;

signal CLK : STD_LOGIC; -- Global clock input
signal RST : STD_LOGIC; -- User input debounced Reset
signal NXT : STD_LOGIC; -- User input debounced Next
signal Output : SIGNED (output_size -1 downto 0); -- Output of RAM

-- Record declared that contains Column/Row integers used to identify coefficient
-- position within RAM and Output_rcd which is pre-calculated expected output of
RAM.
-- These a grouped together under the name output_coefficient
type output_coefficient is record
    column : integer;
    row : integer;
    output_rcd : SIGNED (output_size-1 downto 0);
end record;

-- Declaring a type of array made up of the output_coefficient type called
-- results_array
type results_array is array
    (natural range <>) of output_coefficient;

-- Creating a results_array called results and filling it
-- with a completed output matrix of correct expected outputs
-- and their corresponding positions within that matrix. These
-- were calculated using MatLab. It contains one erroneous output.
constant results : results_array := (
    -- Column, Row,      Output_RCD
    -- Testing maximum positive number.
    (0,          0,      TO_SIGNED(768, output_size)),
    -- Testing maximum negative number, uses MSB.
    (1,          0,      TO_SIGNED(-720, output_size)),
    -- Testing 0 output.
    (2,          0,      TO_SIGNED(0, output_size)),
    -- Further sets of inputs to check circuit is operating correctly
    (3,          0,      TO_SIGNED(-432, output_size)),
    (4,          0,      TO_SIGNED(-480, output_size)),
    (0,          1,      TO_SIGNED(-240, output_size)),
    (1,          1,      TO_SIGNED(225, output_size)),
    (2,          1,      TO_SIGNED(0, output_size)),
    (3,          1,      TO_SIGNED(145, output_size)),
    (4,          1,      TO_SIGNED(160, output_size)),
    (0,          2,      TO_SIGNED(-384, output_size)),
    (1,          2,      TO_SIGNED(360, output_size)),
    (2,          2,      TO_SIGNED(0, output_size)),
    (3,          2,      TO_SIGNED(226, output_size)),
    (4,          2,      TO_SIGNED(250, output_size)),
    (0,          3,      TO_SIGNED(-528, output_size)),
    (1,          3,      TO_SIGNED(495, output_size)),
    (2,          3,      TO_SIGNED(0, output_size)),
    (3,          3,      TO_SIGNED(307, output_size)),
    -- The final result is erroneous to test error checking
    -- functionality of test-bench.
    (4,          3,      TO_SIGNED(342, output_size)));


begin
UUT: entity work.Matrix_Multiplier
    -- Remapping generic and port maps with testbench constants and signals
    GENERIC MAP ( data_size => data_size,
                  Matrix_A_Height => Matrix_A_Height,
                  Matrix_B_Width => Matrix_B_Width,
                  Matrix_Common => Matrix_Common)
    PORT MAP ( CLK => CLK,
               RST => RST,
               NXT => NXT,
               Output => Output);

-- Clock process
clk_process :process
begin
    clk <= '0';
    wait for CLK_PERIOD/2;
    clk <= '1';
    wait for CLK_PERIOD/2;
end process;


test : process
begin
    wait for 100 ns;
    wait until falling_edge(CLK);

    -- Needed for resetting debouncers
    -- All inputs need to last for 2*CLK_PERIOD's
    -- for the debouncer to accept them as inputs
    -- and outputs the signals to the counter.
    RST <= '0';
    NXT <= '0';
    wait for CLK_PERIOD*2;
    RST <= '1';
    NXT <= '1';
    wait for CLK_PERIOD*2;
```

```vhdl
    RST <= '0';
    NXT <= '0';
    wait for CLK_PERIOD*2;

    -- Looping through the results_array - results.
    for i in results'range loop
    NXT <= '1'; -- NXT is pressed to begin the calculation of a coefficient
    wait for CLK_PERIOD*2;
    NXT <= '0';

    wait for CLK_PERIOD*6; -- Waiting 6 Clock cycles between DISPLAY states.

    assert (output = results(i).output_rcd) -- comparing expected and actual values
    -- If assert not true report occurs displaying expected and actual values
    report "Test failed for column: {" & integer'image(results(i).column)
            & "} and row: {" & integer'image(results(i).row) &
           "} Expected output = {" &
            integer'image(to_integer(results(i).output_rcd)) &
           "} Actual output = {" & integer'image(to_integer(Output)) & "}"
    severity error;

    -- If first report doesn't occur test was successful and this report occurs
    -- displaying expected and actual value.
    assert (output /= results(i).output_rcd)
    report "Test, no errors for column: {" & integer'image(results(i).column)
            & "} and row: {" & integer'image(results(i).row) &
           "} Expected output = {" &
            integer'image(to_integer(results(i).output_rcd)) &
           "} Actual output = {" & integer'image(to_integer(Output)) & "}"
    severity note;
    end loop;

    -- Resetting after full matrix calculated
    RST <= '1';
    wait for CLK_PERIOD*2;
    RST <= '0';

    -- Running through loop again after reset
    for i in results'range loop
    NXT <= '1'; -- NXT is pressed to begin the calculation of a coefficient
    wait for CLK_PERIOD*2;
    NXT <= '0';

    wait for CLK_PERIOD*6; -- Waiting 6 Clock cycles between DISPLAY states.

    assert (output = results(i).output_rcd) -- comparing expected and actual values
    -- If assert not true report occurs displaying expected and actual values
    report "Test failed for column: {" & integer'image(results(i).column)
            & "} and row: {" & integer'image(results(i).row) &
           "} Expected output = {" &
            integer'image(to_integer(results(i).output_rcd)) &
           "} Actual output = {" & integer'image(to_integer(Output)) & "}"
    severity error;

    -- If first report doesn't occur test was successful and this report occurs
    -- displaying expected and actual value.
    assert (output /= results(i).output_rcd)
    report "Test, no errors for column: {" & integer'image(results(i).column)
            & "} and row: {" & integer'image(results(i).row) &
           "} Expected output = {" &
            integer'image(to_integer(results(i).output_rcd)) &
           "} Actual output = {" & integer'image(to_integer(Output)) & "}"
    severity note;
    end loop;
    wait;

end process;
end Behavioral;
```

## Simulation Output

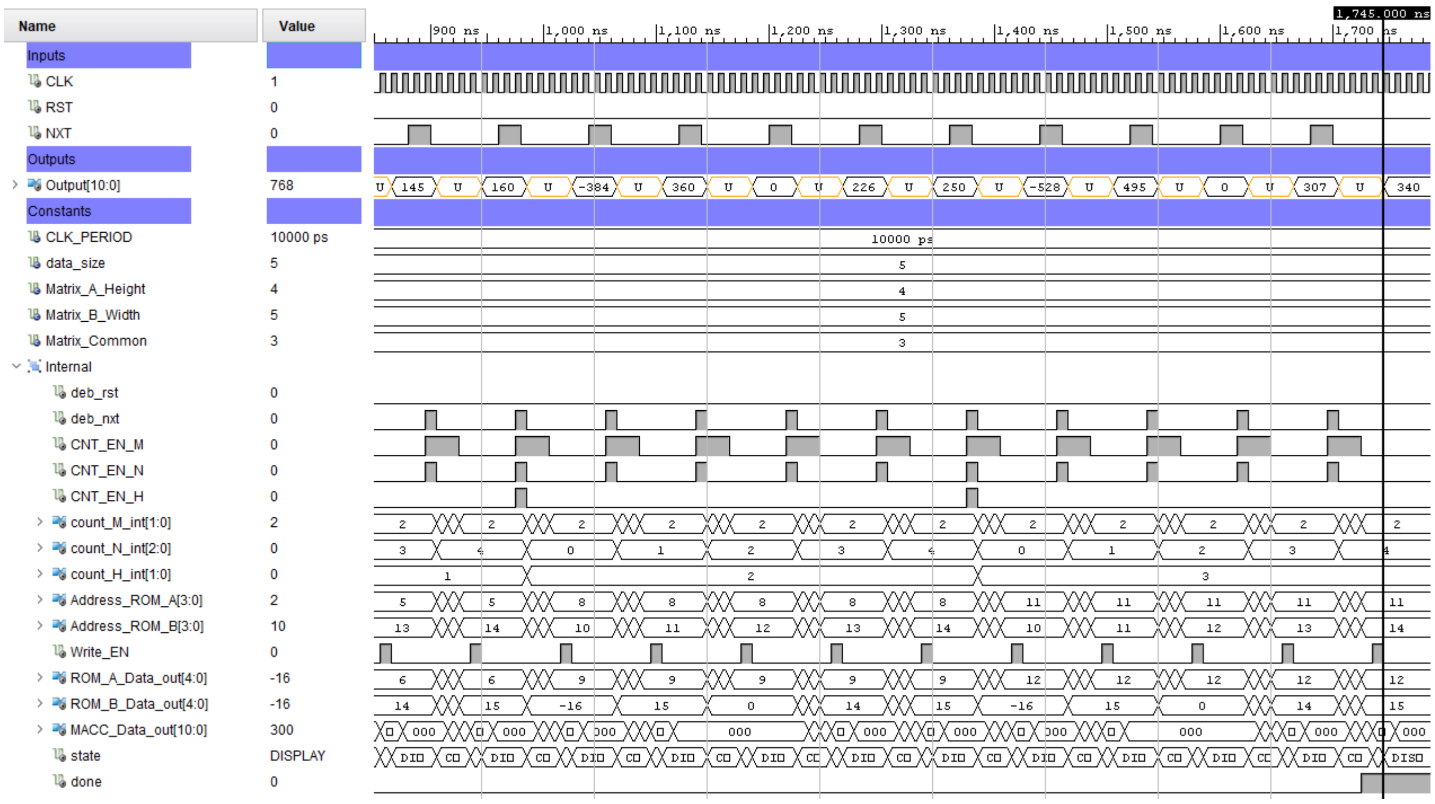
### Final value of every coefficient of the product matrix

Marker 225.000ns – First DISPLAY state with output 768.



| Name | Value |
|---|---|
| Inputs | |
| CLK | 1 |
| RST | 0 |
| NXT | 0 |
| Outputs | |
| Output[10:0] | 768 |
| Constants | |
| CLK_PERIOD | 10000 ps |
| data_size | 5 |
| Matrix_A_Height | 4 |
| Matrix_B_Width | 5 |
| Matrix_Common | 3 |
| Internal | |
| deb_rst | 0 |
| deb_nxt | 0 |
| CNT_EN_M | 0 |
| CNT_EN_N | 0 |
| CNT_EN_H | 0 |
| count_M_int[1:0] | 2 |
| count_N_int[2:0] | 0 |
| count_H_int[1:0] | 0 |
| Address_ROM_A[3:0] | 2 |
| Address_ROM_B[3:0] | 10 |
| Write_EN | 0 |
| ROM_A_Data_out[4:0] | -16 |
| ROM_B_Data_out[4:0] | -16 |
| MACC_Data_out[10:0] | 300 |
| state | DISPLAY |
| done | 0 |

Marker 1,745.000ns – Final DISPLAY state with output 340. Done signal goes high.



| Name | Value |
|---|---|
| Inputs | |
| CLK | 1 |
| RST | 0 |
| NXT | 0 |
| Outputs | |
| Output[10:0] | 768 |
| Constants | |
| CLK_PERIOD | 10000 ps |
| data_size | 5 |
| Matrix_A_Height | 4 |
| Matrix_B_Width | 5 |
| Matrix_Common | 3 |
| Internal | |
| deb_rst | 0 |
| deb_nxt | 0 |
| CNT_EN_M | 0 |
| CNT_EN_N | 0 |
| CNT_EN_H | 0 |
| count_M_int[1:0] | 2 |
| count_N_int[2:0] | 0 |
| count_H_int[1:0] | 0 |
| Address_ROM_A[3:0] | 2 |
| Address_ROM_B[3:0] | 10 |
| Write_EN | 0 |
| ROM_A_Data_out[4:0] | -16 |
| ROM_B_Data_out[4:0] | -16 |
| MACC_Data_out[10:0] | 300 |
| state | DISPLAY |
| done | 0 |

## Full cycle of computation for first and last product matrix coefficients

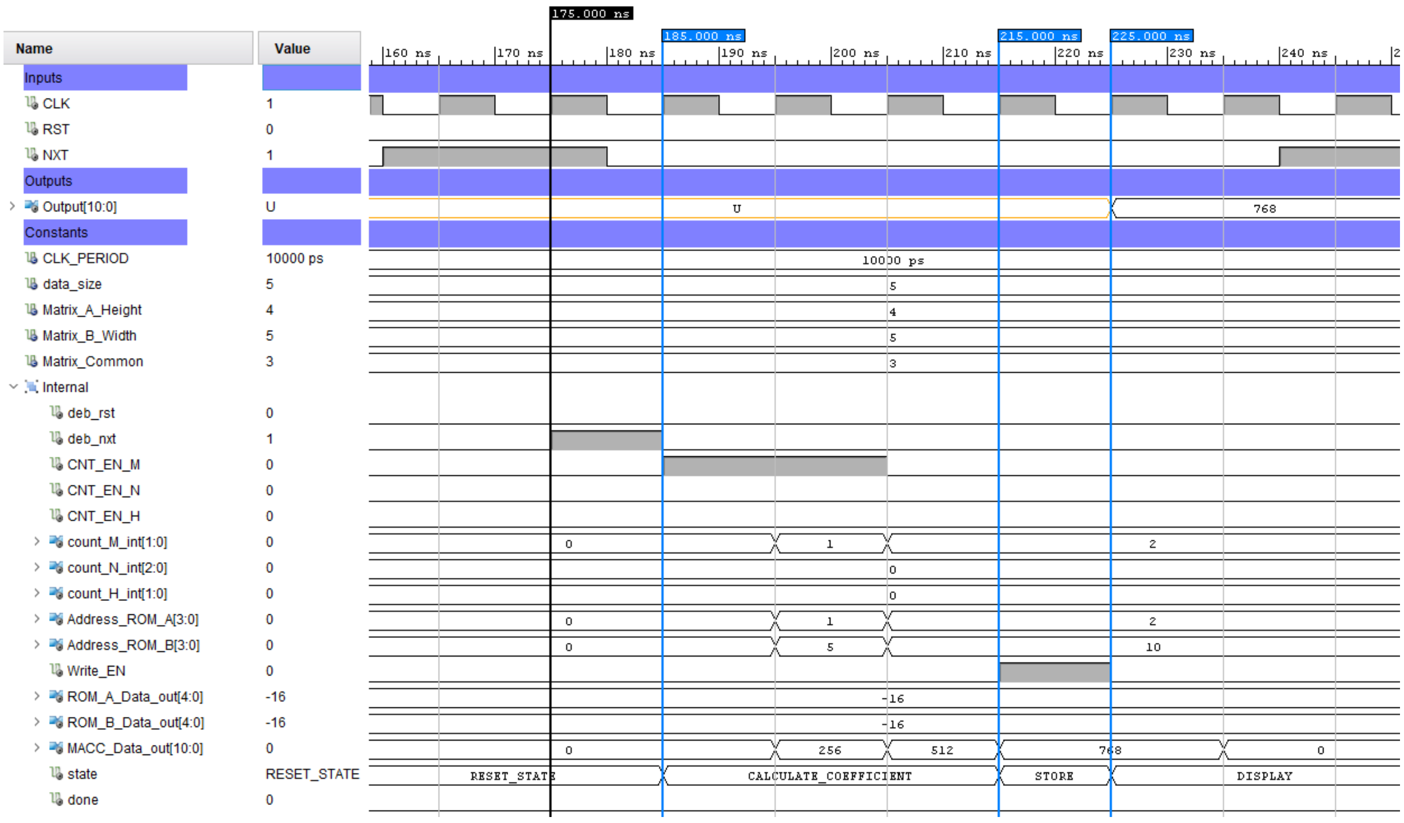
### First

Marker 175.000ns – deb_nxt goes high

Marker 185.000ns – State CALCULATE_COEFFICIENT

Marker 215.000ns – State STORE

Marker 225.000ns – State DISPLAY



| Name | Value |
|---|---|
| Inputs | |
| CLK | 1 |
| RST | 0 |
| NXT | 1 |
| Outputs | |
| Output[10:0] | U |
| Constants | |
| CLK_PERIOD | 10000 ps |
| data_size | 5 |
| Matrix_A_Height | 4 |
| Matrix_B_Width | 5 |
| Matrix_Common | 3 |
| Internal | |
| deb_rst | 0 |
| deb_nxt | 1 |
| CNT_EN_M | 0 |
| CNT_EN_N | 0 |
| CNT_EN_H | 0 |
| count_M_int[1:0] | 0 |
| count_N_int[2:0] | 0 |
| count_H_int[1:0] | 0 |
| Address_ROM_A[3:0] | 0 |
| Address_ROM_B[3:0] | 0 |
| Write_EN | 0 |
| ROM_A_Data_out[4:0] | -16 |
| ROM_B_Data_out[4:0] | -16 |
| MACC_Data_out[10:0] | 0 |
| state | RESET_STATE |
| done | 0 |

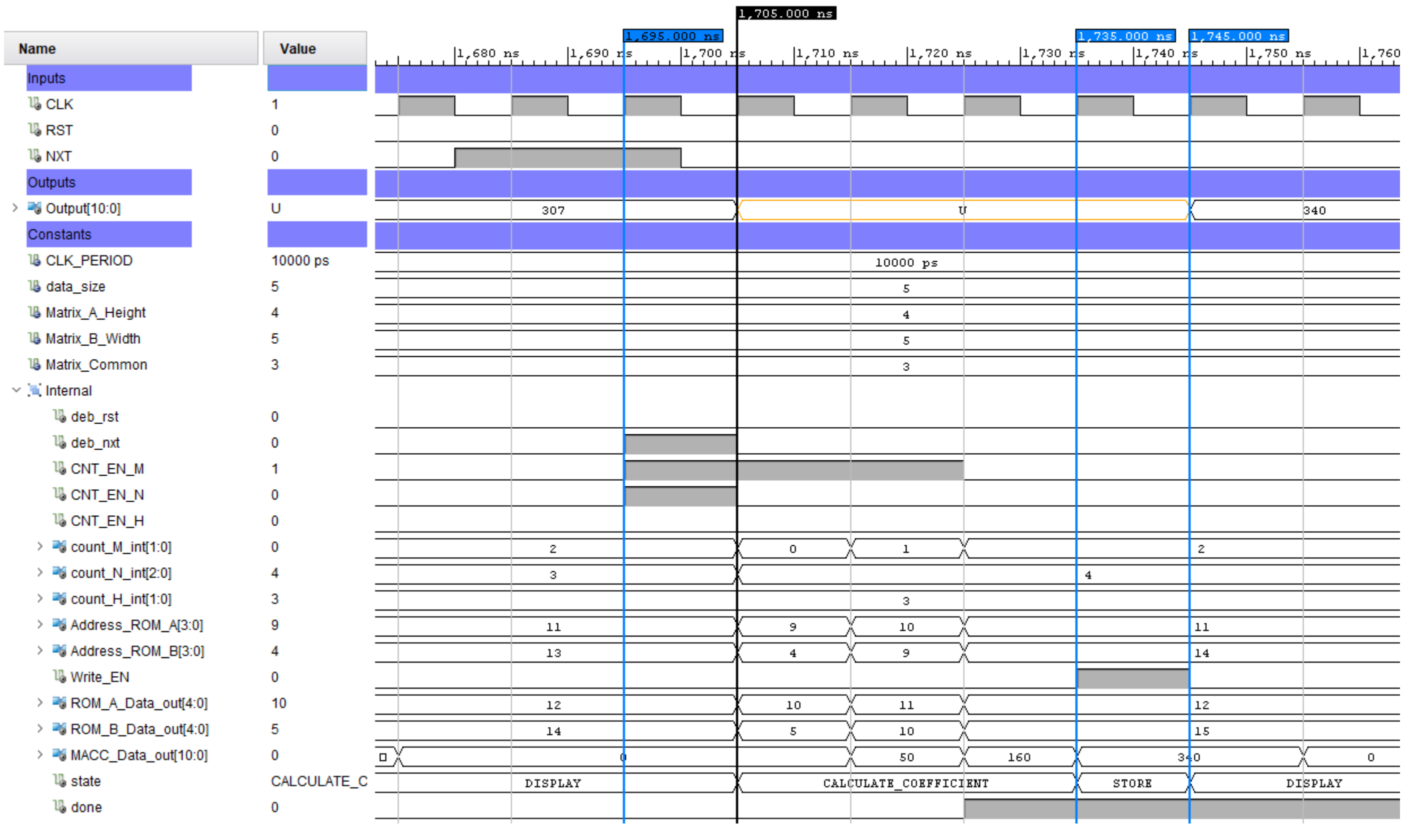
## Last

Marker 1,695.00ns – deb_nxt goes high

Marker 1,705.000ns – State CALCULATE_COEFFICIENT

Marker 1,735.000ns – State STORE

Marker 1,745.00ns – State DISPLAY



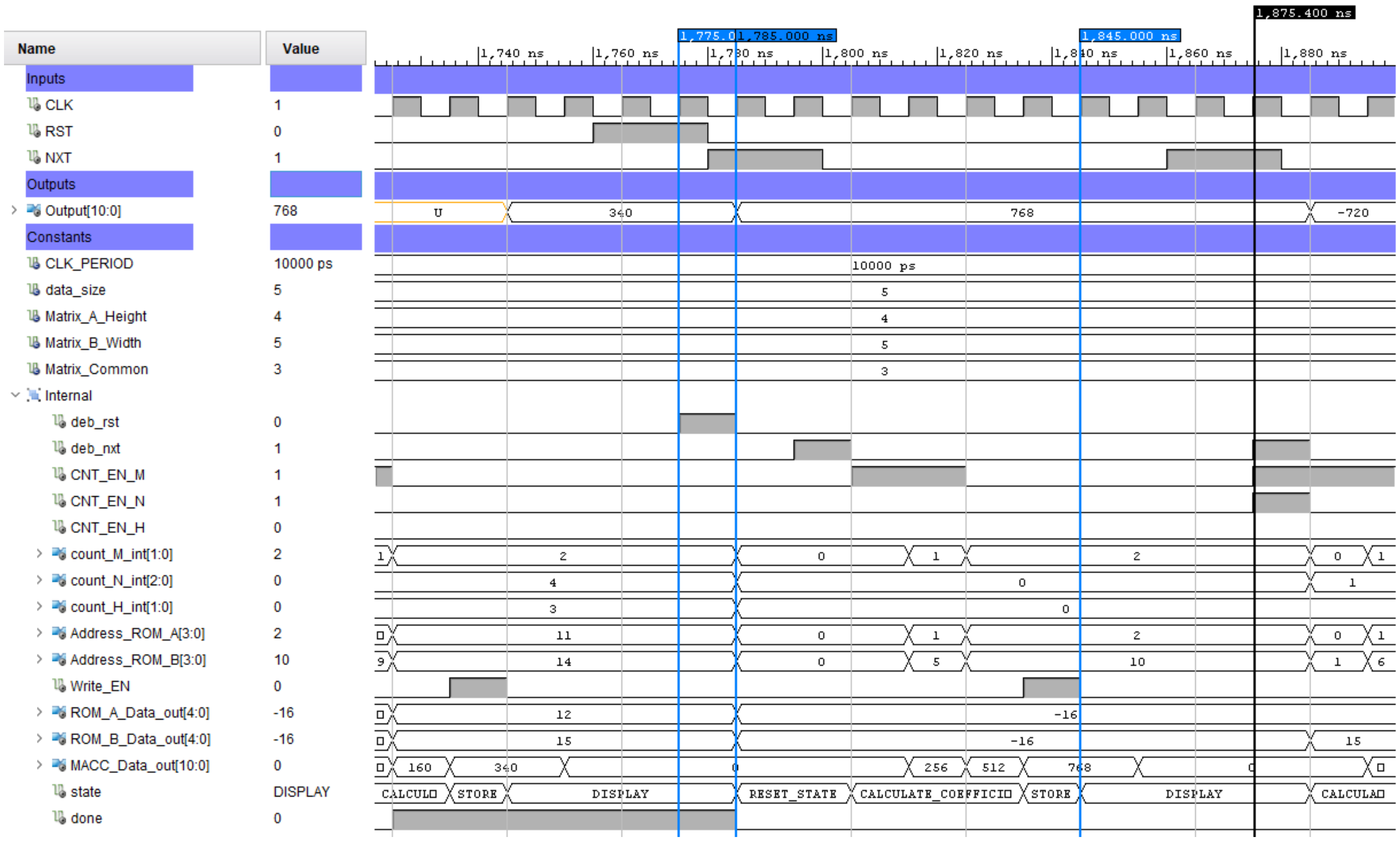
## Testing return to RESET_STATE

Marker 1,750.000ns – deb_rst goes high

Marker 1,785.000ns – State RESET_STATE

Marker 1,845.000ns – State DISPLAY

Marker 1,875.400ns – deb_nxt goes high



| Name | Value |
|---|---|
| Inputs | |
| CLK | 1 |
| RST | 0 |
| NXT | 1 |
| Outputs | |
| Output[10:0] | 768 |
| Constants | |
| CLK_PERIOD | 10000 ps |
| data_size | 5 |
| Matrix_A_Height | 4 |
| Matrix_B_Width | 5 |
| Matrix_Common | 3 |
| Internal | |
| deb_rst | 0 |
| deb_nxt | 1 |
| CNT_EN_M | 1 |
| CNT_EN_N | 1 |
| CNT_EN_H | 0 |
| count_M_int[1:0] | 2 |
| count_N_int[2:0] | 0 |
| count_H_int[1:0] | 0 |
| Address_ROM_A[3:0] | 2 |
| Address_ROM_B[3:0] | 10 |
| Write_EN | 0 |
| ROM_A_Data_out[4:0] | -16 |
| ROM_B_Data_out[4:0] | -16 |
| MACC_Data_out[10:0] | 0 |
| state | DISPLAY |
| done | 0 |

## Console displaying the confirmation messages

Showing two full run cycles separated by a RST. The final output on each cycle has an erroneous expected output to test the error message functionality.

```
INFO: [Simtcl 6-17] Simulation restarted

run 100 us

Note: Test, no errors for column: {0} and row: {0} Expected output = {768} Actual output = {768}

Time: 240 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {1} and row: {0} Expected output = {-720} Actual output = {-720}

Time: 320 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {2} and row: {0} Expected output = {0} Actual output = {0}

Time: 400 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {3} and row: {0} Expected output = {-432} Actual output = {-432}

Time: 480 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {4} and row: {0} Expected output = {-480} Actual output = {-480}

Time: 560 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {0} and row: {1} Expected output = {-240} Actual output = {-240}

Time: 640 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {1} and row: {1} Expected output = {225} Actual output = {225}

Time: 720 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {2} and row: {1} Expected output = {0} Actual output = {0}

Time: 800 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {3} and row: {1} Expected output = {145} Actual output = {145}

Time: 880 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {4} and row: {1} Expected output = {160} Actual output = {160}

Time: 960 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {0} and row: {2} Expected output = {-384} Actual output = {-384}
```

```
Time: 1040 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {1} and row: {2} Expected output = {360} Actual output = {360}

Time: 1120 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {2} and row: {2} Expected output = {0} Actual output = {0}

Time: 1200 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {3} and row: {2} Expected output = {226} Actual output = {226}

Time: 1280 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {4} and row: {2} Expected output = {250} Actual output = {250}

Time: 1360 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {0} and row: {3} Expected output = {-528} Actual output = {-528}

Time: 1440 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {1} and row: {3} Expected output = {495} Actual output = {495}

Time: 1520 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {2} and row: {3} Expected output = {0} Actual output = {0}

Time: 1600 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {3} and row: {3} Expected output = {307} Actual output = {307}

Time: 1680 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Error: Test failed for column: {4} and row: {3} Expected output = {342} Actual output = {340}

Time: 1760 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {0} and row: {0} Expected output = {768} Actual output = {768}

Time: 1860 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {1} and row: {0} Expected output = {-720} Actual output = {-720}

Time: 1940 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {2} and row: {0} Expected output = {0} Actual output = {0}

Time: 2020 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {3} and row: {0} Expected output = {-432} Actual output = {-432}
```

```
Time: 2100 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {4} and row: {0} Expected output = {-480} Actual output = {-480}

Time: 2180 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {0} and row: {1} Expected output = {-240} Actual output = {-240}

Time: 2260 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {1} and row: {1} Expected output = {225} Actual output = {225}

Time: 2340 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {2} and row: {1} Expected output = {0} Actual output = {0}

Time: 2420 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {3} and row: {1} Expected output = {145} Actual output = {145}

Time: 2500 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {4} and row: {1} Expected output = {160} Actual output = {160}

Time: 2580 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {0} and row: {2} Expected output = {-384} Actual output = {-384}

Time: 2660 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {1} and row: {2} Expected output = {360} Actual output = {360}

Time: 2740 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {2} and row: {2} Expected output = {0} Actual output = {0}

Time: 2820 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {3} and row: {2} Expected output = {226} Actual output = {226}

Time: 2900 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {4} and row: {2} Expected output = {250} Actual output = {250}

Time: 2980 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {0} and row: {3} Expected output = {-528} Actual output = {-528}

Time: 3060 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {1} and row: {3} Expected output = {495} Actual output = {495}
```

```
Time: 3140 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {2} and row: {3} Expected output = {0} Actual output = {0}

Time: 3220 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Note: Test, no errors for column: {3} and row: {3} Expected output = {307} Actual output = {307}

Time: 3300 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd

Error: Test failed for column: {4} and row: {3} Expected output = {342} Actual output = {340}

Time: 3380 ns  Iteration: 0  Process: /Matrix_Multiplier_tb/test  File:
M:/VivadoProjects/Final_Project/Final_Project.srcs/sim_1/new/Matrix_Multiplier_tb.vhd
```

## 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 :
	  15 Input      5 Bit        Muxes := 1
	   2 Input      3 Bit        Muxes := 1
	   2 Input      2 Bit        Muxes := 2
	   4 Input      2 Bit        Muxes := 1
	   5 Input      1 Bit        Muxes := 1
	   3 Input      1 Bit        Muxes := 1
	   2 Input      1 Bit        Muxes := 3
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```

## Bibliography

Tempesti, D. G. (2018). Project Script. Retrieved from https://bit.ly/2S5PmXM