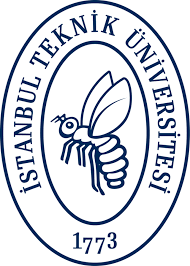
****

**DIGITAL SYSTEM DESIGN APPLICATION – EHB 436E**

**Experiment VII**

**Yiğit Bektaş GÜRSOY**

**040180063**

**Class Lecturer: Sıddıka Berna Örs Yalçın**

**Class Assistant:  
Serdar Duran  
Yasin Fırat Kula  
Mehmet Onur Demirtürk**

1. **STRUCTURAL MULTIPLIER - UNSIGNED**

* Verilog Code

**module** MULTS

**(**

**input** **[**7**:**0**]** A**,**

**input** **[**7**:**0**]** X**,**

**output** **[**15**:**0**]** result

**);**

**wire** **[**15**:**0**]**PP**[**7**:**0**];**

**genvar** i**;**

**generate**

**for(**i **=** 0**;** i **<=** 7**;** i **=** i **+** 1**)**

**begin:** gen\_PP //PARTIAL PRODUCT

**assign** PP**[**i**][**15**:**0**]** **=** **(**X**[**i**]** **\*** A**)** **<<** i**;**

**end**

**endgenerate**

**wire** **[**15**:**0**]** sum **[**6**:**0**];**

**wire** cout **[**6**:**0**];**

// sum of partial product

CLA CLA1**(**PP**[**0**][**15**:**0**],** PP**[**1**][**15**:**0**],** 0**,** cout**[**0**],** sum**[**0**][**15**:**0**]);**

CLA CLA2**(**PP**[**2**][**15**:**0**],** PP**[**3**][**15**:**0**],** 0**,** cout**[**1**],** sum**[**1**][**15**:**0**]);**

CLA CLA3**(**PP**[**4**][**15**:**0**],** PP**[**5**][**15**:**0**],** 0**,** cout**[**2**],** sum**[**2**][**15**:**0**]);**

CLA CLA4**(**PP**[**6**][**15**:**0**],** PP**[**7**][**15**:**0**],** 0**,** cout**[**3**],** sum**[**3**][**15**:**0**]);**

// sum of result of partial product

CLA CLA5**(**sum**[**0**][**15**:**0**],** sum**[**1**][**15**:**0**],** 0**,** cout**[**4**],** sum**[**4**][**15**:**0**]);**

CLA CLA6**(**sum**[**2**][**15**:**0**],** sum**[**3**][**15**:**0**],** 0**,** cout**[**5**],** sum**[**5**][**15**:**0**]);**

CLA CLA7**(**sum**[**4**][**15**:**0**],** sum**[**5**][**15**:**0**],** 0**,** cout**[**6**],** sum**[**6**][**15**:**0**]);**

**assign** result**[**15**:**0**]** **=** sum**[**6**][**15**:**0**];**

**endmodule**

* Test Bench Code

**module** MULTS\_tb**();**

**reg** **[**7**:**0**]** A**;**

**reg** **[**7**:**0**]** X**;**

**wire** **[**15**:**0**]** result**;**

MULTS DUT

**(**

**.**A**(**A**),**

**.**X**(**X**),**

**.**result**(**result**)**

**);**

**initial**

**begin**

A**=**0**;** X**=**0**;**

**#**10**;**

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**8**;** X**=**14**;**

**#**15

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**13**;** X**=**6**;**

**#**20

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**2**;** X**=**11**;**

**#**10

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**36**;** X**=**82**;**

**#**15

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**4**;** X**=**75**;**

**#**20

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**121**;** X**=**139**;**

**#**10

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**194**;** X**=**237**;**

**#**15

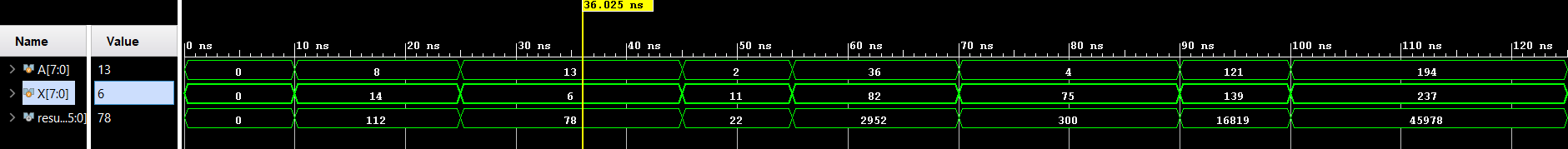
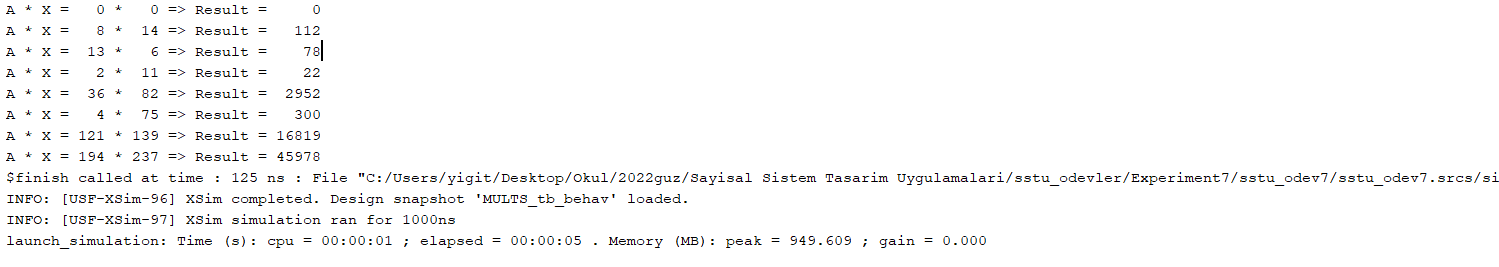
$write**(**"A \* X = %d \* %d => Result = %d\n"**,** A**,** X**,** result**);**

**#**10

$finish**;**

**end**

**endmodule**

* Behavioral Simulation Result
* TCL Console Output

1. **STRUCTURAL MULTIPLIER – SIGNED**

* Verilog Code

**module** MULTS\_signed

**(**

**input** **[**7**:**0**]** A**,**

**input** **[**7**:**0**]** X**,**

**output** **[**15**:**0**]** result

**);**

**wire** **[**7**:**0**]** PP **[**7**:**0**];**

**wire** **[**7**:**0**]** PP\_BW **[**7**:**0**];**

**wire** **[**15**:**0**]** PP\_shift **[**7**:**0**];**

**genvar** i**;**

**generate**

**for(**i **=** 0**;** i **<=** 7**;** i **=** i **+** 1**)**

**begin:** PP\_LOOP

**assign** PP**[**i**][**7**:**0**]** **=** **(**X**[**i**]** **\*** A**);**

**end**

**endgenerate**

**genvar** z**,** t**;**

**generate**

**for(**z **=** 0**;** z **<=** 7**;** z **=** z **+** 1**)**

**begin:** PP\_BW\_LOOP

**for(**t **=** 0**;** t **<=** 7**;** t **=** t **+** 1**)**

**begin**

**if(**z **!=** 7**)**

**begin**

**if(**t **!=** 7**)**

**assign** PP\_BW**[**z**][**t**]** **=** PP**[**z**][**t**];**

**else**

**assign** PP\_BW**[**z**][**t**]** **=** **~**PP**[**z**][**t**];**

**end**

**else**

**begin**

**if(**t **==** 7**)**

**assign** PP\_BW**[**z**][**t**]** **=** PP**[**z**][**t**];**

**else**

**assign** PP\_BW**[**z**][**t**]** **=** **~**PP**[**z**][**t**];**

**end**

**end**

**end**

**endgenerate**

**genvar** j**;**

**generate**

**for(**j **=** 0**;** j **<=** 7**;** j **=** j **+** 1**)**

**begin:** PP\_SHIFT\_LOOP

**assign** PP\_shift**[**j**][**15**:**0**]** **=** PP\_BW**[**j**]** **<<** j**;**

**end**

**endgenerate**

**wire** **[**15**:**0**]** sum **[**7**:**0**];**

**wire** cout **[**7**:**0**];**

CLA CLA1**(**PP\_shift**[**0**][**15**:**0**],** PP\_shift**[**1**][**15**:**0**],** 0**,** cout**[**0**],** sum**[**0**][**15**:**0**]);**

CLA CLA2**(**PP\_shift**[**2**][**15**:**0**],** PP\_shift**[**3**][**15**:**0**],** 0**,** cout**[**1**],** sum**[**1**][**15**:**0**]);**

CLA CLA3**(**PP\_shift**[**4**][**15**:**0**],** PP\_shift**[**5**][**15**:**0**],** 0**,** cout**[**2**],** sum**[**2**][**15**:**0**]);**

CLA CLA4**(**PP\_shift**[**6**][**15**:**0**],** PP\_shift**[**7**][**15**:**0**],** 0**,** cout**[**3**],** sum**[**3**][**15**:**0**]);**

CLA CLA5**(**sum**[**0**][**15**:**0**],** sum**[**1**][**15**:**0**],** 0**,** cout**[**4**],** sum**[**4**][**15**:**0**]);**

CLA CLA6**(**sum**[**2**][**15**:**0**],** sum**[**3**][**15**:**0**],** 0**,** cout**[**5**],** sum**[**5**][**15**:**0**]);**

CLA CLA7**(**sum**[**4**][**15**:**0**],** sum**[**5**][**15**:**0**],** 0**,** cout**[**6**],** sum**[**6**][**15**:**0**]);**

CLA CLA8**(**sum**[**6**][**15**:**0**],** 16'b1000000100000000**,** 0**,** cout**[**7**],** sum**[**7**][**15**:**0**]);**

**assign** result**[**15**:**0**]** **=** sum**[**7**][**15**:**0**];**

**endmodule**

* Test Bench Code

**module** MULTS\_signed\_tb**();**

**reg** **signed** **[**7**:**0**]** A**;**

**reg** **signed** **[**7**:**0**]** X**;**

**wire** **signed** **[**15**:**0**]** result**;**

MULTS\_signed DUT

**(**

**.**A**(**A**),**

**.**X**(**X**),**

**.**result**(**result**)**

**);**

**initial**

**begin**

A**=**0**;** X**=-**1**;**

**#**5**;**

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=-**13**;** X**=**6**;**

**#**50

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=-**40**;** X**=-**9**;**

**#**5

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**12**;** X**=-**12**;**

**#**5

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**47**;** X**=**68**;**

**#**50

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=-**31**;** X**=-**47**;**

**#**5

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=**342**;** X**=-**19**;**

**#**5

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=-**48**;** X**=**18**;**

**#**5

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=-**36**;** X**=-**11**;**

**#**5

$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

A**=-**150**;** X**=-**15**;**

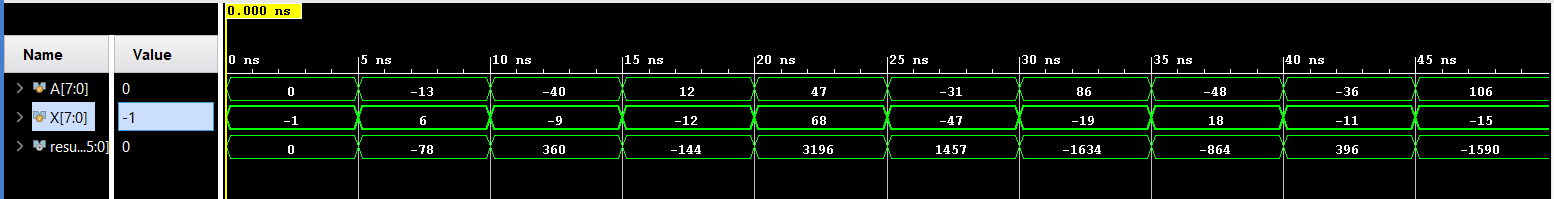
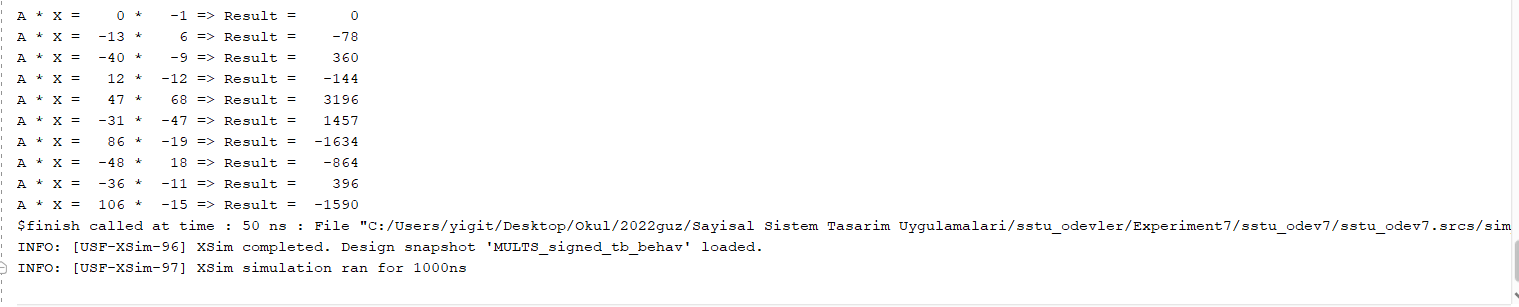
**#**5

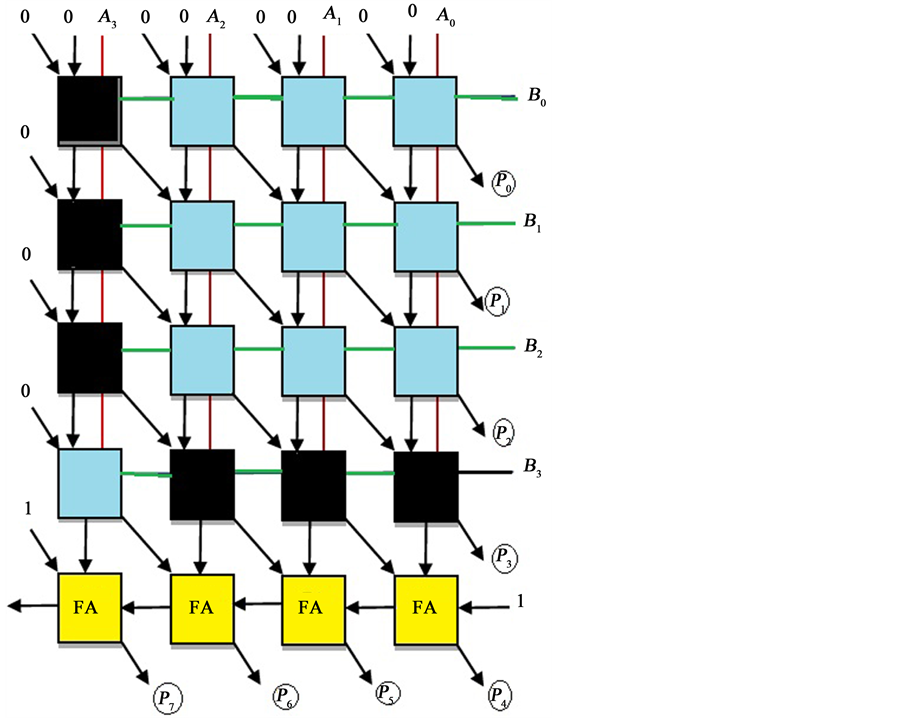
$write**(**"A \* X = %d \* %d => Result = %d\n"**,**A**,** X**,** result**);**

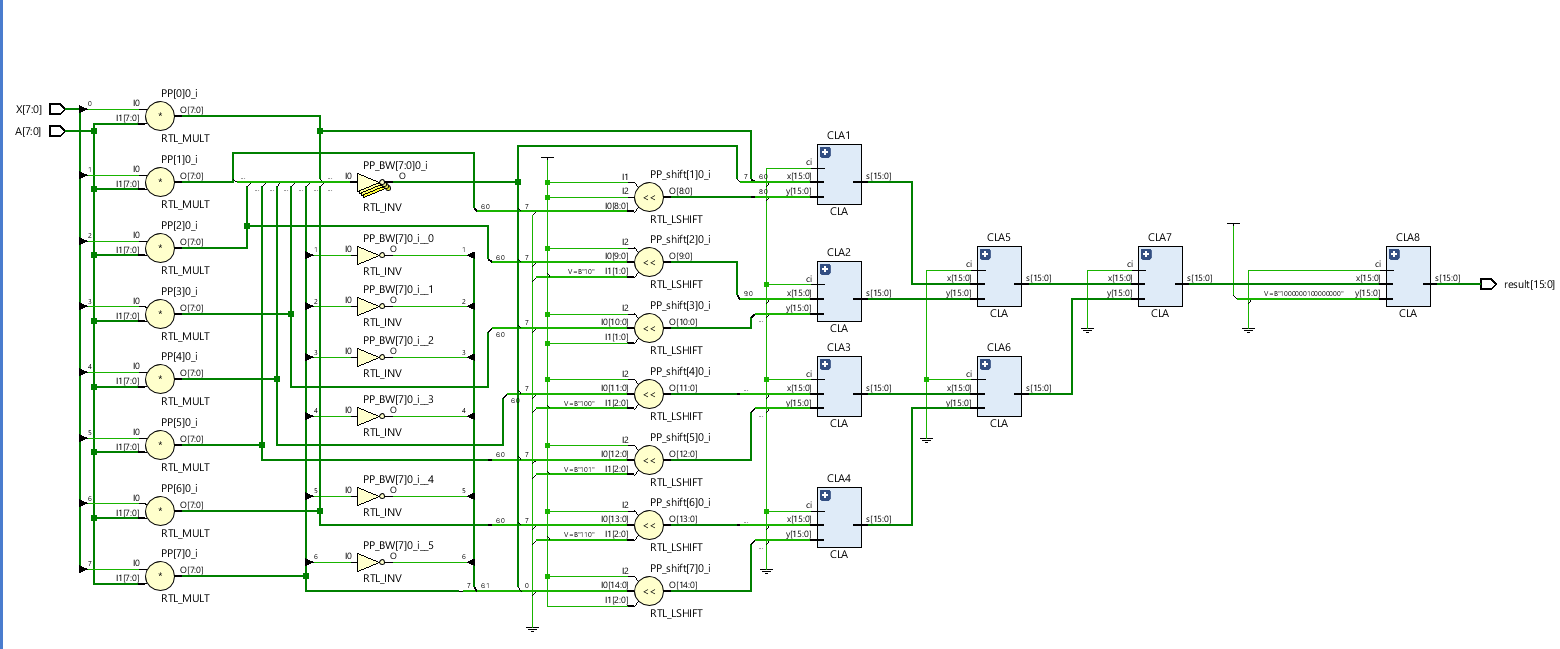
$finish**;**

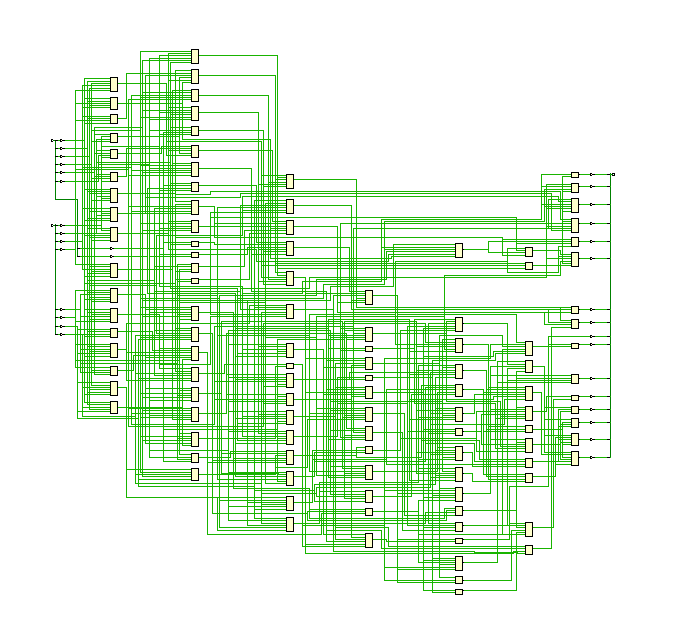
**end**

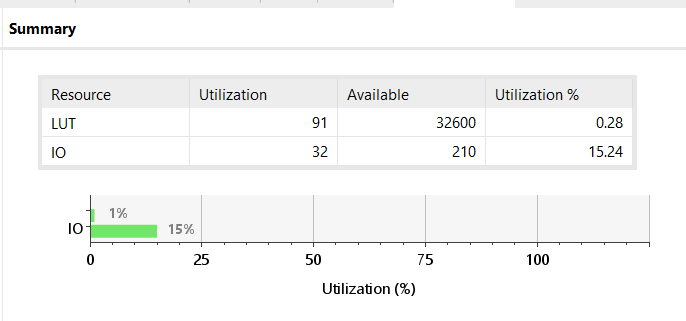
**endmodule**

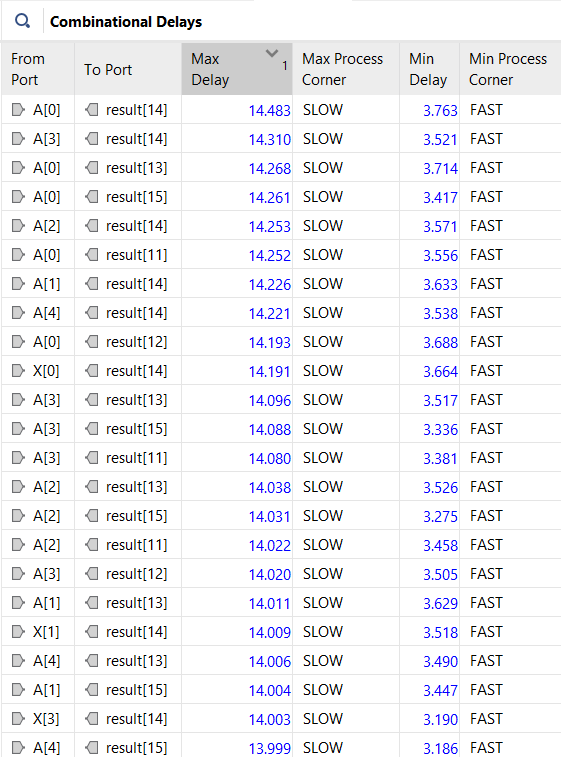
* Behavioral Simulation Result
* TCL Console Output
* Number of Stages
* As a result of two 8-bit operations, a 16-bit number and 8 partial multiplication stages are obtained in partial multiplication in the last stage. As a result of these data, 4 collection sections and 8 CLAs are needed.
* Baugh – Wooley Method
* The Baugh-Wooley method is an algorithm used to multiply 2 numbers, these numbers can be negative or positive. Multiplication is performed by taking not the most valuable bit of the partial products other than the last partial product. In the last partial multiplication, all other bits except the most significant bit are taken. In the photo below, this process is explained with an example. Multiplication is done in each of the boxes. The blue parts are the part where the products are the same, the black parts are the parts where the products become its compliment. Then these found numbers are summed as follows and written in order from the most significant bit to the least significant bit.



* Technology Schematic
* Technology Schematic



*  Utilization Report
* Timing Summary



* Combinational delays are indicated in the table above. The maximum delay of the designed circuit was 14.48 ns.

1. **BEHAVIORAL MULTIPLIER**

* Verilog Code

**module** MULTB

**(**

**input** **signed** **[**7**:**0**]** A**,**

**input** **signed** **[**7**:**0**]** B**,**

**output** **reg** **signed** **[**15**:**0**]** result

**);**

**always** **@** **\***

**begin**

result **<=** A **\*** B**;**

**end**

**endmodule**

* Test Bench Code

**module** MULTB\_tb**();**

**reg** **signed** **[**7**:**0**]** A**;**

**reg** **signed** **[**7**:**0**]** B**;**

**wire** **signed** **[**15**:**0**]** result**;**

MULTB DUT

**(**

**.**A**(**A**),**

**.**B**(**B**),**

**.**result**(**result**)**

**);**

**initial**

**begin**

A**=**45**;** B**=**87**;**

**#**5**;**

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=**19**;** B**=**78**;**

**#**5

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=-**40**;** B**=**17**;**

**#**5

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=**101**;** B**=**101**;**

**#**5

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=**34**;** B**=**103**;**

**#**5

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=-**13**;** B**=-**42**;**

**#**5

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=**12**;** B**=-**155**;**

**#**5

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=-**43**;** B**=-**19**;**

**#**5

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=-**61**;** B**=**43**;**

**#**5

$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

A**=-**18**;** B**=-**18**;**

**#**5

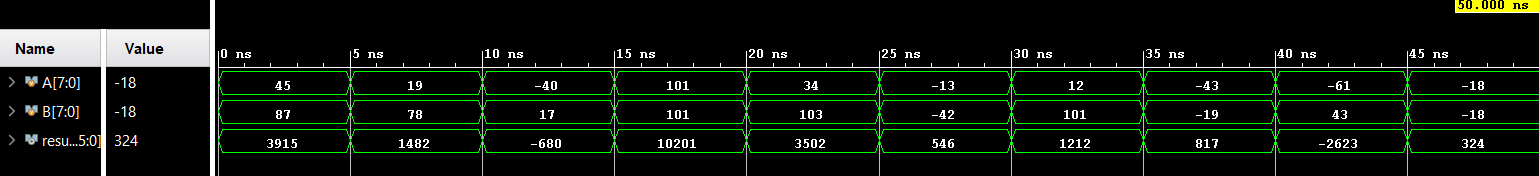
$write**(**"A \* B = %d \* %d => Result = %d\n"**,**A**,** B**,** result**);**

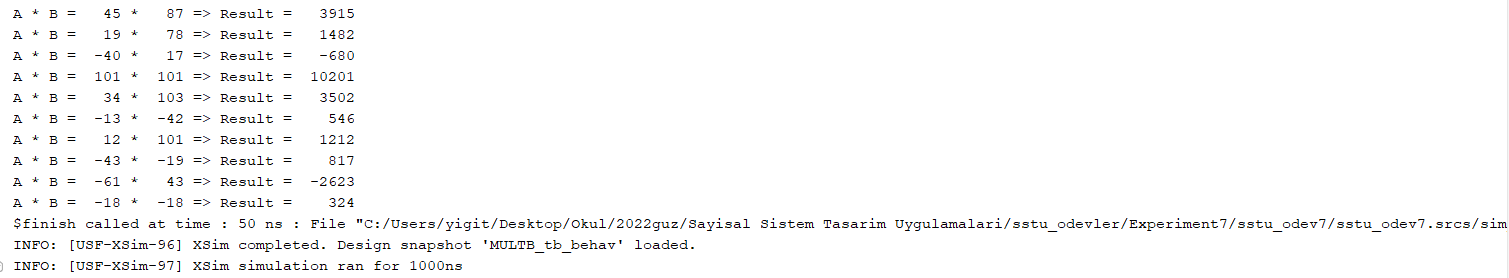
$finish**;**

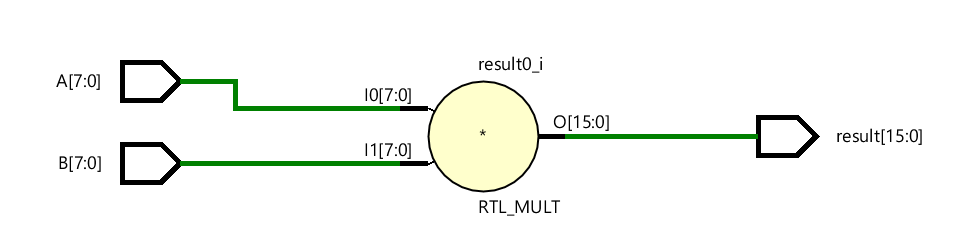
**end**

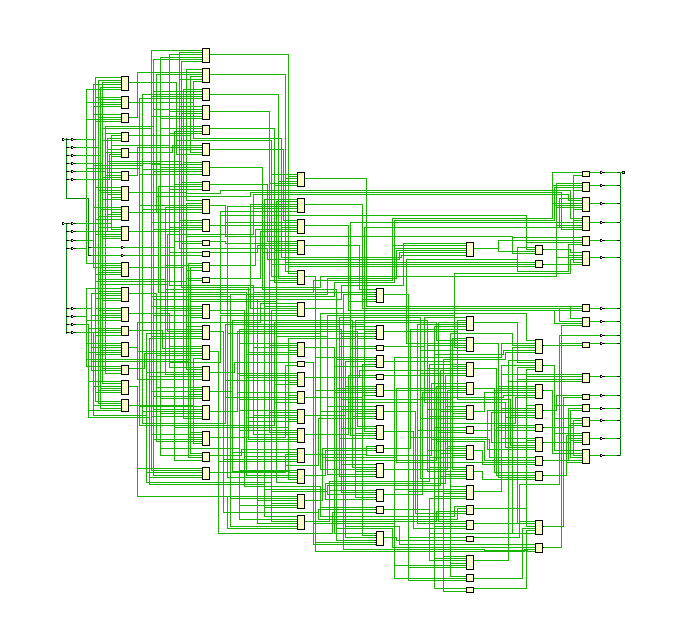
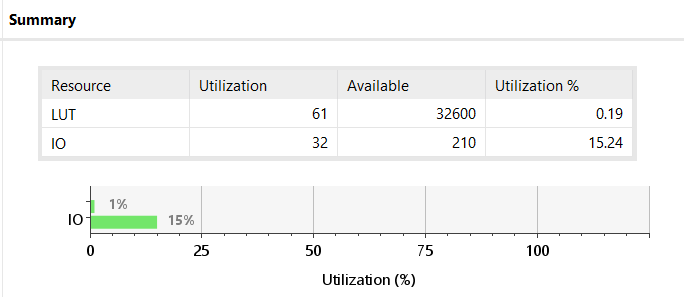
**endmodule**

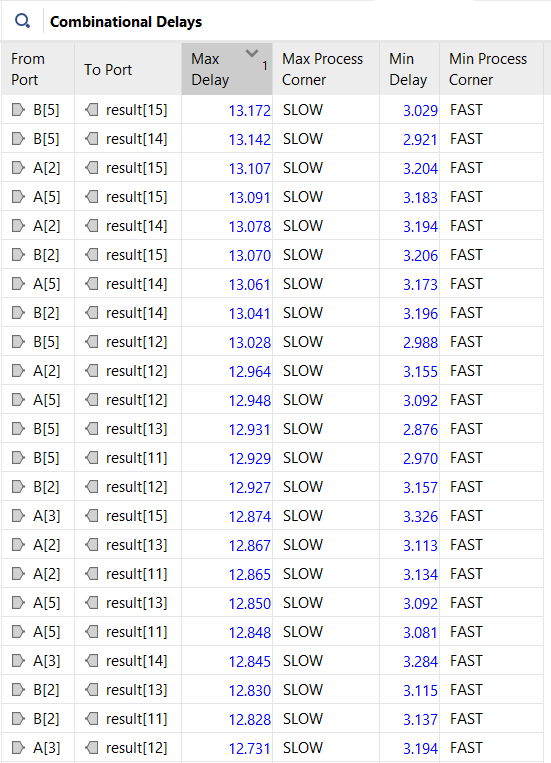
* Behavioral Simulation



* TCL Console
* RTL Schematic



* Technology Schematic
* Utilization Report
* Timing Summary



* Combinational delays are indicated in the table above. The maximum delay of the designed circuit was 13.17ns.
* Comparing structural and behavioral designs, behavioral design appears to be more efficient in terms of delay and space. 91 LUTs are used in the structural structure and 61 LUTs are used in behavioral design. When the delays are compared, it seems to have a maximum delay of 13.17 ns in the structural 14.48 ns behavioral design.

1. **MULTIPLY AND ACCUMULATE (MAC)**

* Verilog Code

**module** MAC

**(**

**input** clk**,**

**input** reset**,**

**input** **signed** **[**23**:**0**]** data**,**

**input** **signed** **[**23**:**0**]** weight**,**

**output** **reg** **signed** **[**15**:**0**]** result

**);**

**wire** **signed** **[**15**:**0**]** product **[**2**:**0**];**

**wire** **signed** **[**15**:**0**]** sum **[**1**:**0**];**

**reg** **[**1**:**0**]** count**;**

MULTB MULT0**(**data**[**7**:**0**],** weight**[**7**:**0**],** product**[**0**][**15**:**0**]);**

MULTB MULT1**(**data**[**15**:**8**],** weight**[**15**:**8**],** product**[**1**][**15**:**0**]);**

MULTB MULT2**(**data**[**23**:**16**],** weight**[**23**:**16**],** product**[**2**][**15**:**0**]);**

Behav\_Adder ADD1**(**product**[**0**][**15**:**0**],** product**[**1**][**15**:**0**],** sum**[**0**][**15**:**0**]);**

Behav\_Adder ADD2**(**product**[**2**][**15**:**0**],** sum**[**0**][**15**:**0**],** sum**[**1**][**15**:**0**]);**

**always** **@** **(posedge** clk **or** **posedge** reset**)**

**begin**

**if(**reset**)**

**begin**

count **<=** 0**;**

result **<=** 0**;**

**end**

**else**

**begin**

result **<=** result **+** sum**[**1**][**15**:**0**];**

count **<=** count **+** 1**;**

**end**

**end**

**endmodule**

* Testbench Code

**module** MAC\_tb**();**

**reg** clk**,** reset**;**

**reg** **signed** **[**23**:**0**]** data**;**

**reg** **signed** **[**23**:**0**]** weight**;**

**wire** **signed** **[**15**:**0**]** result**;**

MAC UUT

**(**

**.**clk**(**clk**),**

**.**reset**(**reset**),**

**.**data**(**data**),**

**.**weight**(**weight**),**

**.**result**(**result**)**

**);**

**initial**

**begin**

clk **=** 0**;**

reset **=** 1**;**

data **=** 24'b00000000\_0000100\_00000000**;** // 0 4 0

weight **=** 24'b11111111\_11111111\_11111111**;**

**#**12**;**

reset **=** 0**;**

**#**9**;**

data **=** 24'b00000001\_00001000\_00000000**;** // 1 8 0

weight **=** 24'b11111111\_00001000\_11111111**;**

**#**14**;**

data **=** 24'b00000000\_00000101\_00000011**;** // 0 6 3

weight **=** 24'b11111111\_11111111\_11111111**;**

**#**5**;**

$write**(**"Dataset = [0 4 0; 1 8 0; 0 6 3], Weights = [-1 -1 -1; -1 8 -1; -1 -1 -1]\n"**);**

$write**(**"R11 = D11 \* W11, R12 = D12 \* W12, R13 = D13 \* W13\n"**);**

$write**(**"R21 = D21 \* W21, R22 = D22 \* W22, R23 = D23 \* W23\n"**);**

$write**(**"R31 = D31 \* W31, R32 = D32 \* W32, R33 = D33 \* W33\n"**);**

$write**(**"Result Matrix = [0 -4 0; -1 64 0; 0 -6 -3]\n"**);**

$write**(**"Result = R11 + R12 + R13 + R21 + R22 + R23 + R31 + R32 + R33\n"**);**

$write**(**"Result = %d\n"**,** result**);**

$finish**;**

**end**

**always**

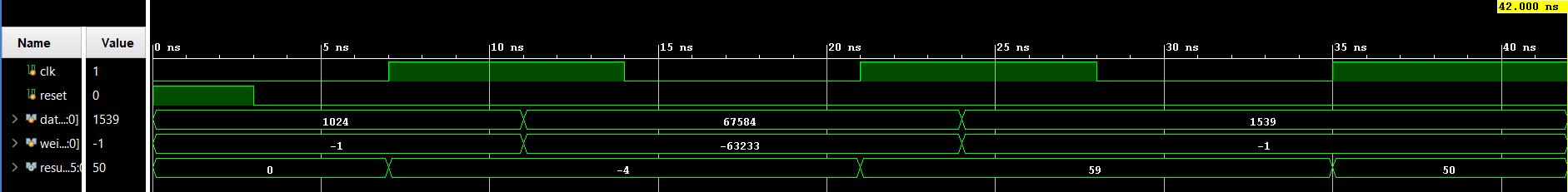
**begin**

**#**10**;** clk **=** **~**clk**;**

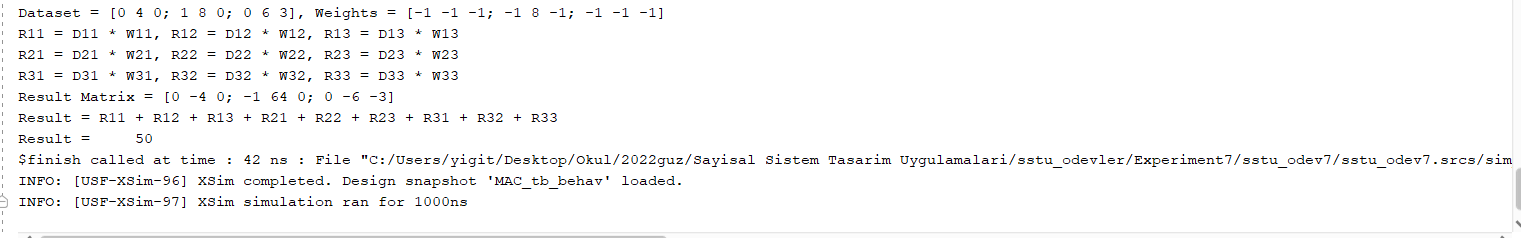
**end**

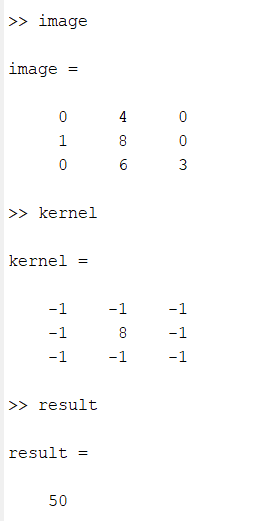
**endmodule**

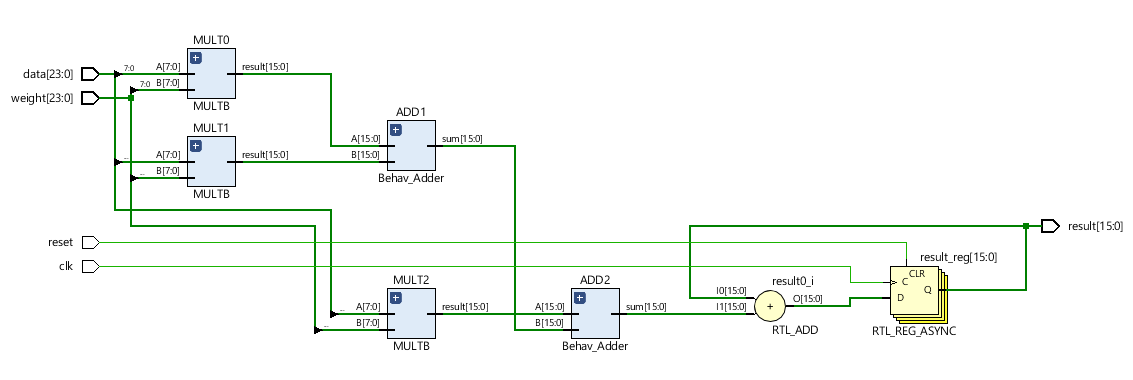
* Behavioral Simulation
* According to the code written, when the posedge signal of the clock comes, the product of each line is calculated. Since we have a 3\*3 matrix, the matrix multiplication will be calculated in the case where reset=0 and posedge comes 3 times in a row. The following is the description of the simulation.

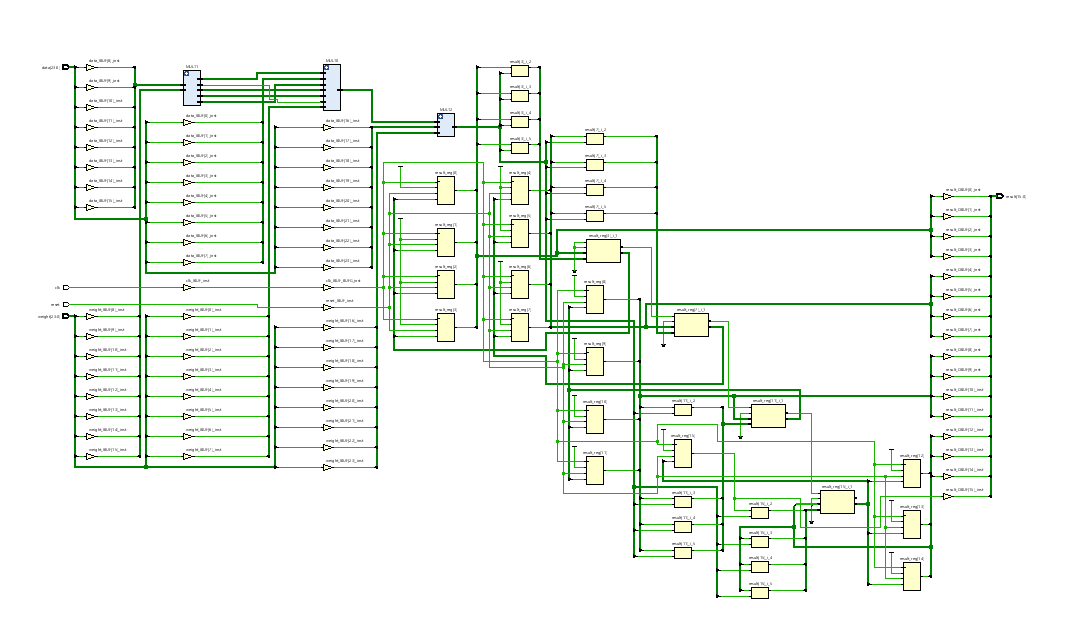


* TCL Console and MATLAB console output for verification

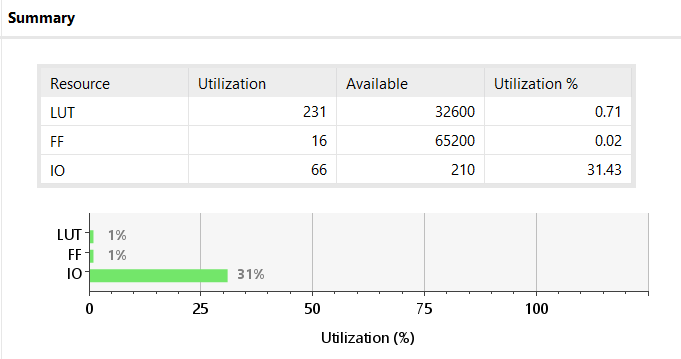


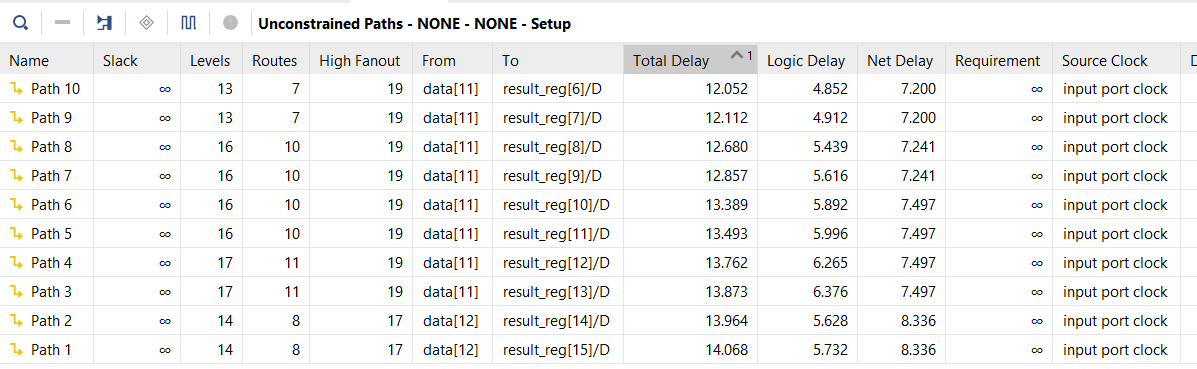


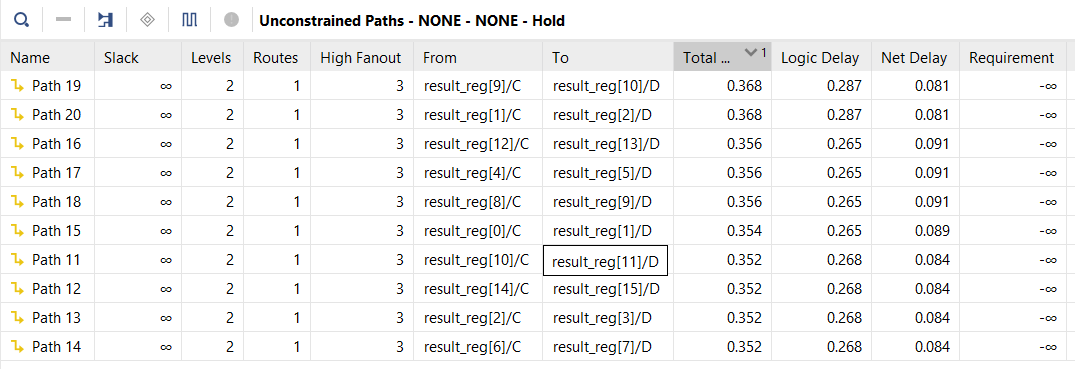
* RTL Schematic
* Technology Schematic



* Utilization Report



* Timing Summary
* Setup Delays
* Hold Delays



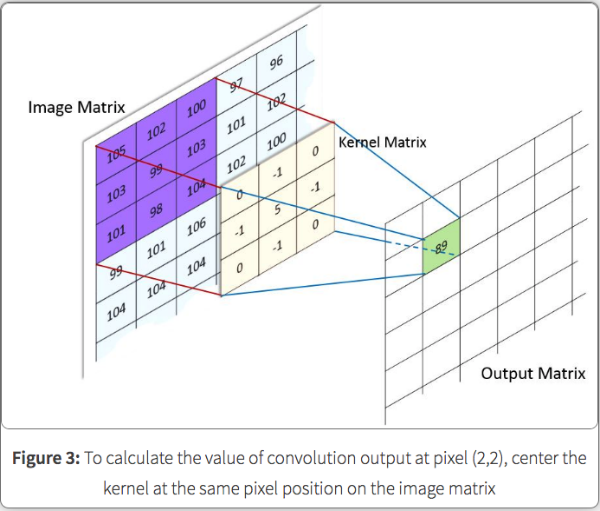
* Maximum delay is 14.06ns. If the maximum clock frequency is calculated from f=1/T, the approximation is 1/14.06 = 73MHz. While doing simulation, we should pay attention to this value and determine the frequency accordingly. This value shows the maximum frequency value it can receive.

1. **2-D CONVOLUTION**

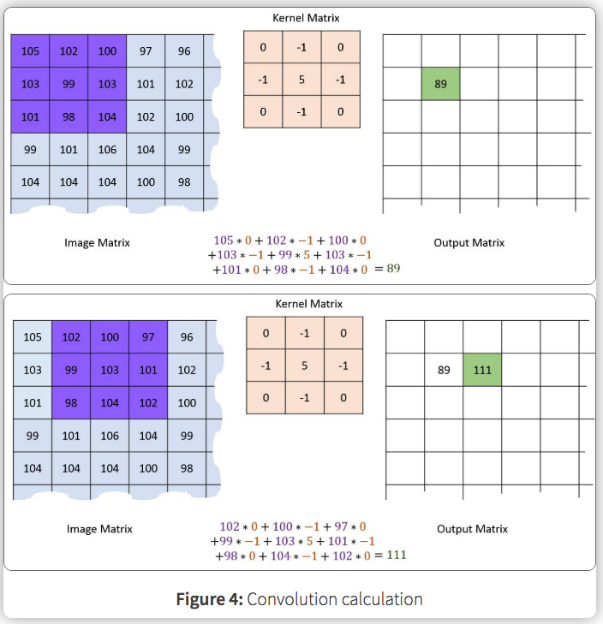
Generally, 1D convolution is used in speech processing, 2D convolution is in image processing, and 3D convolution is commonly used in video processing. 2D convolution can be used to define the edges of images or to remove noise.

The 2D convolution process is done as follows:

The first element of the kernel matrix is placed in the first element of the image matrix. In other words, each element of the kernel matrix rests on an element on the image matrix. Next, each element of the kernel matrix is multiplied by its corresponding (i.e. overlapping) element in the image matrix. The values obtained as a result of the multiplications are collected and placed in the same location, which is the center of the kernel, in the image matrix in the output matrix.



In the picture above 105\*0 + 102\*(-1) + 100 \* 0 + 103 \*(-1) + 99 \* 5 + 103 \* (-1) + 101 \* 0 + 98\*(-1) + 104\*0 = The value is 89 and placed in the center. Although the operation may seem complicated, it is essentially dot multiplication in vectors. That is,



we say multiply the directly corresponding elements, add them together, assign that value to the central position. After this process is finished, the kernel matrix is shifted on the image matrix and the same process is repeated and the output matrix is filled. Note that no calculation can be made for the edges of the output matrix with this method. There are several suggested solutions to this situation. We can cancel the edges and make the output matrix smaller. Depending on the kernel size we use, we can add padding to our input matrix, in the example above, we can apply 1-pixel padding (3x3 kernel size).

* Verilog Code

**module** Conv

**(**

**input** clk**,**

**input** reset**,**

**input** **signed** **[**23**:**0**]** data**,**

**input** **signed** **[**23**:**0**]** weight**,**

**output** **signed** **[**15**:**0**]** result

**);**

MAC MAC

**(**

**.**clk**(**clk**),**

**.**reset**(**reset**),**

**.**data**(**data**),**

**.**weight**(**weight**),**

**.**result**(**result**)**

**);**

**endmodule**

* Test Bench Code

**module** Conv\_tb**();**

**reg** clk**,** reset**;**

**reg** **signed** **[**23**:**0**]** data**;**

**reg** **signed** **[**23**:**0**]** weight**;**

**wire** **signed** **[**15**:**0**]** result**;**

Conv DUT

**(**

**.**clk**(**clk**),**

**.**reset**(**reset**),**

**.**data**(**data**),**

**.**weight**(**weight**),**

**.**result**(**result**)**

**);**

**initial**

**begin**

clk **=** 0**;**

reset **=** 1**;**

data **=**24'b10000000\_10000000\_10000000**;** weight **=** 24'b11111111\_11111111\_11111111**;** **#**8**;**

reset **=** 0**;** **#**10**;**

data **=** 24'b11111111\_11111111\_10000000**;** weight **=** 24'b11111111\_00001000\_11111111**;** **#**15**;**

data **=** 24'b11111111\_11111111\_10000000**;** weight**=** 24'b11111111\_11111111\_11111111**;** **#**10**;**

$write**(**"Result(11) = %d\n"**,**result**);**

reset **=** 1**;**

data **=** 24'b10000000\_10000000\_10000000**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

reset **=** 0**;** **#**10

data **=** 24'b11111111\_10000000\_11111111**;** weight **=** 24'b11111111\_00001000\_11111111**;** **#**10

data **=** 24'b11111111\_10000000\_11111111**;**weight **=** 24'b11111111\_11111111\_11111111**;** **#**10**;**

$write**(**"Result(12) = %d\n"**,** result**);**

reset **=** 1**;**

data **=** 24'b10000000\_10000000\_10000000**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

reset **=** 0**;** **#**10

data **=**24'b10000000\_11111111\_11111111**;** weight **=**24'b11111111\_00001000\_11111111**;** **#**10**;**

data **=**24'b10000000\_11111111\_11111111**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

$write**(**"Result(13) = %d\n"**,**result**);**

reset **=** 1**;**

data **=** 24'b11111111\_11111111\_10000000**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

reset **=** 0**;** **#**10

data **=**24'b11111111\_11111111\_10000000**;** weight **=** 24'b11111111\_00001000\_11111111**;** **#**10**;**

data **=** 24'b11111111\_11111111\_10000000**;** weight **=** 24'b11111111\_11111111\_11111111**;** **#**10**;**

$write**(**"Result(21) = %d\n"**,**result**);**

reset **=** 1**;**

data **=** 24'b11111111\_10000000\_11111111**;** weight **=** 24'b11111111\_11111111\_11111111**;** **#**10**;**

reset **=** 0**;** **#**10

data **=** 24'b11111111\_10000000\_11111111**;** weight **=**24'b11111111\_00001000\_11111111**;** **#**10**;**

data **=** 24'b11111111\_10000000\_11111111**;** weight **=** 24'b11111111\_11111111\_11111111**;** **#**10**;**

$write**(**"Result(22) = %d\n"**,**result**);**

reset **=** 1**;**

data **=**24'b10000000\_11111111\_11111111**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

reset **=** 0**;** **#**10

data **=**24'b10000000\_11111111\_11111111**;** weight **=**24'b11111111\_00001000\_11111111**;** **#**10**;**

data **=**24'b10000000\_11111111\_11111111**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

$write**(**"Result(23) = %d\n"**,** result**);**

reset **=** 1**;**

data **=**24'b11111111\_11111111\_10000000**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

reset **=** 0**;** **#**10

data **=** 24'b11111111\_11111111\_10000000**;** weight **=** 24'b11111111\_00001000\_11111111**;** **#**10**;**

data **=**24'b11111111\_11111111\_10000000**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

$write**(**"Result(31) = %d\n"**,**result**);**

reset **=** 1**;**

data **=**24'b11111111\_10000000\_11111111**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

reset **=** 0**;** **#**10

data **=**24'b11111111\_10000000\_11111111**;** weight **=**24'b11111111\_00001000\_11111111**;** **#**10**;**

data **=**24'b11111111\_10000000\_11111111**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

$write**(**"Result(32) = %d\n"**,**result**);**

reset **=** 1**;**

data **=**24'b10000000\_11111111\_11111111**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**10**;**

reset **=** 0**;** **#**10

data **=**24'b10000000\_11111111\_11111111**;** weight **=**24'b11111111\_00001000\_11111111**;** **#**10**;**

data **=**24'b10000000\_11111111\_11111111**;** weight **=**24'b11111111\_11111111\_11111111**;** **#**7**;**

$write**(**"Result(33) = %d\n"**,**result**);**

$finish**;**

**end**

**always**

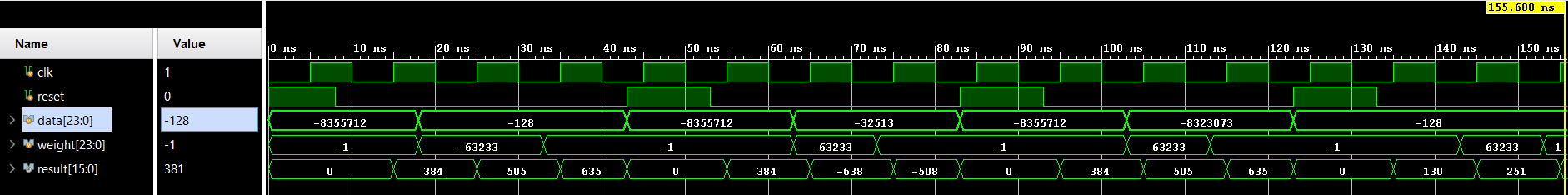
**begin**

**#**5**;** clk **=** **~**clk**;**

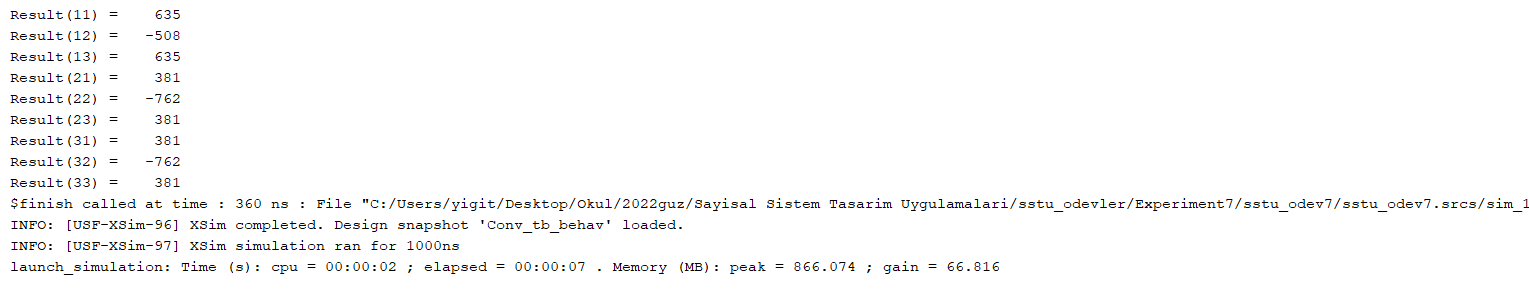
**end**

**endmodule**

* Behavioral Simulation
* In cases where reset=0 and 3 consecutive posedge signals, the simulation gives results. Demonstrate that this simulation result and codes work correctly. Information about the dataset is available in the MATLAB code.



* TCL Console



* **R:**
* MATLAB Code

% 3x3 Image Convolution

image = [128 128 128 128 128; 255 255 128 255 255 ;255 255 128 255 255 ; 255 255 128 255 255 ; 255 255 128 255 255 ];

%image = [0 4 0; 1 9 0; 7 4 5];

kernel **=** **[** **-**1 **-**1 **-**1**;** **-**1 8 **-**1**;** **-**1 **-**1 **-**1**];**

result **=** conv2**(**image**,**kernel**,** 'valid'**);**

%result = uint8(result);

subplot**(**1**,**2**,**1**);**

imagesc**(**image**);**

colormap gray

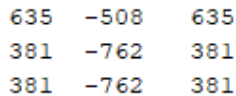
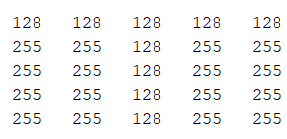
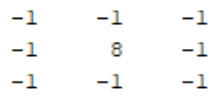
subplot**(**1**,**2**,**2**);**

imagesc**(**result**);**

colormap gray

* Output

****

****

* **REFERENCES FOR EXPLANATION OF 2D CONVOLUTION**
* ILERI, A. (2018, August 26). *2D convolution*. Medium. Retrieved December 13, 2022, from https://abdulsamet-ileri.medium.com/2d-convolution-ced5d339aa5