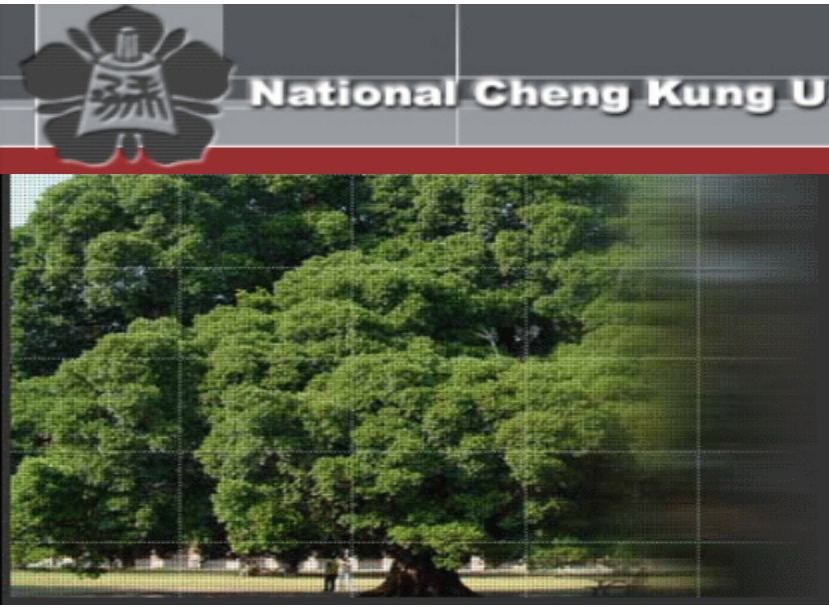
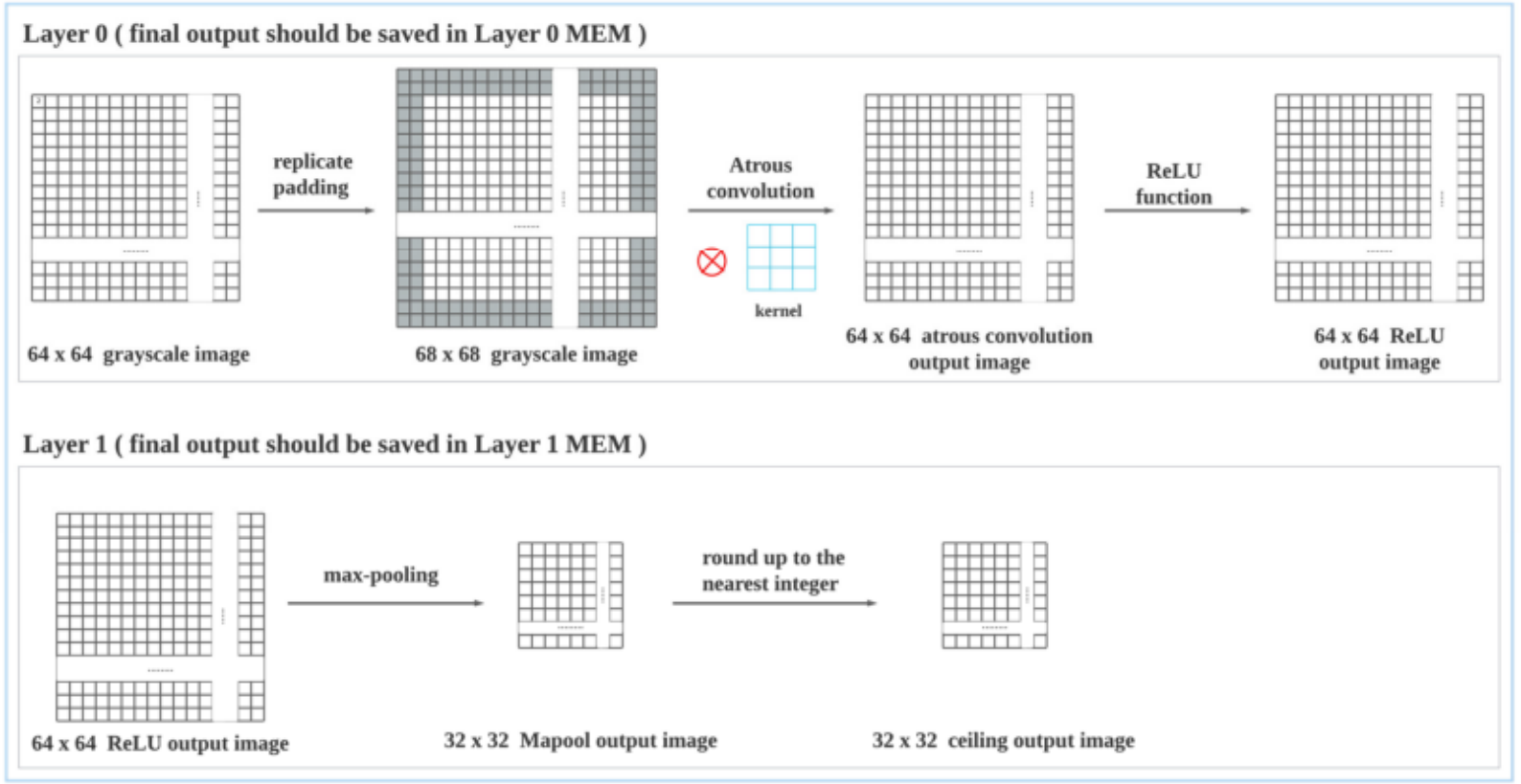


Homework 4 explanation

NCKU CSIE DICLAB

Introduction

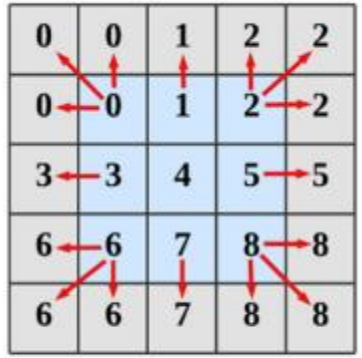
- Design a two-layered atrous convolution circuit
- Layer 0 consists of padding, atrous convolution, and ReLU
- Layer 1 consists of max-pooling and rounding up



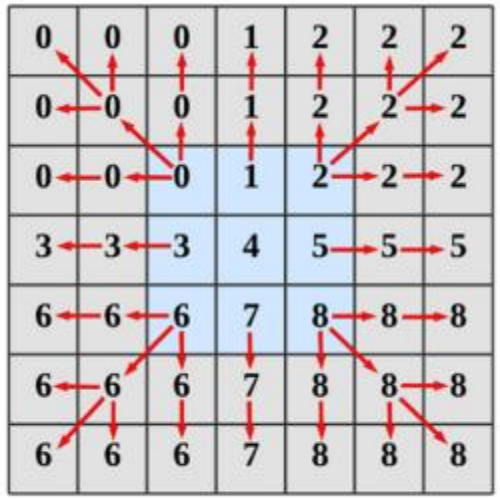
Introduction

- To maintain the same image size after atrous convolution, replicate padding is adopted
- In this homework, the *padding* parameter is set as 2
- The 64x64 image becomes 68x68 after the replicate padding operation

3x3 image



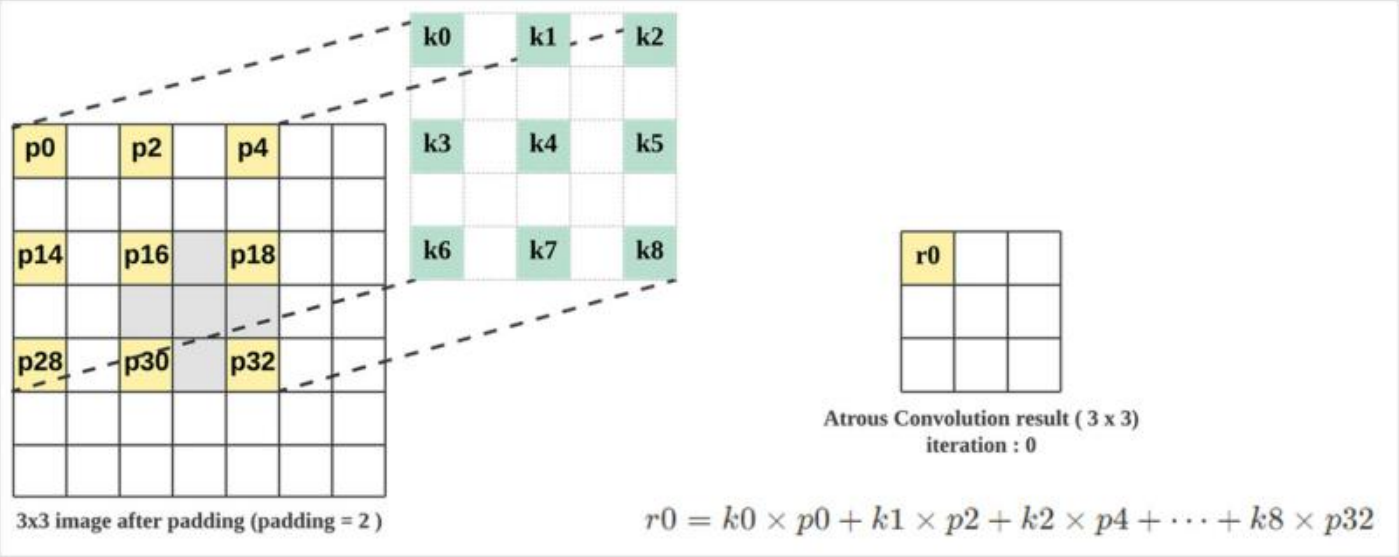
padding = 1



padding = 2

Introduction

- The hyperparameter *dilation* is set as 2, and *stride* is set as 1
- 9 pixel values are multiplied with the kernel values and then summed up
- The summation results have to be added with *bias* and pass through ReLU function
- Finally, the atrous convolution result for the center pixel is obtained



r0	r1	r2
r3	r4	r5
r6	r7	r8

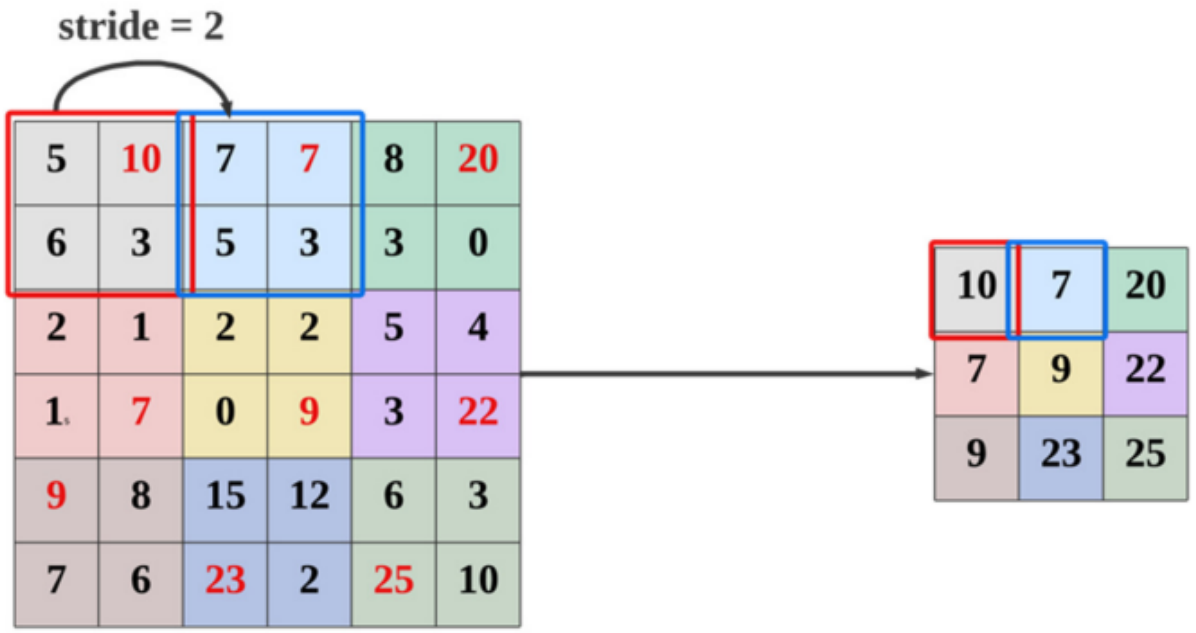
+

bias	bias	bias
bias	bias	bias
bias	bias	bias

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

Introduction

- The kernel size of max-pooling is 2x2 and the hyperparameter *stride* is 2
- The maximum among the four values will be selected



Finite State Machine

1. INIT：Wait for *ready* signal
2. ATCONV_9PIXELS：Read 9 pixels from IMAGE_MEM and calculate convolution
3. LAYER0_WRITERELU：Write the results after ReLU operation to LAYER 0 MEM
4. MAXPOOL_4PIXELS：Read 4 pixels from LAYER 0 MEM and find the maximum
5. LAYER1_WRITECEILING：Write rounded up results to LAYER 1 MEM
6. FINISH：Pull down *busy* signal

Data Registers

- center: used to record the coordinate of processing pixel
- counter: count the number of read in pixels
- convSum: used to accumulate the convolution results

```
36    //regs
37    reg [2:0] state, nextState;
38    reg [11:0] center; // Coordinate (row, column) = (center[11:6], center[5:0])
39    reg [3:0] counter;
40    reg signed [25:0] convSum; // {mul_integer(18bits), mul_fraction(8bits)}
```

Reset

- Pulled down *busy* to 0
- Set *cwr* as 0 to prevent writing invalid value to MEM
- Set *center* as {6'd0, 6'd0}, which corresponds to the pixel (0, 0)
- Initialize *counter* to 0
- Set *convSum* as {{9{1'b1}}, bias, 4'd0}
  - {9{1'b1}}：sign extension
  - bias：pre-add to *convSum*
  - 4'd0：pad zeros to the fractional part

```
73    always @(posedge clk or posedge reset) begin
74        if (reset) begin
75            busy <= 1'd0;
76            iaddr <= 12'd0;
77            cwr <= 1'd0;
78            caddr_wr <= 12'd0;
79            cdata_wr <= 13'd0;
80            crd <= 1'd1;
81            caddr_rd <= 12'd0;
82            csel <= 1'd0;
83
84            center <= {6'd0 , 6'd0};
85            counter <= 4'd0;
86            convSum <= {{9{1'b1}}, bias, 4'd0}; // Sign extension
87        end
```

ATCONV_9PIXELS

- Execute 10 cycles
  - 1st cycle：start reading pixel from IMAGE_MEM (*counter* = 0)
  - 2nd to 10th cycles：accumulate convolution result (*counter* = 1 ~ 9)
- *crd* is set as 1, and *cwr* is set as 0 (read operation)

```
96      ATCONV_9PIXELS:begin
97          csel <= 1'd0;
98          crd <= 1'd1;
99          cwr <= 1'd0;
100
101         // use the pixel get and conv with corresponding kernel ( counter==0 means no pixel get yet )
102         if(counter > 4'd0) begin
103             convSum <= convSum + idata*kernel[counter];
104         end
105         counter <= counter + 4'd1;
```

ATCONV_9PIXELS

- From the 1st to 9th cycles (*counter* = 0 ~ 8), address of the next pixel has to be assigned
- *center*[11:6] represents y coordinate (row), and *center*[5:0] represents x coordinate (column)
- If the pixel to read is out of bound, replicate padding is adopted
  - When the center is at (X, 0) or (X, 1), the y coordinate of the 1st row of kernel is set as 0
  - When the center is at (X, 62) or (X, 63), the y coordinate of the 3rd row of kernel is set as 63
  - When the center is at (0, X) or (1, X), the x coordinate of the 1st column of kernel is set as 0
  - When the center is at (62, X) or (63, X), the x coordinate of the 3rd column of kernel is set as 63
- Otherwise
  - The y coordinate of the 1st row of kernel is set as the y coordinate of *center* minus 2
  - The y coordinate of the 3rd row of kernel is set as the y coordinate of *center* plus 2
  - The x coordinate of the 1st column of kernel is set as the x coordinate of *center* minus 2
  - The x coordinate of the 3rd column of kernel is set as the x coordinate of *center* plus 2

ATCONV_9PIXELS


```
107                 // request the next corresponding pixel for Atrous convolution
108                 case (counter) // -> for y axis (row)
109                     0,1,2: iaddr[11:6] <= ((center[11:6] == 6'd0) || (center[11:6] == 6'd1))? ZERO : cy_minus2;
110                     3,4,5: iaddr[11:6] <= center[11:6];
111                     6,7,8: iaddr[11:6] <= ((center[11:6] == LENGTH - 6'd1) || (center[11:6] == LENGTH))? LENGTH : cy_add2;
112                 endcase
113
114                 case (counter) // -> for x axis (column)
115                     0,3,6: iaddr[5:0] <= ((center[5:0] == 6'd0) || (center[5:0] == 6'd1))? ZERO : cx_minus2;
116                     1,4,7: iaddr[5:0] <= center[5:0];
117                     2,5,8: iaddr[5:0] <= ((center[5:0] == LENGTH - 6'd1) || (center[5:0] == LENGTH))? LENGTH : cx_add2;
118                 endcase
```

LAYER0_WRITERELU

- Select LAYER 0 MEM and pull up *cwr* signal
- *caddr_wr* is set as center, and the value of *center* is increased by 1
  - The value of center will traverse from 0 to 4095
  - When *center*[5:0] is 63 (6'b111111), increment it by 1 will change to the next row and go back to column 0
  - Ex: {000000 111111} + 12'd1 = {000001 000000}
- *cdata_wr* is assigned the result after ReLU operation
  - *convSum*[25] = 1：*convSum* < 0, *cdata_wr* = 0
  - *convSum*[25] = 0：*convSum* ≥ 0, *cdata_wr* = *convSum*[16:4] (The left-most 9 bits and right-most 4 bits are truncated)

```
121             LAYER0_WRITERELU: begin
122                 csel <= 1'd0;
123                 crd <= 1'd0;
124                 cwr <= 1'd1;
125                 caddr_wr <= center;
126                 cdata_wr <= (convSum[25])? 13'd0 : convSum[16:4]; // ReLU
127                 // init the convSum and center --> kernel move to the next center and ready for atrous convolution
128                 convSum <= {{9{1'b1}}, bias, 4'd0};
129                 center <= center + 12'd1;
130                 counter <= 4'd0;
131             end
```

MAXPOOL_4PIXELS

- Select LAYER 0 MEM. Pull up *crd* signal and pull down *cwr* signal (read operation).
  - Read convolution results from LAYER 0 MEM
- Execute 5 cycles
  - 1st cycle：start reading pixel from LAYER 0 MEM (*counter* = 0)
  - 2nd to 5th cycles：record the maximum among data read from LAYER 0 MEM (*counter* = 1 ~ 4)
- Cdata_wr is used as the register that records the maximum value, and it is initialized as 0 at 1st cycle

```
133          MAXPOOL_4PIXELS: begin
134              csel <= 1'd0;
135              crd <= 1'd1;
136              cwr <= 1'd0;
137
138              // counter==0 means this cycle would send request for 1st pixel value, else comparison starts
139              if (counter==0) begin
140                  cdata_wr <= 13'd0;
141              end
142              else if (cdata_rd > cdata_wr) begin
143                  cdata_wr <= cdata_rd;
144              end
145              counter <= counter + 4'd1;
```

MAXPOOL_4PIXELS

- From the 1st to 4th cycles (*counter* = 0 ~ 3), address of the next pixel has to be assigned
- *center*[9:5] represents y coordinate (row), and *center*[4:0] represents x coordinate (column)
  - *center* is used to record the index of processing pixel
  - The size of the resulting image after max-pooling is 32x32, so *center* will traverse from 0 to 1023
  - However, there are pixels 0~4095 in LAYER 0 MEM
  - The x, y coordinate have to be multiplied by 2, and offset is used to select requiring pixel
- Let processing pixel is (X, Y), pixels (2X, 2Y)、(2X + 1, 2Y)、(2X, 2Y + 1)、(2X + 1, 2Y + 1) are read from LAYER 0 MEM
  - *counter* = 0，*caddr_rd* = {*center*[9:5]*2, *center*[4:0]*2}
  - *counter* = 1，*caddr_rd* = {*center*[9:5]*2, *center*[4:0]*2 + 1}
  - *counter* = 2，*caddr_rd* = {*center*[9:5]*2 + 1, *center*[4:0]*2}
  - *counter* = 3，*caddr_rd* = {*center*[9:5]*2 + 1, *center*[4:0]*2 + 1}

MAXPOOL_4PIXELS


```
147                 // request the corresponding address' pixel value
148 ▼               case(counter) // -> for y axis  (row)
149                     0,1: caddr_rd[11:6] <= {center[9:5], 1'd0};
150                     2,3: caddr_rd[11:6] <= {center[9:5], 1'd1};
151                 endcase
152
153 ▼               case(counter) // -> for x axis  (column)
154                     0,2: caddr_rd[5:0] <= {center[4:0], 1'd0};
155                     1,3: caddr_rd[5:0] <= {center[4:0], 1'd1};
156                 endcase
157             end
158
```

LAYER1_WRITECEILING

- Select LAYER 1 MEM and pull up *cwr* signal
- *caddr_wr* is set as center, and the value of *center* is increased by 1
  - The value of center will traverse from 0 to 1023
  - When *center*[4:0] is 31 (5'b11111), increment it by 1 will change to the next row and go back to column 0
  - Ex: {00 00000 11111} + 12'd1 = {00 00001 00000}
- *cdata_wr* is assigned the result that is rounded up
  - Integer part：
    - If any bit among *cdata_wr*[3:0] is 1 -> *cdata_wr*[12:4] + 9'd1
    - Else -> *cdata_wr*[12:4]
  - Fractional part：4'd0

```
159             LAYER1_WRITECEILING: begin
160                 csel <= 1'd1;
161                 crd <= 1'd0;
162                 cwr <= 1'd1;
163                 caddr_wr <= center;
164                 cdata_wr <= { cdata_wr[12:4] + {8'd0,|cdata_wr[3:0]} , 4'd0 }; // Round up
165                 // init for next center -> kernel move to the next center
166                 center <= center + 12'd1;
167                 counter <= 4'd0;
168             end
```

FINISH

- Pull down *busy* signal

```
170                 FINISH: begin
171                     busy <= 1'd0;
172                 end
```

- After *busy* is pulled down, testbench will verify the value in LAYER 0 MEM and LAYER 1 MEM