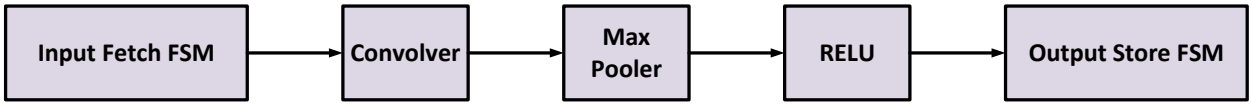
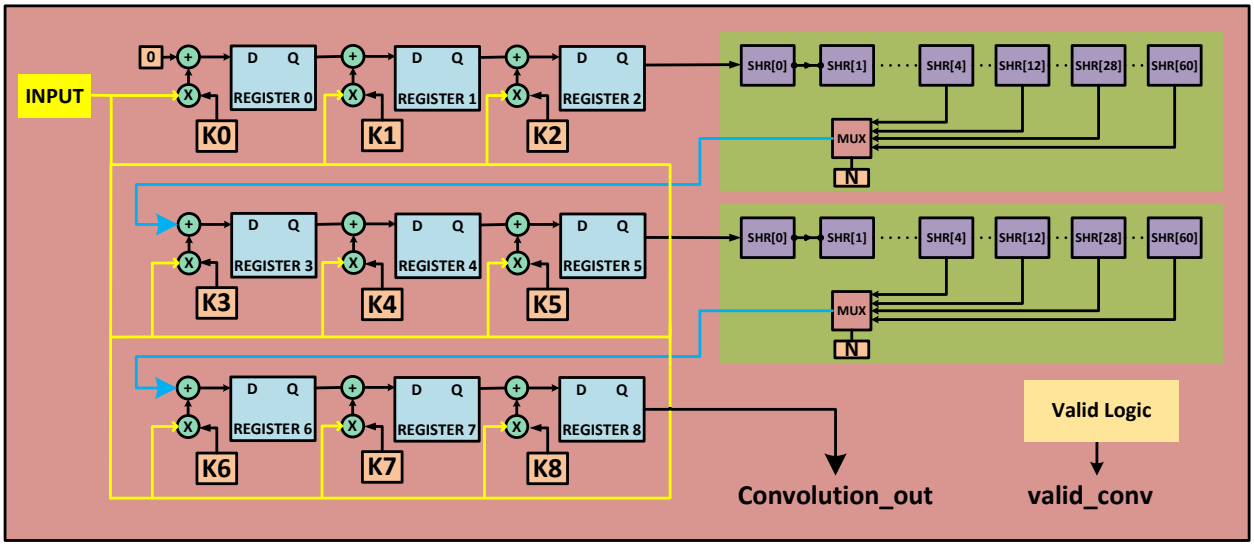
Efficient Systolic Array Convolution with Pipelined MaxPooling and ReLU

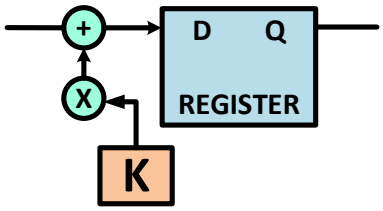
High-level Description of the System



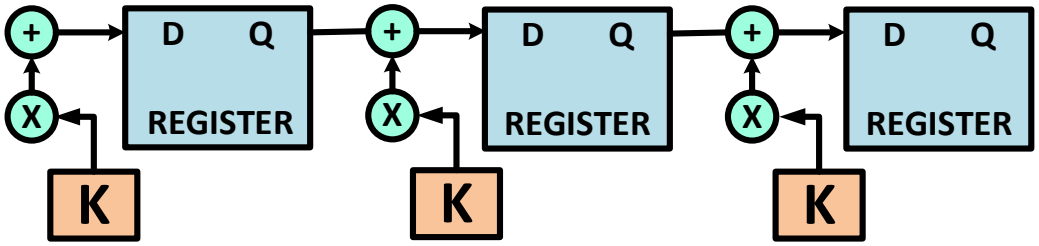
Convolver:



The system utilizes a systolic array-based architecture for efficient convolution computation. It consists of 9 processing elements, each functioning as a Multiply-Accumulate (MAC) unit with an associated register.



A single input is given to all the MAC's from index (0,0) to (N, N) shown by yellow in the block diagram. I call 3 MAC's together a cascade.



To understand the flow better look at this 4x4 input image example.

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

And the 3x3 kernel.

K0	K1	K2
K3	K4	K5
K6	K7	K8

To perform convolution, we will have to multiply similar colors then add them up.

After that we will have to move the window by stride 1

K0	K1	K2
K3	K4	K5
K6	K7	K8

Convolved with

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

OP0 = K0*C0 + K1*C1 + K2*C2 + K3*C4 + K4*C5 + K5*C6 + K6*C8 + K7*C9 + K8*C10

K0	K1	K2
K3	K4	K5
K6	K7	K8

Convolved with

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

OP1 = K0*C1 + K1*C2 + K2*C3 + K3*C5 + K4*C6 + K5*C7 + K6*C9 + K7*C10 + K8*C11

K0	K1	K2
K3	K4	K5
K6	K7	K8

Convolved with

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

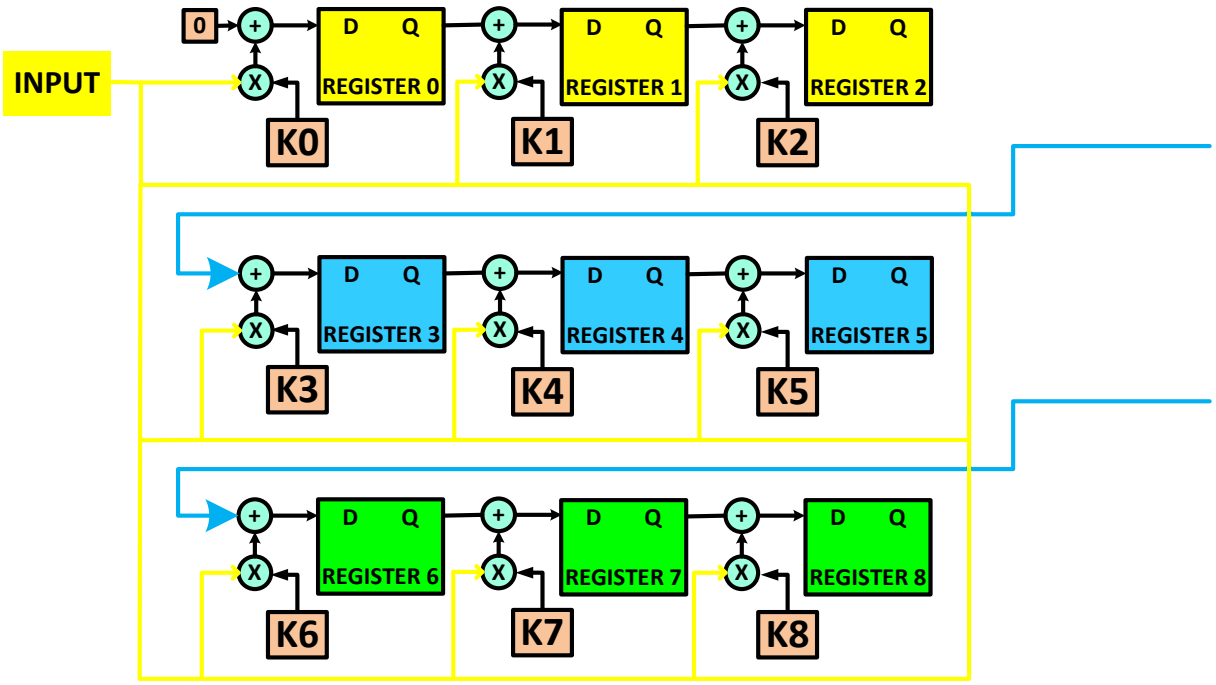
OP2 = K0*C4 + K1*C5 + K2*C6 + K3*C8 + K4*C9 + K5*C10 + K6*C12 + K7*C13 + K8*C14

K0	K1	K2
K3	K4	K5
K6	K7	K8

Convolved with

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

OP3 = K0*C5 + K1*C6 + K2*C7 + K3*C9 + K4*C10 + K5*C11 + K6*C13 + K7*C14 + K8*C15



Three cascades of MAC units are employed to compute the outputs for each color component (Yellow, Blue, Green). This approach allows for parallel computation and efficient pipelining.

- We have 9 MAC units i.e. 3 Cascades.
- Yellow input will be computed using the first Cascade
- Blue input will be computed using the second Cascade
- Green input will be computed using the third Cascade

The main idea here is to compute yellow of OP0 first and store them in a shift register.

When blue of OP0 is being computed the stored yellow of OP0 in shift register is added to it and stored in another shift register.

Finally when green of OP0 is computed the summation of yellow and blue of OP0 stored in a shift register is also added to green to get a convolution output

Now when blue of OP0 is being computed yellow does not stop it is computing the yellow of OP1

And when green of OP0 is being computed yellow of OP2 and blue of OP1 are parallelly being computed

As a result, we get OP0, OP1, OP2, OP0 each after 1 cycle from the previous output.

Only two garbage values are encountered during the computation, which has minimal impact on the overall efficiency The garbage values are due to convolution of below highlighted input.

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

So output is as follows O0 O1 X X O2 O3

This is not that bad because we get only 2 garbage values for any value if N and because the way our pipeline is structured the two X X won't slow us down.

Eg: for N=8 output is as follows

O0 O1 O2 O3 O4 O5 O6 X X O7 O8 O9 O10 O11 O12 O13 X X ….O35

To detect this I created a valid convolution logic that tells when to ignore output.

So in summary It takes 2*N + 3 cycles to load the pipeline as we need 2*N+3 input values to perform first valid convolution but after that for every input we get a valid output.

i.e after the last input in the next cycle we get the last output.

In the following section, I have elucidated the functioning of the shift registers in storing MAC (Multiply-Accumulate) outputs. However, to save time and avoid delving into less engaging details, I recommend proceeding directly to the more intriguing topic of MaxPooling+ReLu logic.

As we have fixed number of N values and we need N-3 shift registers lets create shift registers of width 61 to accommodate max value of N which is 64 then select the output of the SHR based on N.

Note : For a 4x4 image only 1 shift reg is required the 2nd column is supposed to represent the output of SHR.

C0	C1	C2	C3
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

First 3 MAC will compute Yellow

The Yellow computation is performed by the first 3 MAC units. Then stored in a shift Register SHR0.

K0*C1 + K1*C2 + K2*C3	K0*C0 + K1*C1 + K2*C2

SHR0

~~C0~~	~~C1~~	~~C2~~	~~C3~~
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

First 3 MAC will compute Yellow

~~C0~~	~~C1~~	~~C2~~	~~C3~~
C4	C5	C6	C7
C8	C9	C10	C11
C12	C13	C14	C15

Next 3 MAC will compute Blue

The Blue computation if performed by the next 3 MAC units. And the output if SHR0 containing Yellow is added to the Blue computation and stored in SHR1.

K0*C5 + K1*C6 + K2*C7	K0*C4 + K1*C5 + K2*C6

SHR0

K0*C0 + K1*C1 + K2*C2+ K3*C5 + K4*C6 + K5*C7	K0*C0 + K1*C1 + K2*C2+ K3*C4 + K4*C5 + K5*C6+

SHR1

The Green computation is performed by the final 3 MAC units. And SHR1 outputs are added to it to get the final convolution output.

~~C0~~	~~C1~~	~~C2~~	~~C3~~
~~C4~~	~~C5~~	~~C6~~	~~C7~~
C8	C9	C10	C11
C12	C13	C14	C15

First 3 MAC will compute Yellow

~~C0~~	~~C1~~	~~C2~~	~~C3~~
~~C4~~	~~C5~~	~~C6~~	~~C7~~
C8	C9	C10	C11
C12	C13	C14	C15

Next 3 MAC will compute Blue

~~C0~~	~~C1~~	~~C2~~	~~C3~~
~~C4~~	~~C5~~	~~C6~~	~~C7~~
C8	C9	C10	C11
C12	C13	C14	C15

Last 3 MAC will compute Green

-	-

SHR0

K0*C5 + K1*C6 + K2*C7+ K3*C9 + K4*C10 + K5*C11	K0*C4 + K1*C5 + K2*C6+ K3*C8 + K4*C9 + K5*C10

SHR1

K0*C0 + K1*C1 + K2*C2+ K3*C5 + K4*C6 + K5*C7 K6*C9 + K7*C10 + K8*C11 OP1	K0*C0 + K1*C1 + K2*C2+ K3*C4 + K4*C5 + K5*C6+ K6*C8 + K7*C9 + K8*C10 OP0

Output

Next output

~~C0~~	~~C1~~	~~C2~~	~~C3~~
~~C4~~	~~C5~~	~~C6~~	~~C7~~
~~C8~~	~~C9~~	~~C10~~	~~C11~~
C12	C13	C14	C15

~~C0~~	~~C1~~	~~C2~~	~~C3~~
~~C4~~	~~C5~~	~~C6~~	~~C7~~
~~C8~~	~~C9~~	~~C10~~	~~C11~~
C12	C13	C14	C15

~~C0~~	~~C1~~	~~C2~~	~~C3~~
~~C4~~	~~C5~~	~~C6~~	~~C7~~
~~C8~~	~~C9~~	~~C10~~	~~C11~~
C12	C13	C14	C15

First 3 MAC will compute Yellow Next 3 MAC will compute Blue Last 3 MAC will compute Green

-	-

SHR0

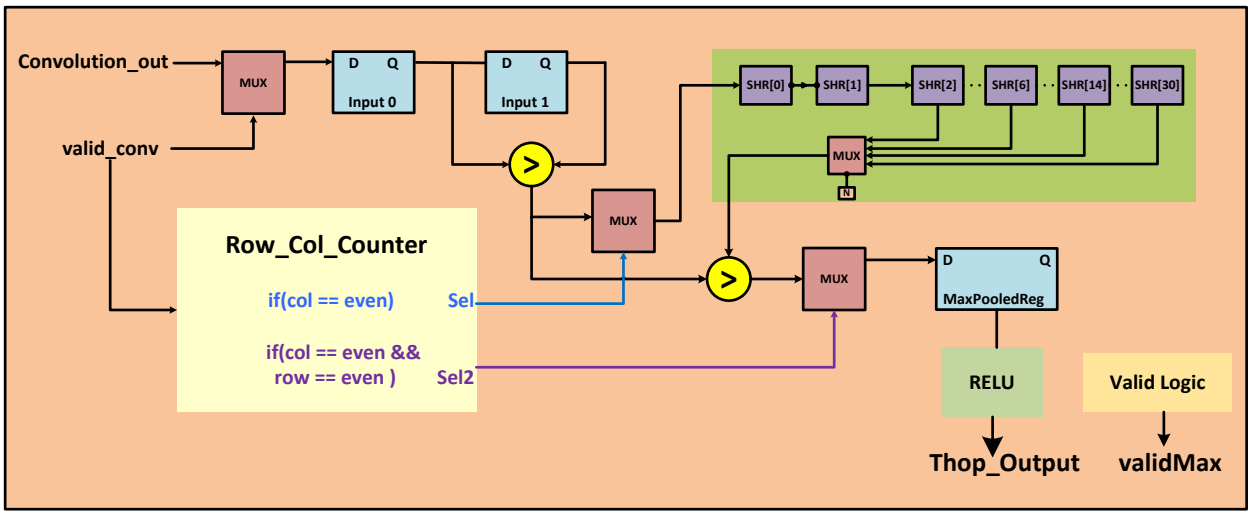
-	-

SHR1

K0*C5 + K1*C6 + K2*C7+ K3*C9 + K4*C10 + K5*C11 K6*C13 + K7*C14 + K8*C15 OP3	K0*C4 + K1*C5 + K2*C6+ K3*C8 + K4*C9 + K5*C10 K6*C12 + K7*C13 + K8*C14 OP2

Output

MaxPooler:



Now that we have out convolution output

OP0	OP1
OP2	OP3

We don't want to store them and then perform maxpooling as that would throw away the entire efficiency obtained by our convolver.

What we need is a maxpooler that takes convolution output as soon as our convolver generates a valid output and produces max pooled output in 1 cycle too.

Below is the working of my maxpooler with relu.

We are comparing OP0 with OP1 then storing the greatest value in SHR.

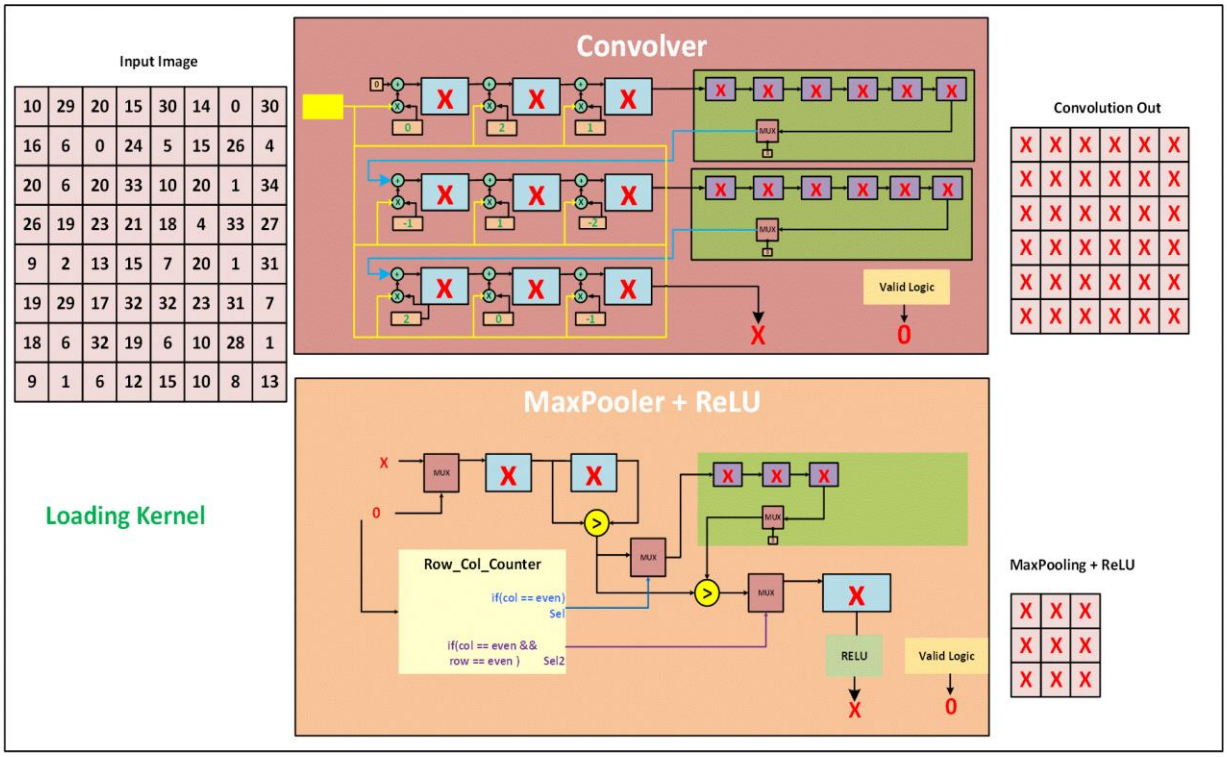
Then OP1 and OP2 are compared but not stored in SHR as the select line will not be active. As OP2 is odd col.

When OP3 and OP3 are compared the max of them is then again compared to the SHR output which will be max(OP1, OP2) hence the output will be Max(OP0,OP1,OP2,OP3).

Both Sel,Sel2 will be active as OP3 is even col and in even row.

Finally this is passed to RELU to get our first final output.

To facilitate better understanding, I have prepared a GIF with an example of an 8x8 input matrix that demonstrates the complete convolution and Maxpooling process. Please refer to the GIF for a clearer visualization of the procedure.



Results:

For input consider this 16 x 16 matrix

```
[14, 25, 7, 43, 7, 23, 28, 19, 10, 12, 41, 3, 44, 27, 10],
[36, 0, 41, 38, 44, 4, 29, 34, 27, 42, 5, 27, 30, 20, 11, 4],
[28, 17, 15, 7, 19, 31, 25, 18, 30, 44, 19, 29, 0, 37, 5, 41],
[17, 25, 28, 18, 22, 15, 27, 34, 42, 4, 27, 32, 34, 6, 28, 17],
[24, 25, 33, 2, 39, 3, 3, 15, 23, 27, 2, 0, 5, 17, 20, 33],
[3, 27, 37, 15, 21, 1, 15, 9, 39, 10, 14, 42, 9, 1, 33, 27],
[4, 17, 37, 34, 1, 0, 2, 13, 16, 14, 25, 4, 40, 15, 26, 7],
[10, 27, 18, 19, 19, 43, 1, 41, 24, 39, 35, 27, 42, 41, 9, 22],
[8, 28, 11, 33, 2, 20, 30, 36, 20, 42, 12, 25, 24, 16, 4, 40],
[4, 7, 0, 31, 26, 26, 20, 25, 36, 29, 3, 31, 37, 8, 3, 1],
[1, 26, 8, 30, 42, 41, 41, 42, 24, 17, 11, 10, 26, 3, 25, 0],
[27, 14, 35, 38, 21, 17, 42, 3, 25, 29, 40, 15, 11, 5, 24, 38],
[36, 41, 7, 20, 33, 15, 8, 43, 30, 3, 2, 8, 36, 9, 28, 33],
[20, 7, 37, 15, 33, 14, 6, 37, 38, 10, 18, 43, 5, 21, 31, 34],
[9, 21, 25, 11, 12, 23, 11, 34, 22, 41, 36, 12, 20, 20, 23, 38]
```

The last input is 38

The MaxPooled + ReLu output is as follows

```
[ 51. 127.  75. 118.  78. 104.  80.]
[127. 108. 101. 107. 122. 114. 118.]
[117. 127.  51.  95.  99. 127.  74.]
[ 84.  63.  60. 106. 124.  62. 127.]
[ 78.  49.  80. 102. 127. 127. 127.]
[ 53.  99. 127. 112.  54. 117.  21.]
[ 65.  85. 102. 124. 125. 119.  47.]
```

The last output is 47

Observations made at a clock period of 10ns revealed the following timings:

1. At 855ns, the first input is dispatched to the convolver.
2. At 1205ns, the first valid convolution output is obtained.
3. At 1395ns, the first valid output of the combined MaxPooling and ReLU process is achieved.
4. At 3405ns, the final input is received.
5. At 3415ns, the last convolution output is obtained.
6. At 3425ns, the last output of the combined MaxPooling and ReLU process is obtained, requiring 2 cycles from the last input.

Total 256 cycles were required for fetching the input, all 16x16 values and 258 cycles for generating output (convolution + MaxPooling +Relu) all 7x7 values.

It can be observed from below waveform that 2 extra cycles were needed for the production of the final output.