2023 Digital IC Design

Final Exam

Meeting Room Number：3

Please check if the last digit of your student id is equal to the meeting room number. Each meeting room has different exam questions. TAs will verify your code according to your student id, the mismatch between student id and meeting room number may cause you to lose part of the score.

Question 1: Arithmetic Expression Calculator

This assignment aims for you to create an Arithmetic Expression Calculator (AEC) with addition, subtraction, and multiplication functions. The AEC circuit must follow the rules of arithmetic expression. The input expressions consist of numeric operands, operators (+, -, *, =), and parentheses. The specification and function of the circuit is detailed in the following sections.

1. Design Specifications

1.1 Block Overview:

Figure 1. System Block Diagram

1.2 I/O Interface:

Table I. I/O interface of the design

Signal Name	I/O	width	Description
clk	I	1	This circuit is a synchronous design triggered at the positive edge of *clk*.
rst	I	1	Active-high asynchronous reset signal.
ready	I	1	Input ready indication signal. When this

			signal is high, the expression to be calculated will be inputted from the '*acsii_in*' port.
ascii_in	I	8	Each operand or operator of the expressions will be inputted with ASCII representation
valid	O	1	Output valid signal.When this signal is high, testbench will check the output *result* signal.
result	O	7	Result of arithmetic expression.

1.3 Basic Principles and Restrictions:

Three basic principles of arithmetic expressions:

1. Calculate multiplication and division before addition and subtraction. (There is no divide operation in test patterns.)
2. Perform operations inside parentheses first.
3. Calculate from left to right.

To simplify calculation, there are three restrictions in this question:

1. The input will only consist of numbers 0 to 15, four operators: +, -, *, =, and parentheses. The inputs are represented in ASCII codes, so you have to convert ASCII codes to their correct meaning in your design.
2. All input values are positive. There will be no negative numbers during the calculation process, and the maximum value will not exceed 127.Therefore, you can treat all values as unsigned numbers.
3. The input expression string will not exceed 16 characters, and it will always end with the "=" operator.

Table II. ASCII Table (Reference)

Represent	ASCII code	Represent	ASCII code	Represent	ASCII code
0	48	8	56	(	40
1	49	9	57	)	41
2	50	a(number 10)	97	*	42
3	51	b(number 11)	98	+	43
4	52	c(number 12)	99	-	45
5	53	d(number 13)	100	=	61
6	54	e(number 14)	101		
7	55	f(number 15)	102		

1.4 Function Description:

1.4.1 Infix to Postfix method:

In human calculation, we are accustomed to interpreting and calculating equations using the infix notation. However, it's difficult for computers to comprehensively process and handle the entire equation due to the limitations of the algorithm. One method to solve this problem is converting the notation to postfix and then performing the operation.

The infix to postfix operation can be performed with the following steps:

1. Create an empty stack to hold operators.
2. Scan the infix notation from left to right.
    (1) If the current token is an operand (a number), append it to the output string.
    (2) If the current token is an operator ["+", "-", "*"], the following rules should be applied:
        a. Pop the top token from the stack if its precedence is higher than or equal to the precedence of the current token. Append the token popped out to the output string.
        b. Repeat step (2.a) until the precedence of the top token of the stack is lower than the precedence of the current token or a left parenthesis is encountered.
        c. Push the current token onto the stack.
    (3) If the current token is a left parenthesis, push it onto the stack.
    (4) If the current token is a right parenthesis, pop operators from the stack and append them to the output string until a left parenthesis is found. Discard the left and right parentheses.
3. After scanning the infix string, pop out all the tokens in the stack. Append the popped-out tokens to the output string.
4. The resulting output string is in postfix notation.

1.4.2 Infix to Postfix Example:

Below is an example of how to converting an expression in infix notation "3 + 4 * (2 - 1) " to postfix notation "3 4 2 1 - * +".

Input Token	Output	Stack	Action
3	3		Append 3 to output
+		+	Push '+' onto stack
4	3 4	+	Append 4 to output
*	3 4	+ *	Push '*' onto stack

(	3 4	+ * (	Push ‘(‘ onto stack
2	3 4 2	+ * (	Append 2 to output
-	3 4 2	+ * (-	Push ‘-‘ onto stack
1	3 4 2 1	+ * (-	Append 1 to output
)	3 4 2 1 -	+ *	Pop stack until ‘(‘ is found, append popped out tokens to output
	3 4 2 1 - * +		Pop remaining tokens in stack to output

1.4.3 Calculate Expression in Postfix Notation

Here is an explanation of how to calculate the postfix expression:

1. Start by scanning the postfix expression from left to right.
   a. Push each scanned number onto the stack.
   b. When an operator is encountered, pop the top two numbers from the stack. Apply the operator to the popped-out numbers, and push the result back onto the stack.
2. Repeat steps 1 until the entire expression has been scanned.
3. The final value left on the stack is the calculation result of the expression.

Below is an example of how to calculate the postfix expression "3 4 2 1 - * +":

Input Token	Stack	Action
3	3	Push 3 onto stack
4	3 4	Push 4 onto stack
2	3 4 2	Push 2 onto stack
1	3 4 2 1	Push 1 onto stack
-	3 4 1	Pop 1 and 2 from the stack, subtract 1 from 2 to get 1. Push 1 onto the stack
*	3 4	Pop 1 and 4 from the stack, multiply them to get 4. Push 4 onto the stack.
+	7	Pop 3 and 4 from the stack, plus them to get 7. Push 7 onto the stack.

1.5 Timing Specifications:

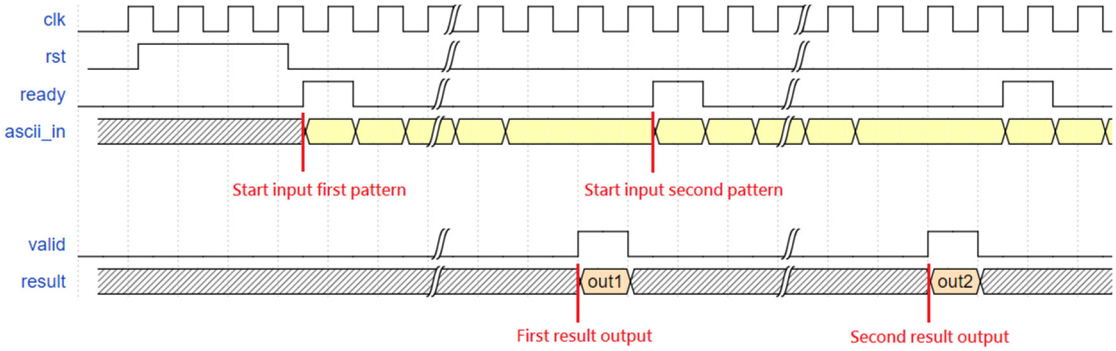


Figure 2. Timing diagram

1. When the '*ready*' signal is pulled up to high, the expression pattern will start being inputted at the same cycle. The '*ready*' signal will maintain at high for only one cycle.
2. After the testbench finishes inputting, it will begin to wait for a valid output from the AEC. When the AEC completes the calculation, the '*valid*' signal must be pulled up to high. The calculation result should be outputted from the '*result*' port in the same cycle. Then, in the next cycle, the '*valid*' signal must be pulled back as low to wait for the next pattern from the testbench.
3. When the testbench detects that the '*valid*' signal is high, it will begin to compare the '*result*' value with the golden data for correctness. The '*ready*' signal will then be pulled up as high in the next cycle.

Question 1(a): Change Precedence of Operators

The principle of arithmetic expressions is 'calculate multiplication before addition and subtraction' (the first principle in Section 1.3). In Question 1(a), the precedence of operators is changed. The '+' operator has the highest precedence, while the '-' and '*' operators have the same precedence. The second and third principles in Section 1.3 remain unchanged.

1.a.1 File Description:

File Name	Description
AEC.v	The top module of your design.
testfixture.sv	The testbench file. In this exam, you are allowed to modify terminate_cycle in testbench.
Pattern.data	Input data and golden data file.

1.a.2 Scoring of Question 1(a) [20%]

The scoring is fully based on the functional simulation results in Modelsim. There is no necessary to synthesis the Verilog codes. All of the results should be generated correctly, and you will get the following message in ModelSim simulation.

(You won't receive partial credit for partially correct answers.)

```
------------------------------------------------
---------------Start Simulation---------------
       Precendnce: '+' > '*' = '-'
------------------------------------------------
      Congratulations!!! You past all patterns!
------------------------------------------------
```

Figure 3. Simulation result for Question 1(a).

Question 1(b): Register Sharing

Register sharing is a method used to optimize the resource usage of hardware. By store the value of two data whose lifetime are not overlapped in the same register, the number of required registers in the hardware can be minimized. In Question 1(b), please reuse previously declared registers (*dataBuffer*、*OpStack*、*OutBuffer* and *arrPt*、*stackPt*、*outPt*) to replace the "*sum*" and "*sumPt*" registers in the sample code of homework 3. You are requested to complete this question by using the provided program 'AEC_reuse.v' in the folder "Q1b". You are allowed to add new states and modify the codes in the original states in the provided program, but you cannot declare any new registers.

Hint：You can reference the explanation file "hw3_explanation.pptx" to examine the lifetime of each register, and reuse the registers whose lifetime are not overlapped with "*sum*" and "*sumPt*".

1.b.1 File Description:

File Name	Description
AEC_reuse.v	The top module of your design.
testfixture.sv	The testbench file. In this exam, you are allowed to modify terminate_cycle in testbench.
Pattern.data	Input data and golden data file.

1.b.2 Scoring of Question 1(b) [20%]

The scoring is fully based on the functional simulation results in Modelsim. There is no necessary to synthesis the Verilog codes. All of the results should be generated correctly, and you will get the following message in ModelSim simulation.

(You won't receive partial credit for partially correct answers.)

```
------------------------------------------------
---------------Start Simulation---------------
------------------------------------------------
      Congratulations!!! You past all patterns!
------------------------------------------------
```

Figure 4. Simulation result for question1(b)

Question 1(c): Check Legality of Expression

In the assignment of homework 3, the input expressions consist of numeric operands, operators (+, -, *, =), and parentheses. Sometimes, there may be mismatched parentheses in an expression (i.e. numbers of left and right parentheses are mismatched, or more right parentheses are input than do left parentheses). In Question 1(c), you are requested to design an AEC circuit that can accurately detect the mismatching conditions of parentheses. The specification and function of the circuit is detailed in the following sections.

1.c.1 Block Overview:

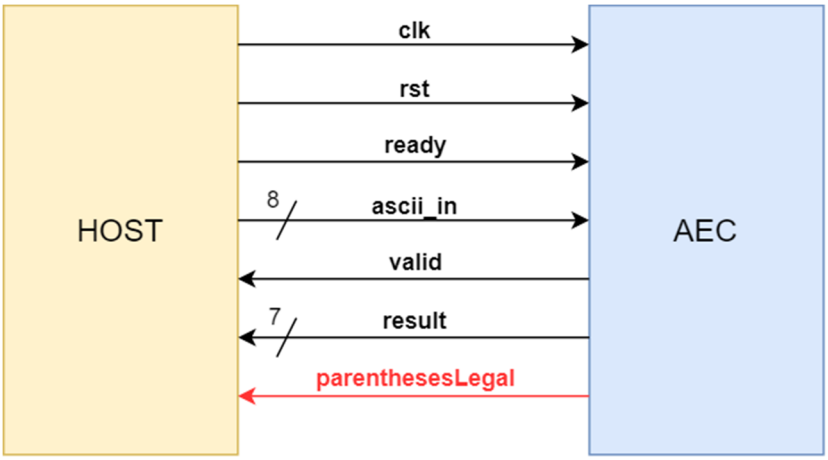


Figure 5. System Block Diagram

In Question 1(c), a new output port '*parenthesesLegal*' is added. When the parentheses in the expression is match, '*parenthesesLegal*' should be set as 1; otherwise, it should be set as 0. Testbench will verify the legality results when the '*valid*' signal is high. If the input expression is illegal, testbench will not check the calculation result. Consequently, '*result*' can be set as arbitrary value when '*parenthesesLegal*' is 0.

1.c.2 Validate Legality of Parentheses

You can use a stack to determine the legality of parentheses. When a left parenthesis is encountered, place it into the stack. When a right parenthesis is encountered, pop out the left parenthesis from the stack.

Mismatch condition 1: If right parenthesis is encountered and the stack is empty, there is no matching left parenthesis for the input right parenthesis.

Below is an example of this condition: (a*1) +) 5 (=

Input Token	Stack	Action
(	(	Push “(“ onto stack
)		Pop “(“ from stack
)		(no matching left parenthesis)

Mismatch condition 2: The expression ends, but there are still left parentheses remaining in the stack.

Below is an example of this condition: (1) + ((3*2) =

Input Token	Stack	Action
(	(	Push “(“ onto stack
)		Pop “(“ from stack
(	(	Push “(“ onto stack
(	((	Push “(“ onto stack
)	(	Pop “(“ from stack (expression ends)

1.c.3 Timing Specification:

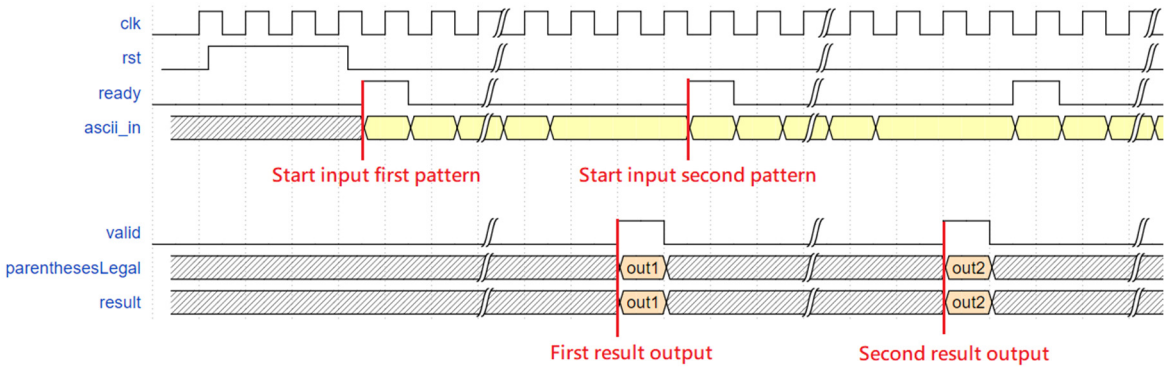


Figure 6. Timing diagram for Question 1(c)

When testbench detects that the ‘*valid*’ signal is high, it will verify the values of ‘*result*’ and ‘*parenthesesLegal*’ with the golden data for correctness. If ‘*parenthesesLegal*’ is low, testbench will not verify the value of ‘*result*’. The '*ready*' signal will then be pulled up as high in the next cycle.

1.c.4 File Description:

File Name	Description
AEC.v	The top module of your design.
testfixture.sv	The testbench file. In this exam, you are allowed to modify

	the defined num_pattern path and terminate_cycle in testbench.
Pattern1.data	Input data and golden data file.
Pattern2.data	Input data and golden data file.

1.c.5 Scoring of Question 1(c) [20%]

The scoring is fully based on the functional simulation results in Modelsim. There is no necessary to synthesis the Verilog codes. There are 2 test patterns for this question. The test pattern can be selected by modifying lines 8 and 9 in the testbench.

1.c.5.1: Pattern 1 [10%]

All of the results should be generated correctly, and you will get the following message in ModelSim simulation.

(You won't receive partial credit for partially correct answers.)

```
--------------------------------------------
--------------Start Simulation--------------
             (Pattern1 simulation)
--------------------------------------------------
      Congratulations!!! You past all patterns!
--------------------------------------------------
```

Figure 7. Simulation result for pattern1 of Question 1(c)

1.c.5.2: Pattern 2 [10%]

All of the results should be generated correctly, and you will get the following message in ModelSim simulation.

(You won't receive partial credit for partially correct answers.)

```
--------------------------------------------
--------------Start Simulation--------------
             (Pattern2 simulation)
--------------------------------------------------
      Congratulations!!! You past all patterns!
--------------------------------------------------
```

Figure 8. Simulation result for pattern2 of Question 1(c)

Question 2: Convolution

Convolution is a fundamental operation in deep learning used to extract features from input data. It involves applying a set of filters, each represented by a matrix of learnable weights, to the input data. The convolution operation is performed by sliding these filters over the input data and computing a dot product between the filter weights and the corresponding input data.

In Question 2, you are requested to design a two-layered convolutional circuit. The effect of the convolution layer is shown in Figure 9. The input image size is fixed as 64x64. Layer 0 consists of padding, convolution, and ReLU operations. While Layer 1 scales the feature map from Layer 0 to the size of the output image (32x32) with the max-pooling operation, and rounds up the results to the nearest integers. More detailed circuit functionalities will be described in the subsequent sections.

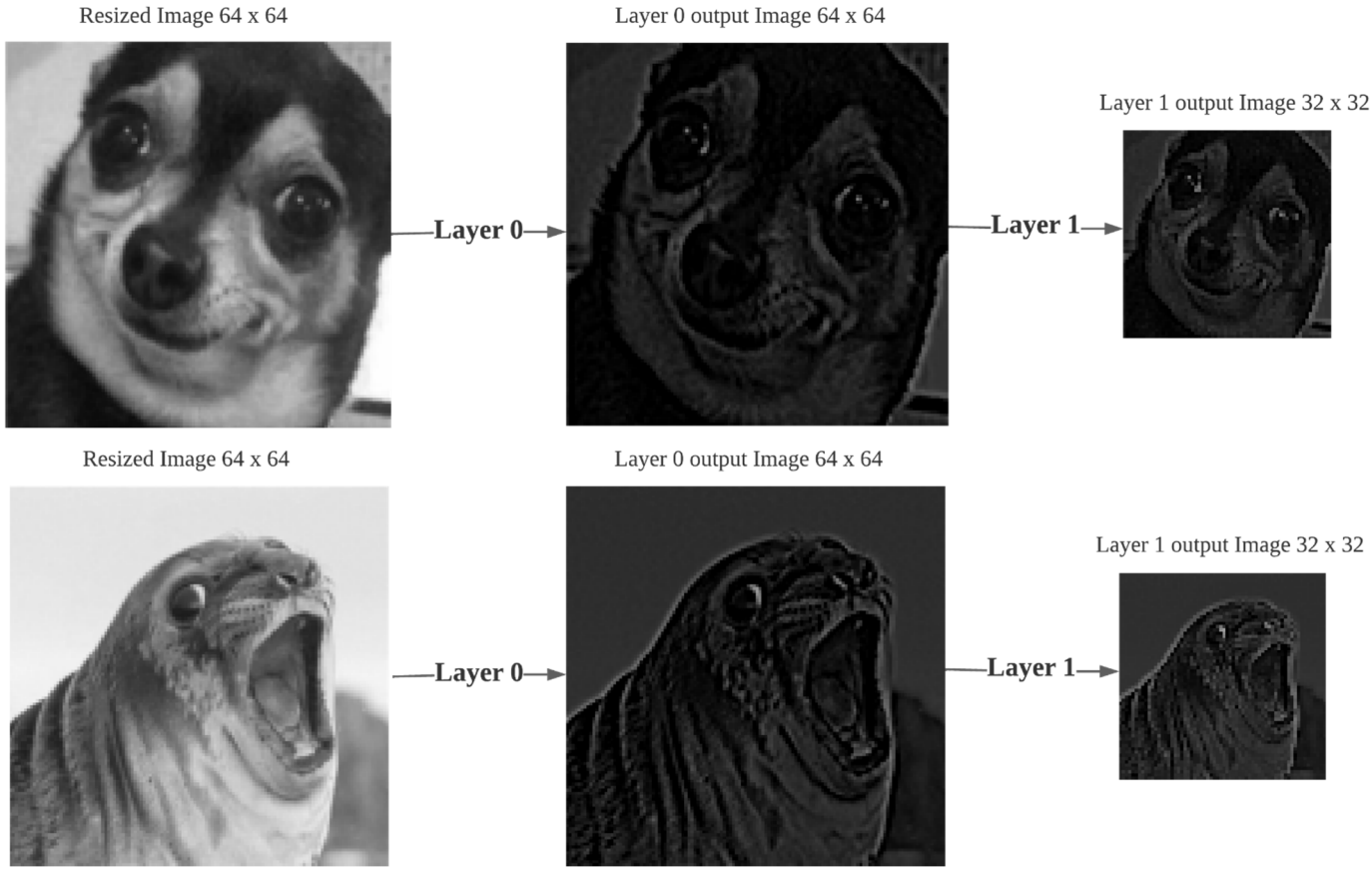


Figure 9. convolution example.

2. Design Specifications:

2.1 Block Overview:

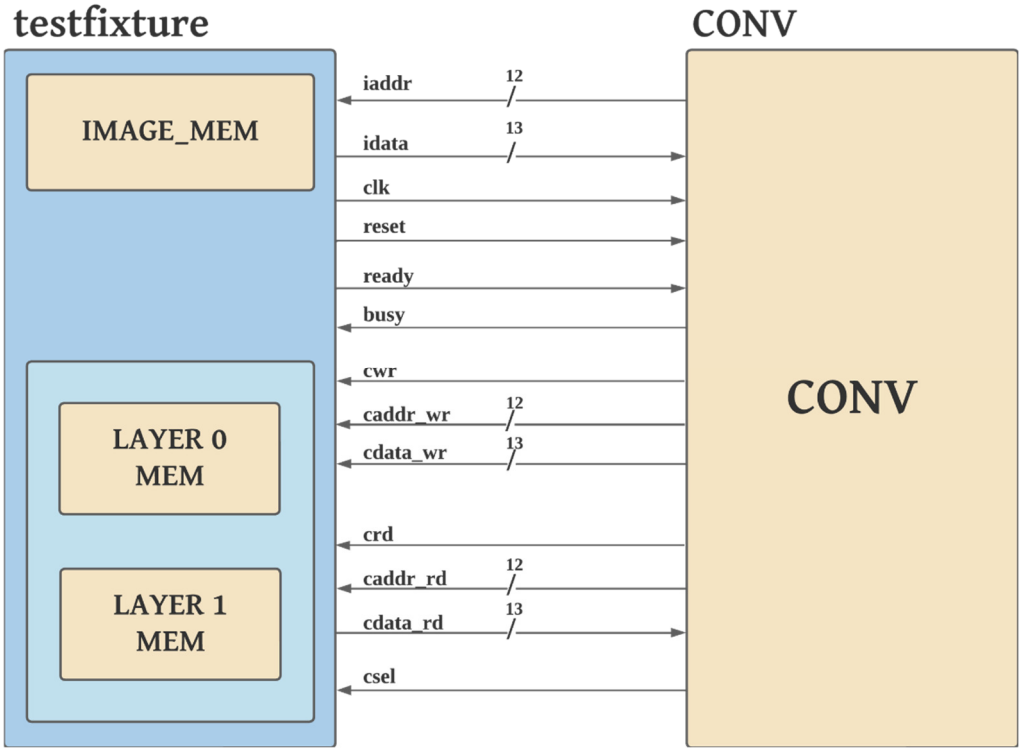


Figure 10. System Block Diagram

2.2 I/O Interface:

Table II. I/O interface of the design

Signal Name	I/O	width	Description
clk	I	1	System clock signal. This system is synchronized at the positive edge of the clock.
reset	I	1	Active-high asynchronous reset signal.
ready	I	1	Image memory ready signal. After the 64x64 grayscale image has already been loaded to the IMAGE_MEM, the *ready* signal is pulled up to high. The top module can start requiring valid *idata* by assigning corresponding *iaddr* after IMAGE_MEM is ready. The *ready* signal will be pulled down when the testbench detects the *busy* signal is high.
busy	O	1	After *ready* signal is set as high, the top module should pull the *busy* signal to high. After the top

			module finishing all the tasks, pull the *busy* signal to low, and the testfixture will check the result.
iaddr	O	12	Image memory address signal. Specifying the address of the requested grayscale image pixel.
idata	I	13	Input pixel data of the grayscale image. The pixel data consists of 9-bit integer part (MSB) and 4-bit fractional part (LSB). The testfixture will send the data whose memory address is *iaddr* with *idata* port to the top module.
csel	O	1	Memory selection signal. When *csel* signal is low, Layer 0 MEM is selected. When *csel* signal is high, Layer 1 MEM is selected. The top module can only read/write the selected memory.
crd	O	1	Read enable signal. If *crd* signal is set as high, the data at the address *caddr_rd* of the selected memory will be read and transmitted to the top module with *cdata_rd* port.
caddr_rd	O	12	Read address signal. Specifying the request data address of the selected memory.
cdata_rd	I	13	Read data signal. The data read from the selected memory would be transmitted to the top module with this port.
cwr	O	1	Write enable signal. If *cwr* signal is set as high, the value of *cdata_wr* will be written to the address *caddr_wr* of the selected memory.
caddr_wr	O	12	Write address signal. Specifying the address of the selected memory to be written.
cdata_wr	O	13	Write data signal. The data to be written to the address *caddr_wr* of the selected memory.

2.3 Memory Mapping Relations:

1. Relation between input grayscale image and IMAGE_MEM

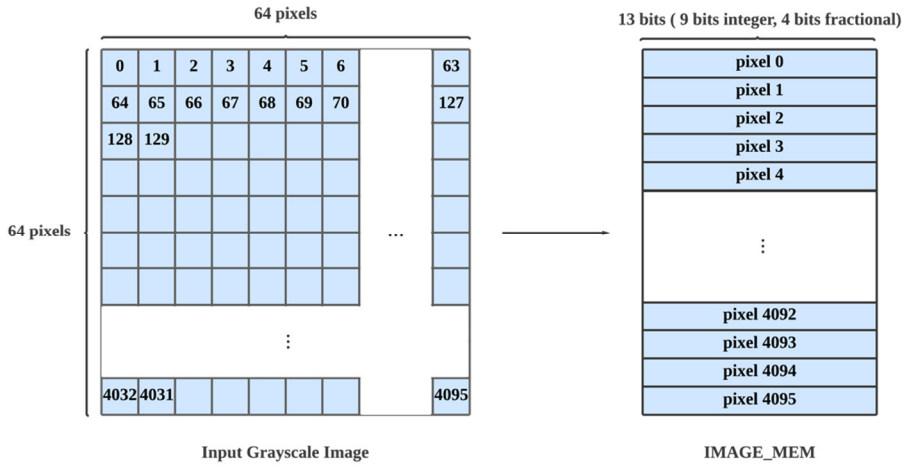


Figure 11. Input Grayscale Image memory mapping

2. Relation between Layer 0 results and Layer 0 MEM

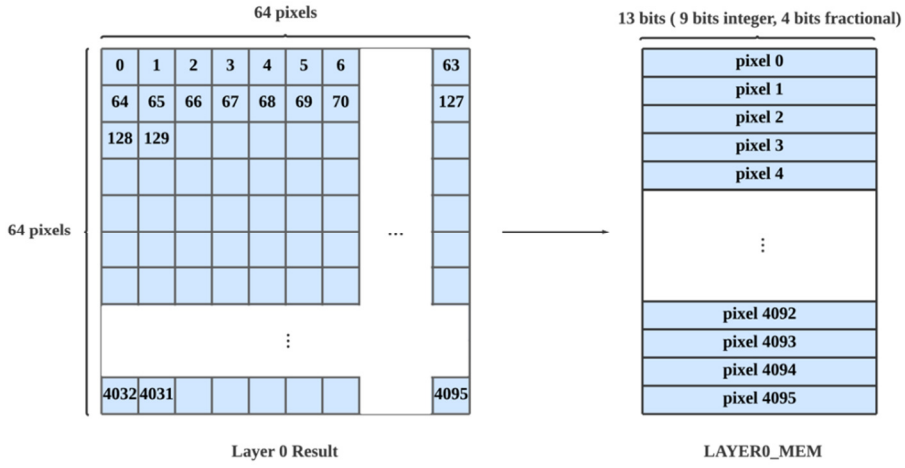


Figure 12. Layer 0 result memory mapping

3. Relation between Layer 1 results and Layer 1 MEM

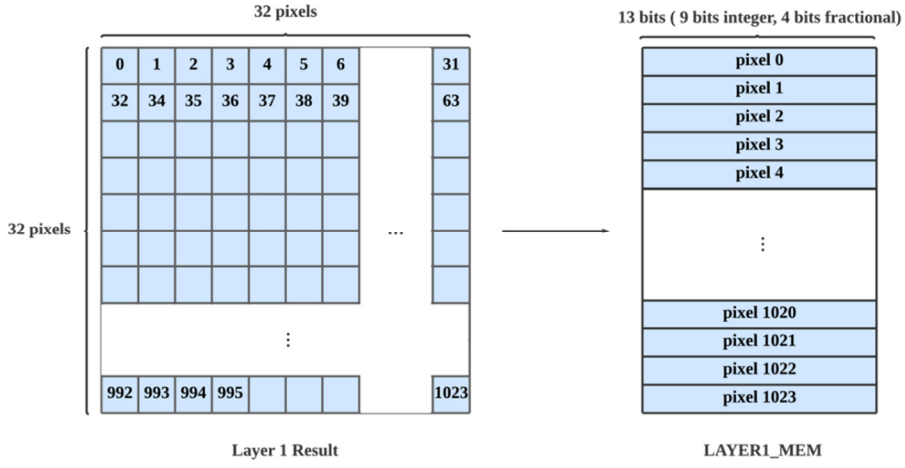


Figure 13. Layer 1 result memory mapping

2.4. Timing Specifications:

2.4.1 System control

When the input grayscale image is ready (***ready*** signal is high), the top module should pull the ***busy*** signal to high before further action. The ***ready*** signal will be pulled down after the testbench detects ***busy*** signal is high. When the top module finishes all the tasks, pull the ***busy*** signal as low to inform the testfixture to validate the results.

Figure 14. system busy control

2.4.2 IMAGE_MEM data reading control

After the input grayscale image is ready, the top module can retrieve data with ***iaddr*** and ***idata*** ports. Assign the address of required data to ***iaddr*** at the positive edge of ***clk***, and the retrieved data will be transmitted to the top module at the negative edge of ***clk***. Therefore, the top module can get the retrieved data at the next positive edge of ***clk***.

Figure 15. input image pixel retrieval control

2.4.3 Layer 0/1 MEM data reading control

When the ***crd*** signal is set to 1, the top module can read data from the memory selected with ***csel*** signal. After assign the data address with ***caddr_rd*** port at the positive edge of ***clk***, the testfixure would transmit the corresponding data to the ***cdata_rd*** port at the negative edge of clk. Therefore, the top module can get the data at the next positive edge of ***clk***.

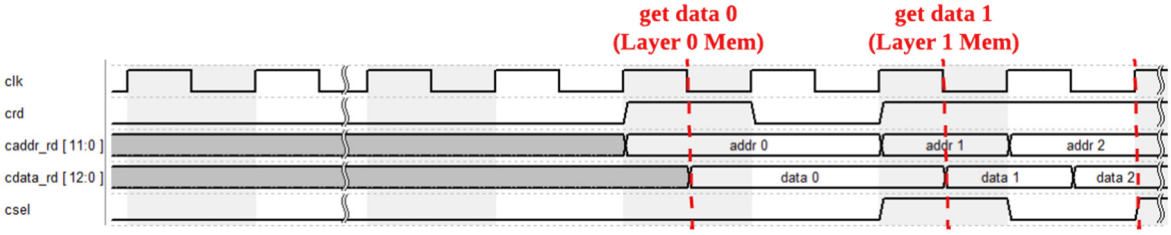


Figure 16. memory data reading control

2.4.4 Layer 0/1 MEM data writing control

When the ***cwr*** signal is set to 1, the top module can write data into the memory selected with ***csel*** signal. Assign the data address and the data to ***caddr_wr*** and ***cdata_wr*** at the positive edge of ***clk***, the data would be written to the corresponding address of the selected memory at the negative edge of ***clk***.

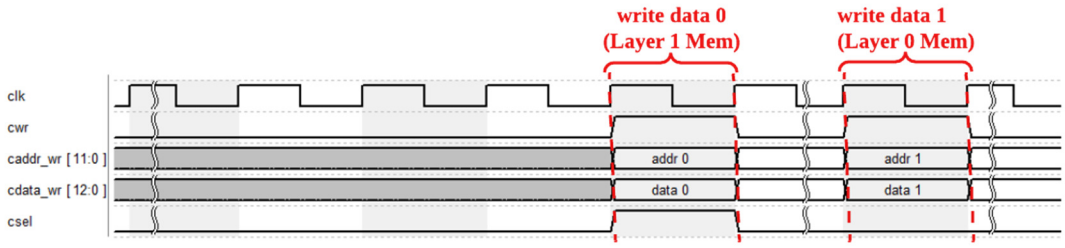


Figure 17. memory data writing control

Question 2(a): 3x3 Convolution

Input image is a 64 x 64 grayscale image stored in the IMAGE_MEM.

In Layer 0, there are three main tasks to accomplish.

1. Replicate padding the 64x64 grayscale input image to 66x66. This operation enables output feature maps after 3x3 convolution to maintain the same image size.
2. Apply 3x3 convolution to the padded grayscale image.
3. Implement ReLU function on the result of convolution.

The result of Layer 0 should be saved in Layer 0 MEM.

In Layer 1, there are two main tasks to accomplish.

1. Max-pooling the output of Layer 0 (stride=2, kernel_size=2x2)
2. Round up the result of Max-pooling to the nearest integer

The result of Layer 1 should be saved in Layer 1 MEM.

Notice：The input data and golden results of Layer 0 and Layer 1 all consist of 9-bit integer part (MSB) and 4-bit fractional part (LSB). However, you may need to use register larger than 13 bits to store the intermediate calculation results.

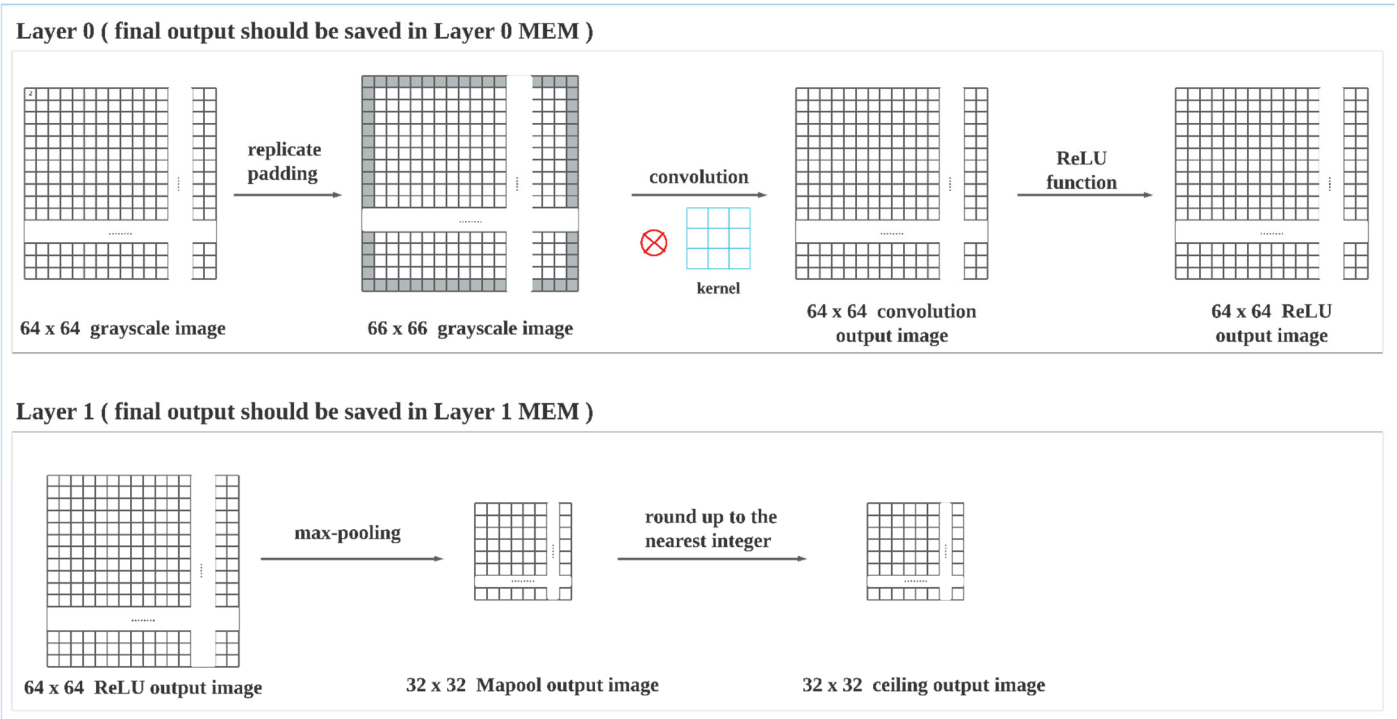


Figure 18. Overiew of 3x3 convolution circuit

2.a.1 Replicate padding operation:

To maintain the same image size (64x64) after 3x3 convolution, replicate padding the input grayscale image to 66x66 is required. Below is an example of how replicate padding works on the grayscale image. When the **padding** parameter is 1, the values at the original boundary would be replicated to form the new boundary.

In Question 2(a), the **padding** parameter is set as 1. The 64x64 image will become 66x66 after the replicate padding operation.

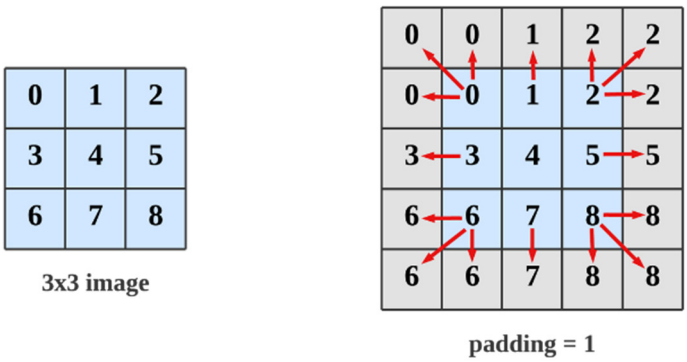


Figure 19. replicate padding example

2.a.2 3x3 Convolution:

The hyperparameter **stride** indicates the step of shifting window, the shifting window will move by *n* pixels when **stride** is *n*. For the 3x3 convolution operation in Question 2(a), the hyperparameter **stride** is set as 1. Figure 20 shows the kernel values and bias value used in Question 2(a). The kernel values and bias value consist of 9-bit integer part (MSB) and 4-bit fractional part (LSB), and the MSB represents sign bit.

bias:	13'h1FF6

13'h0004	13'h1FFF	13'h0004
13'h1FFE	13'h0008	13'h1FFE
13'h1FFF	13'h1FFF	13'h1FFF

Kernel and bias used in this exam (hexadecimal)

bias:	-0.625

0.25	-0.0625	0.25
-0.125	0.5	-0.125
-0.0625	-0.0625	-0.0625

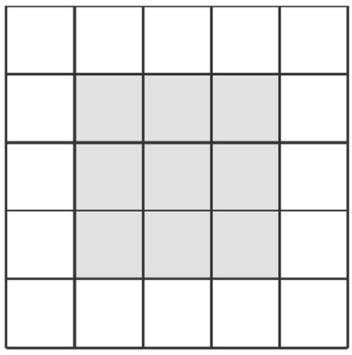
Kernel and bias used in this exam (decimal)

Figure 20. Kernel and bias used in Question 2(a)

The following example will use a 3x3 kernel as an illustration, with the hyperparameter **stride** is 1. The convolution would be implemented on the padded image of size 5x5.

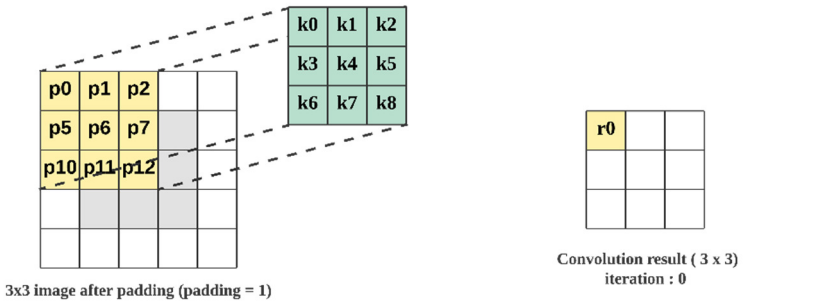
k0	k1	k2
k3	k4	k5
k6	k7	k8

3x3 kernel with dilation rate of 1

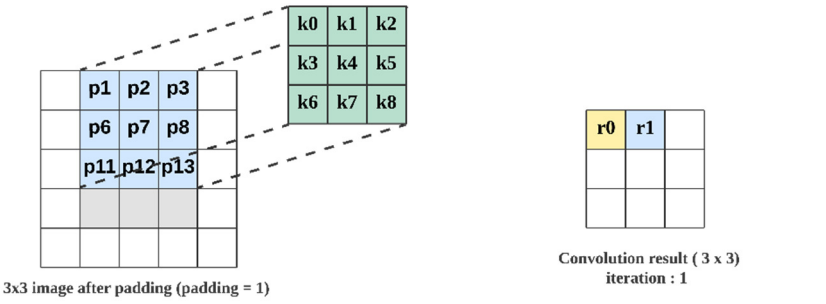


3x3 image after padding (padding = 1)

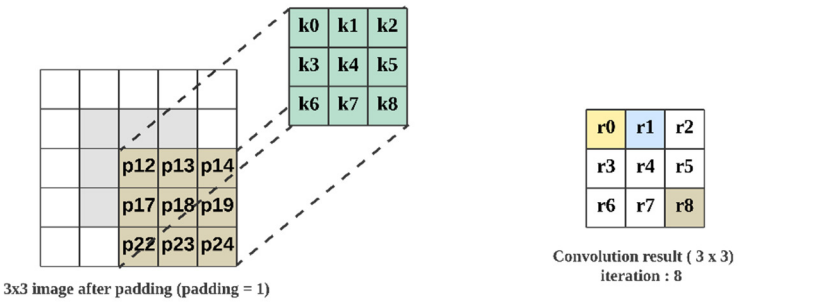
Figure 21. Kernel and image used in example



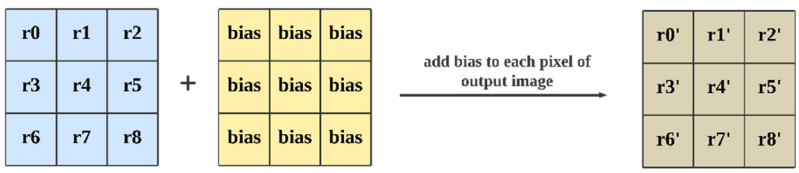
$$r0 = p0 \times k0 + p1 \times k1 + p2 \times k2 + p5 \times k3 + \ \ldots \ + p12 \times k8$$



$$r1 = p1 \times k0 + p2 \times k1 + p3 \times k2 + p6 \times k3 + \ \ldots \ + p13 \times k8$$



$$r8 = p12 \times k0 + p13 \times k1 + p14 \times k2 + p7 \times k3 + \ \ldots \ + p24 \times k8$$



convolution final step

Figure 22. convolution example on 5x5 padded image (iteration 0, 1, 8)

Figure 22 shows the convolution process, which will scan the whole image from left to right, from top to bottom with a shifting window. The pixel values in the window will convolve with the values of the kernel. From the example in Figure 22, it can be seen that the output of convolution has the same size (3x3) as the original input image.

2.a.3 ReLU function:

In the last step of Layer 0, ReLU function is applied to the output of convolution. The definition of ReLU function is shown below, which assigns 0 to the non-positive value.

$$y = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$$

2.a.4 max-pooling example:

Max-pooling is adopted in the first step of Layer 1. How max-pooling works is shown in Figure 23 below. In Question 2(a), the kernel size of max-pooling is 2x2 and the hyperparameter **stride** of max-pooling is 2. The example below adopts the same hyperparameters used in Question 2(a).

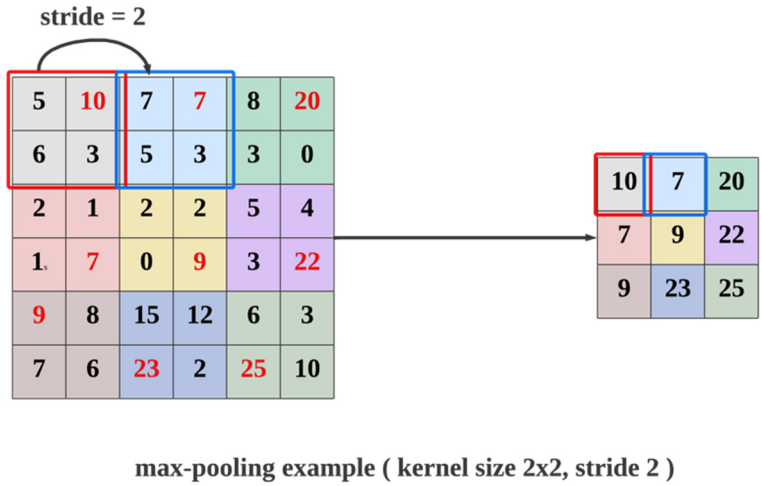


Figure 23. Max-pooling example

2.a.5 File Description:

File Name	Description
CONV3x3.v	The top module of your design.
testfixture.v	The testbench file. The content in this file is not allowed to be modified.

img.dat	Input grayscale image data.
layer0_golden.dat	Golden data of layer 0 output results.
layer1_golden.dat	Golden data of layer 1 output results.

2.a.6 Scoring of Question 2(a) [30%]

The scoring is fully based on the functional simulation results in Modelsim. There is no necessary to synthesis the Verilog codes. All of the result should be generated correctly, and you will get the following message in ModelSim simulation.

(You won't receive partial credit for partially correct answers.)

```
VSIM 2> run -all
# ------------------------------------------------------
#
# START!!! Simulation Start .....
#
# ------------------------------------------------------
#
# Layer 0 output is correct !
# Layer 1 output is correct!
#
# ------------------------------------------------------
#
# --------------------- S U M M A R Y -----------------
#
# Congratulations! Layer 0 data have been generated successfully! The result is PASS!!
#
# Congratulations! Layer 1 data have been generated successfully! The result is PASS!!
#
# terminate at      51206 cycle
# ------------------------------------------------------
#
# ** Note: $finish    : F:/master1/DicHomework/finalExam/3x3/testfixture.v(179)
#    Time: 1536180 ns  Iteration: 0  Instance: /testfixture
```

Figure 24. functional simulation result example

Question 2(b): 5x5 Convolution

Input image is a 64 x 64 grayscale image stored in the IMAGE_MEM.
In Layer 0, there are three main tasks to accomplish.

1. Zero padding the 64x64 grayscale input image to 68x68. This operation enables the output feature map of convolution to maintain the same image size.
2. Apply 5x5 convolution to the padded grayscale image
3. Implement ReLU function on the result of convolution

The result of Layer 0 should be saved in Layer 0 MEM.

In Layer 1, there are two main tasks to accomplish.

1. Max-pooling the output of Layer 0 (stride=2, kernel_size=2x2)
2. Round up the result of Max-pooling to the nearest integer

The result of Layer 1 should be saved in Layer 1 MEM.

Notice：The input data and golden results of Layer 0 and Layer 1 all consist of 9-bit integer part (MSB) and 4-bit fractional part (LSB). However, you may need to use register larger than 13 bits to store the intermediate calculation results.

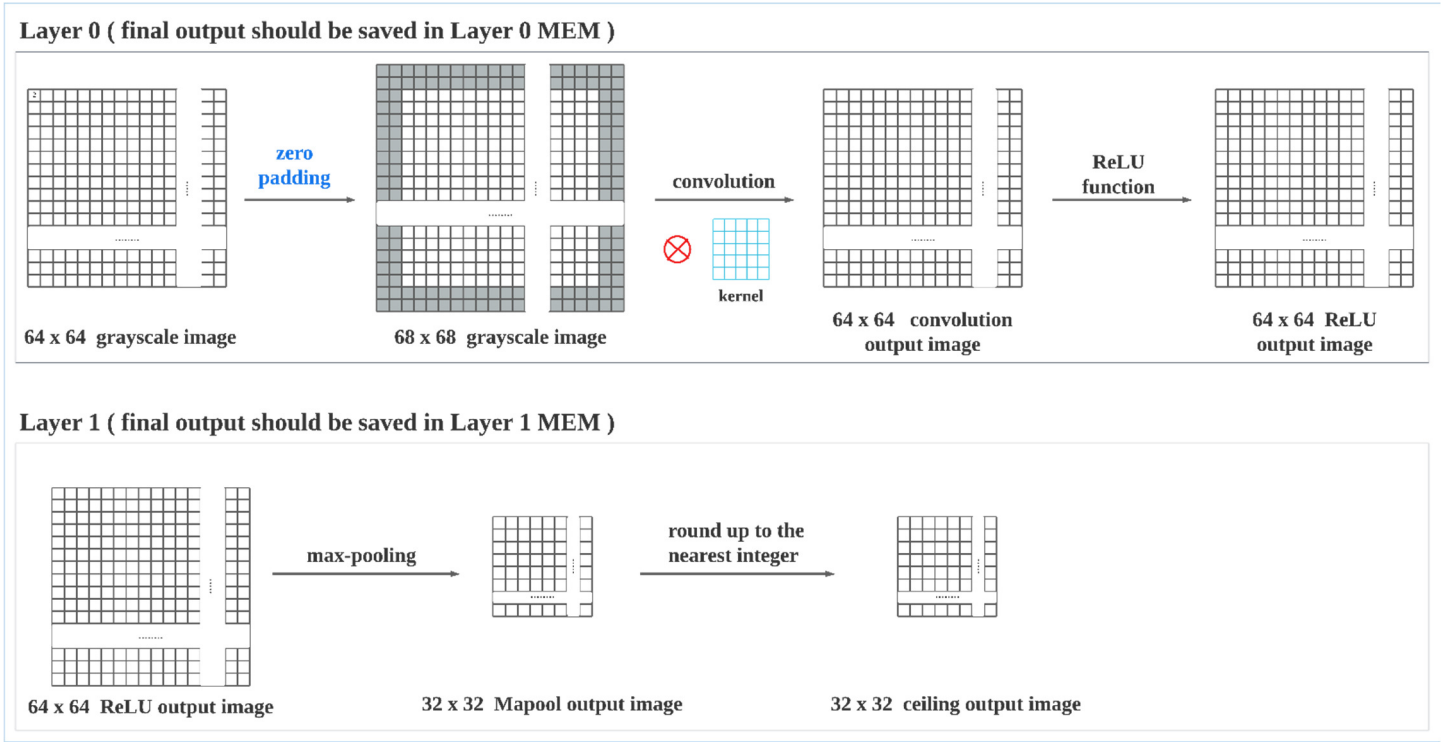


Figure 25. overiew of 5x5 convolution circuit

2.b.1 Zero padding operation:

To maintain the same image size (64x64) after convolution, zero padding the input grayscale image to 68x68 is required. Below is an example of how zero padding works on the grayscale image. When the **padding** parameter is 1, 0 would be used to form the new boundary. When the **padding** parameter is 2, the zero padding operation will be conducted two times to create new pixels at each side on the boundary. The 3x3 image in the example becomes 7x7 after the padding operation with **padding** parameter is 2.

In Question 2(b), the **padding** parameter is set as 2. The 64x64 image will become 68x68 after the replicate padding operation.

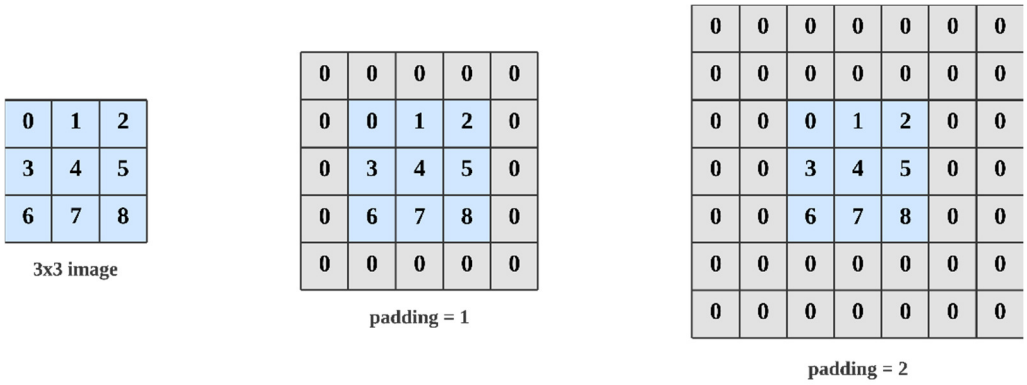


Figure 26. zero padding example

2.b.2 5x5 Convolution:

The hyperparameter **stride** indicates the step of shifting window, the shifting window will move by *n* pixels when **stride** is *n*. For the 5x5 convolution operation in Question 2(b), the hyperparameter **stride** is set as 1.

The following example will use a 5x5 kernel as an illustration, with the hyperparameter **stride** is 1. The convolution would be implemented on the padded image of size 7x7.

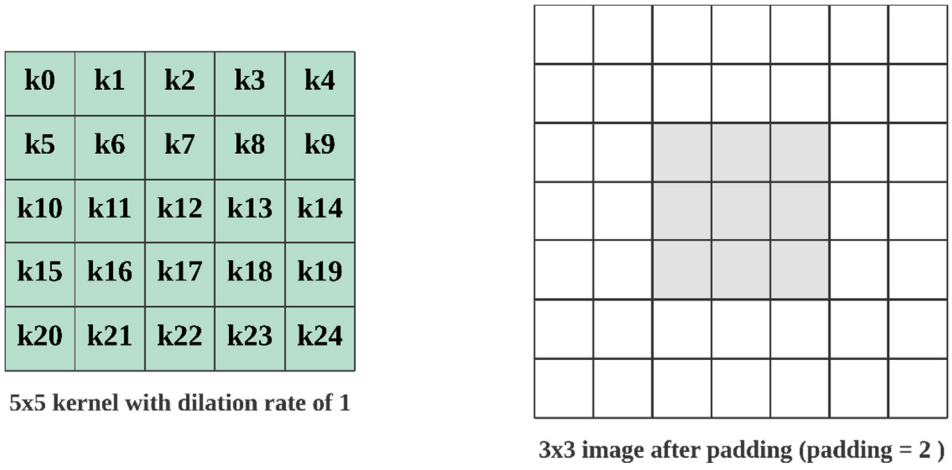
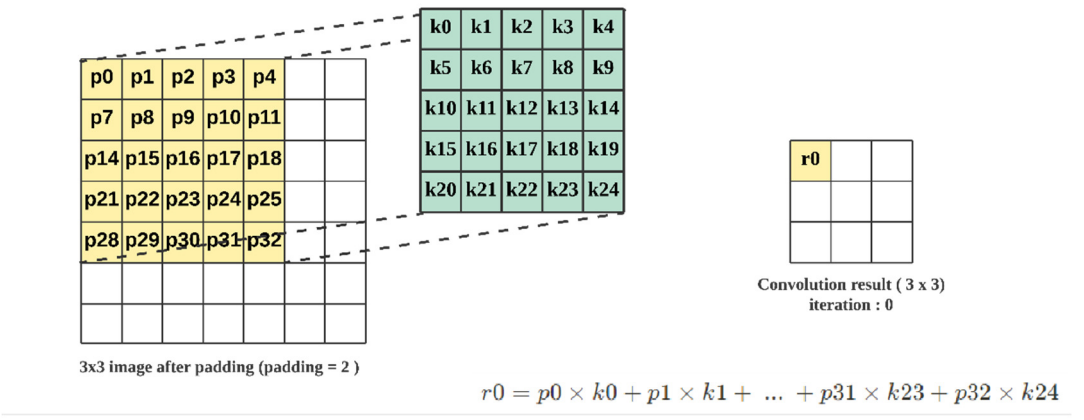
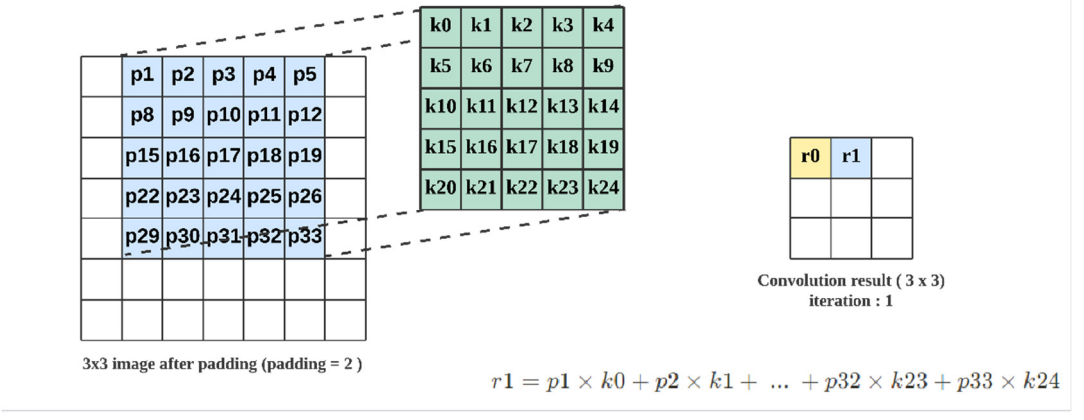


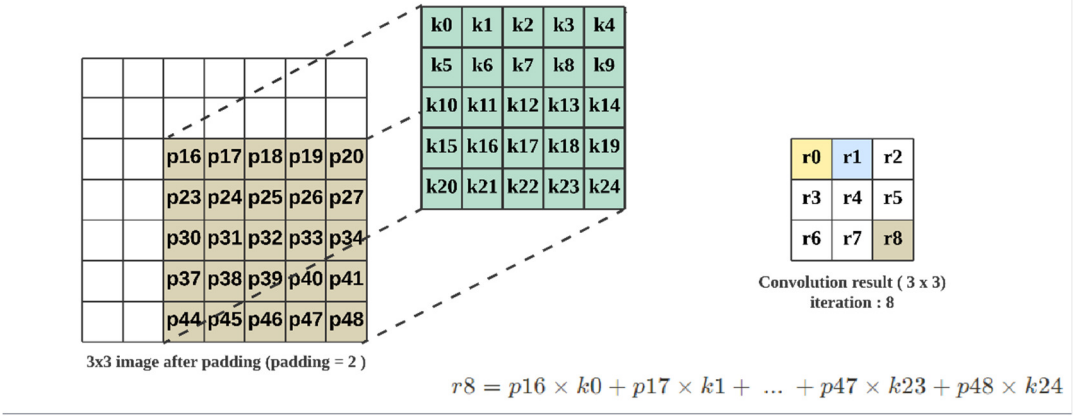
Figure 27. Kernel and image used in example



$$r0 = p0 \times k0 + p1 \times k1 + \ \ldots \ + p31 \times k23 + p32 \times k24$$



$$r1 = p1 \times k0 + p2 \times k1 + \ \ldots \ + p32 \times k23 + p33 \times k24$$



$$r8 = p16 \times k0 + p17 \times k1 + \ \ldots \ + p47 \times k23 + p48 \times k24$$

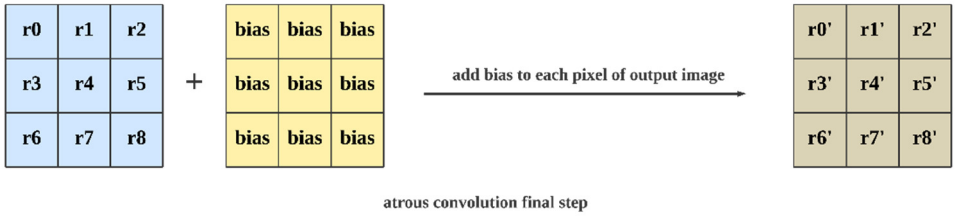


Figure 28. convolution example on 7x7 padded image (iteration 0, 1, 8)

Figure 28 shows the 7x7 convolution process, which will scan the whole image from left to right, from top to bottom with a shifting window. The pixel values in the window will convolve with the values of the kernel. From the example in Figure 28, it can be seen that the output of convolution has the same size (3x3) as the original input image, which results from the padding operation.

Figure 29 shows the kernel values and bias value used in this exam. The kernel values and bias value consist of 9-bit integer part (MSB) and 4-bit fractional part (LSB), and the MSB represents sign bit.

bias: 13'h1FF4

13'h0001	13'h1FFF	13'h0000	13'h1FFF	13'h0001
13'h1FFF	13'h0001	13'h0000	13'h0001	13'h1FFF
13'h1FFE	13'h1FFF	13'h0008	13'h1FFF	13'h1FFE
13'h1FFF	13'h0001	13'h0000	13'h0001	13'h1FFF
13'h0001	13'h1FFF	13'h0000	13'h1FFF	13'h0001

Kernel and bias used in this exam (hexadecimal)

bias: -0.75

0.0625	-0.0625	0.0	-0.0625	0.0625
-0.0625	0.0625	0.0	0.0625	-0.0625
-0.125	-0.0625	0.5	-0.0625	-0.125
-0.0625	0.0625	0.0	0.0625	-0.0625
0.0625	-0.0625	0.0	-0.0625	0.0625

Kernel and bias used in this exam (decimal)

Figure 29. Kernel and bias used in Question 2(b)

2.b.3 ReLU function:

In the last step of Layer 0, ReLU function is applied to the output of convolution. The definition of ReLU function is shown below, which assigns 0 to the non-positive value.

$$y = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$$

2.b.4 max-pooling example:

Max-pooling is adopted in the first step of Layer 1. How max-pooling works is shown in Figure 30 below. In Question 2(b), the kernel size of max-pooling is 2x2 and the hyperparameter **stride** of max-pooling is 2. The example below adopts the same hyperparameters used in this exam.

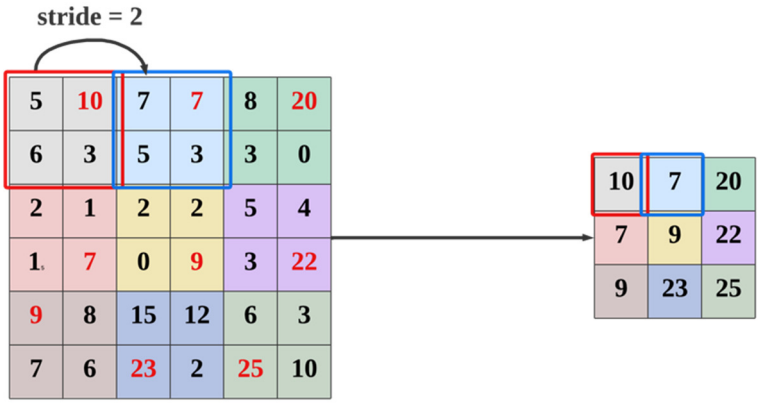


Figure 30. Max-pooling example

2.b.5 File Description

File Name	Description
CONV5x5.v	The top module of your design.
testfixture.v	The testbench file. The content in this file is not allowed to be modified.
img.dat	Input grayscale image data.
layer0_golden.dat	Golden data of layer 0 output results.
layer1_golden.dat	Golden data of layer 1 output results.

2.b.6 Scoring of Question 2(b) [10%]

The scoring is fully based on the functional simulation results in Modelsim. There is no necessary to synthesis the Verilog codes. All of the result should be generated correctly, and you will get the following message in ModelSim simulation.

(You won't receive partial credit for partially correct answers.)

```
# ----------------------------------------------------
#
# START!!! Simulation Start .....
#
# ----------------------------------------------------
#
# Layer 0 output is correct !
# Layer 1 output is correct!
#
# ----------------------------------------------------
#
# -------------------- S U M M A R Y -----------------
#
# Congratulations! Layer 0 data have been generated successfully! The result is PASS!!
#
# Congratulations! Layer 1 data have been generated successfully! The result is PASS!!
#
# terminate at     116742 cycle
# ----------------------------------------------------
#
# ** Note: $finish    : F:/master1/DicHomework/finalExam/5x5/testfixture.v(179)
#    Time: 3502260 ns  Iteration: 0  Instance: /testfixture
# 1
# Break in Module testfixture at F:/master1/DicHomework/finalExam/5x5/testfixture.v line 179
```

Figure 31. functional simulation result example

Submission:

1. Submitted files

You should classify your files into five directories and compress them to .zip format. The naming rule is Final_studentID_name.zip. If your file is not named according to the naming rule, you will lose five points. Please submit your .zip file to folder Final in moodle.

	Mid_studentID_name.zip
Q1a	
*.v	All of your Verilog RTL code for Question 1(a)
Q1b	
*.v	All of your Verilog RTL code for Question 1(b)
Q1c	
*.v	All of your Verilog RTL code for Question 1(c)
Q2a	
*.v	All of your Verilog RTL code for Question 2(a)
Q2b	
*.v	All of your Verilog RTL code for Question 2(b)

2. Notes

Deadline: 2023/5/30 12:00

No late submissions will be accepted, please pay attention to the deadline.