ECE 284: Low Power VLSI for ML
Final Project : Systolic Array Implementation

Naman Sehgal and Pranav Gangwar

Group Name : Bubble

Pytorch Implementation

```
(27): QuantConv2d(
  8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
(28): ReLU(inplace=True)
(29): QuantConv2d(
```

Print of Modified Model

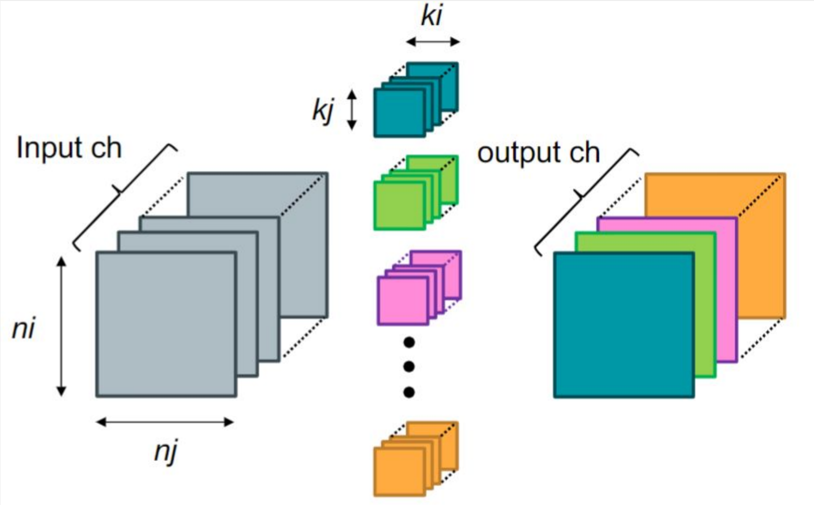


Figure from Lecture Slide “W4S1”

Activation(pre-padding)

- ni=nj=4
- Input Ch=8

Kernel

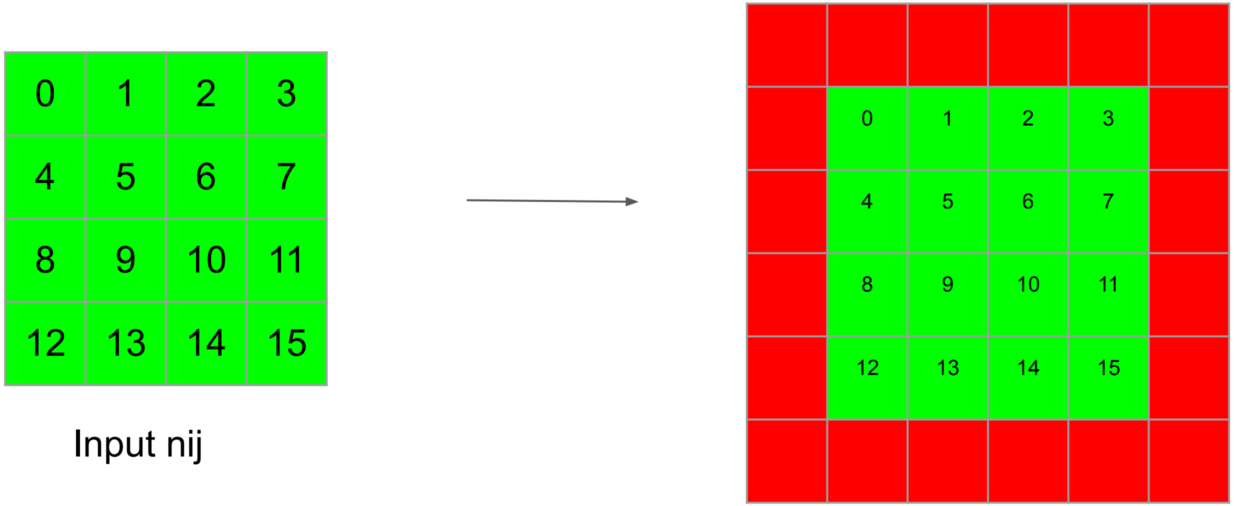
- ki=kj=3
- Input Ch=8
- Output Ch = 8

Output

- ni=nj=4
- Output Ch=8

Design Parameters

- nij_{in}=6x6 (Input padded with zeros to make it 6x6 from 4x4)



Input nij

Padded Input nij

Convolution Recap

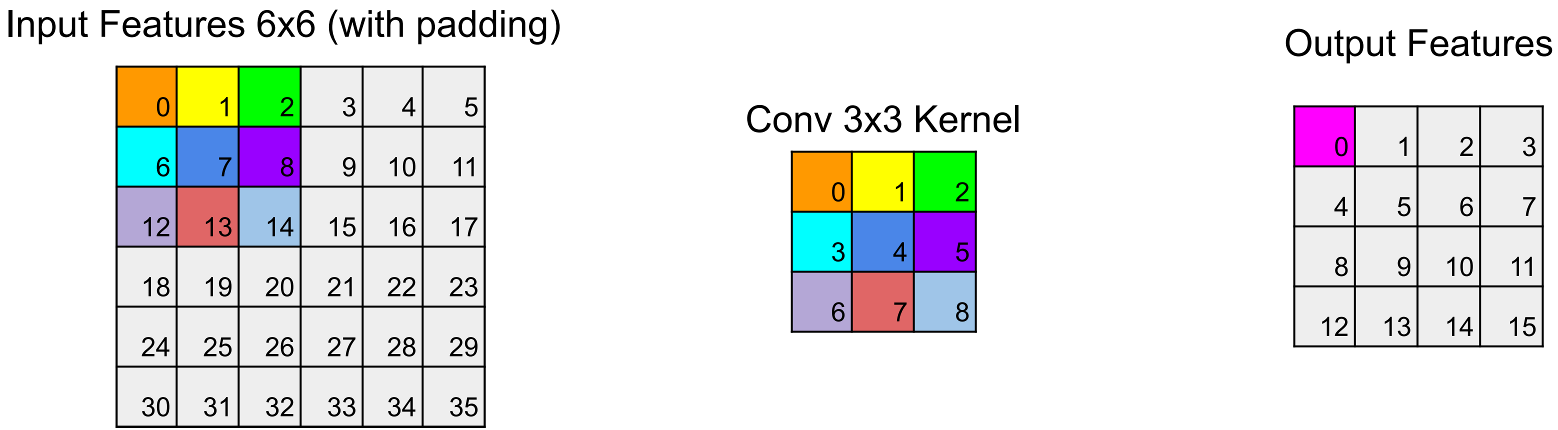
Input Features 6x6 (with padding)

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

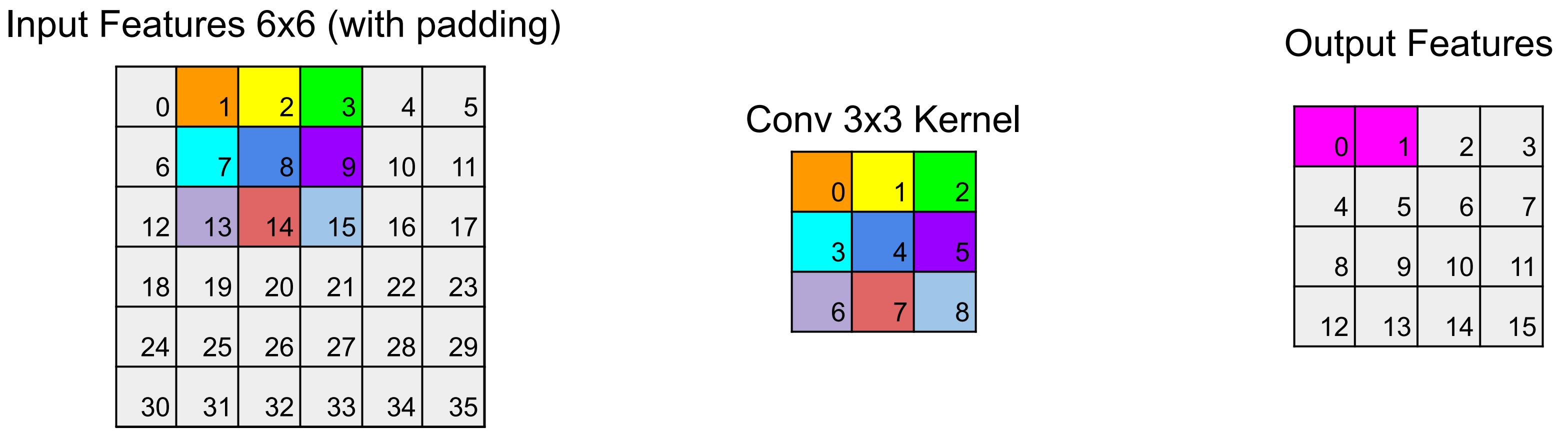
Conv 3x3 Kernel

0	1	2
3	4	5
6	7	8

$Nij_{out} = 0$

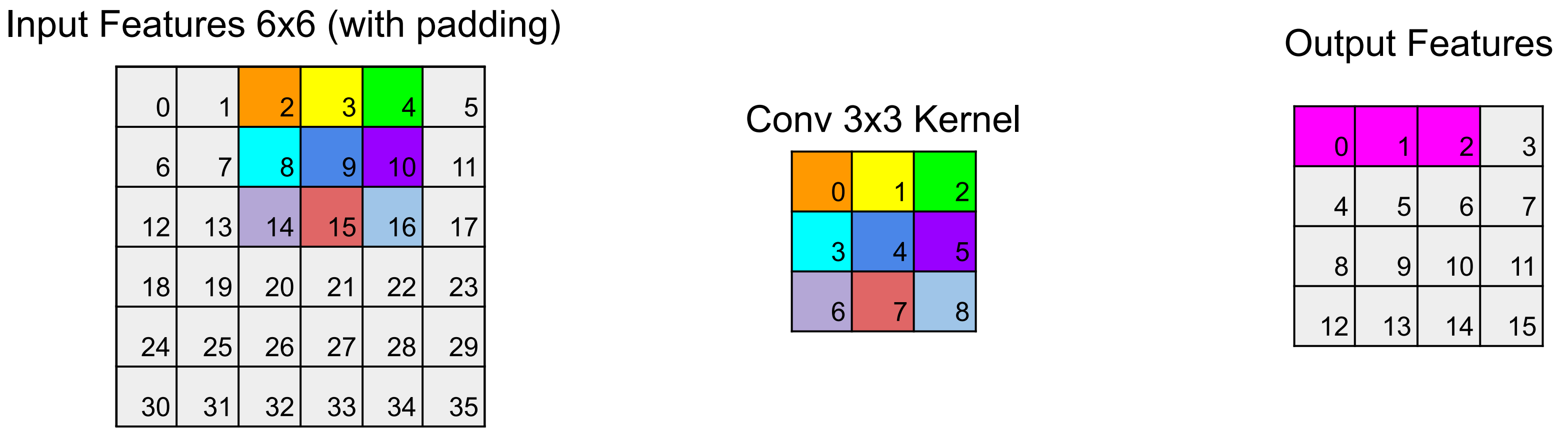

Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

$Nij_{out} = 1$



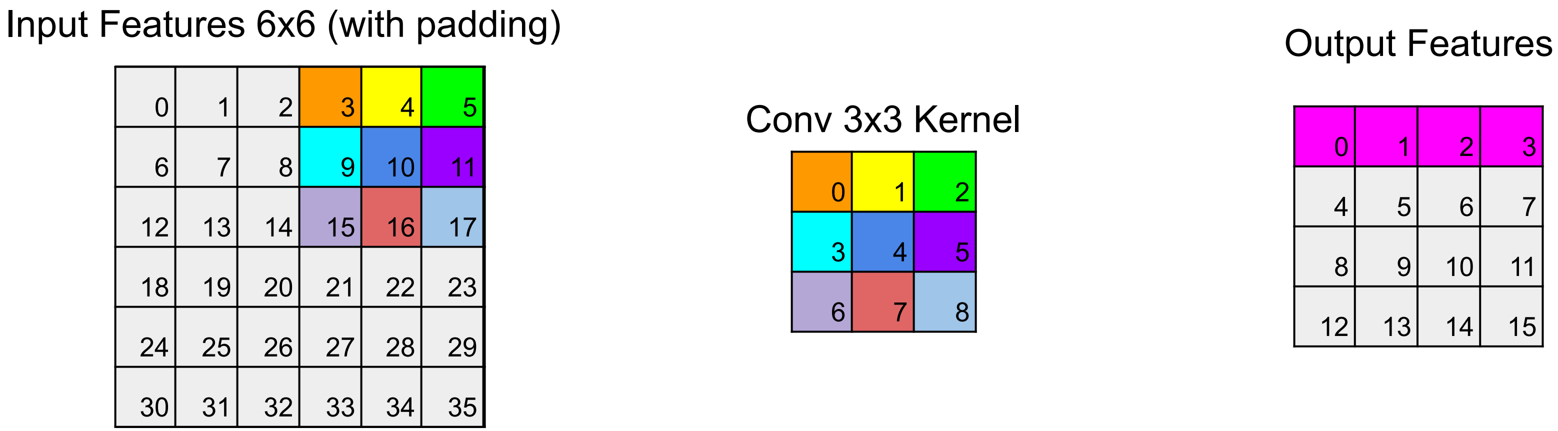
Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

$Nij_{out} = 2$



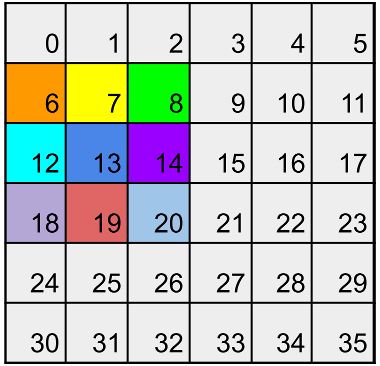
Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

$Nij_{out} = 3$

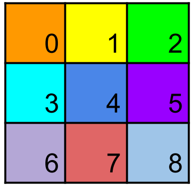

Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

$Nij_{out} = 4$

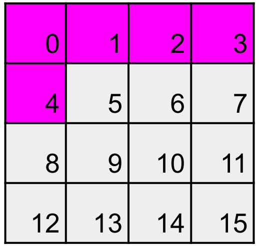
Input Features 6x6 (with padding)



Conv 3x3 Kernel

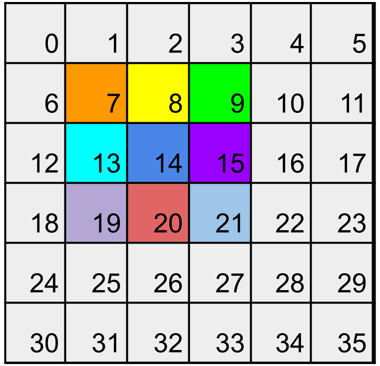


Output Features

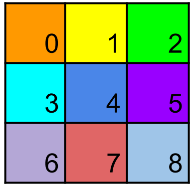


$Nij_{out} = 5$

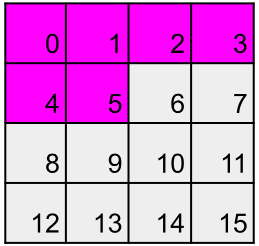
Input Features 6x6 (with padding)



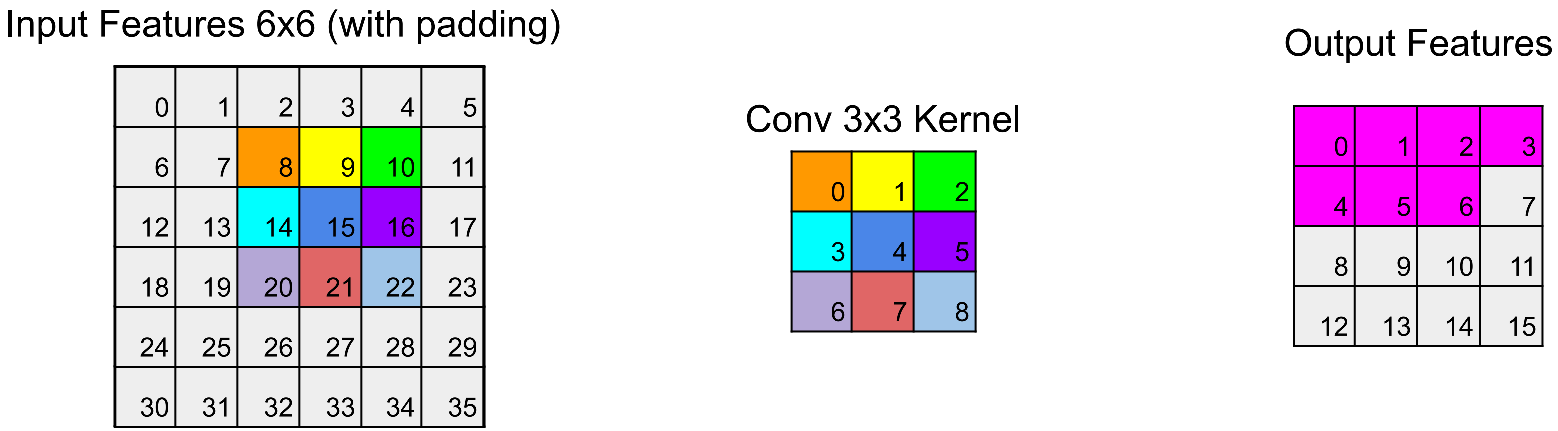
Conv 3x3 Kernel



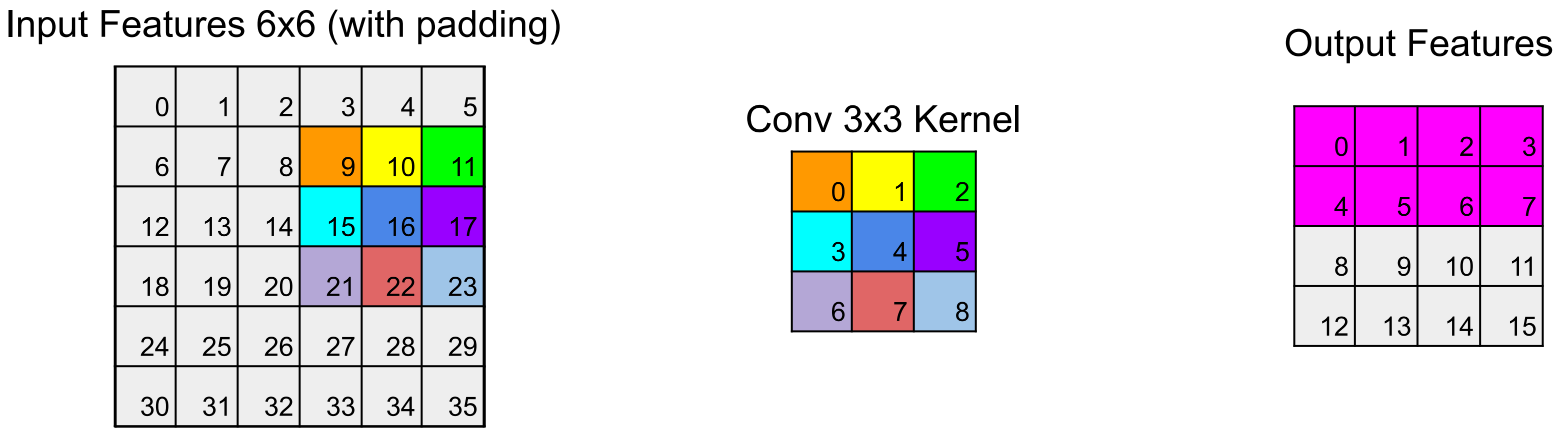
Output Features



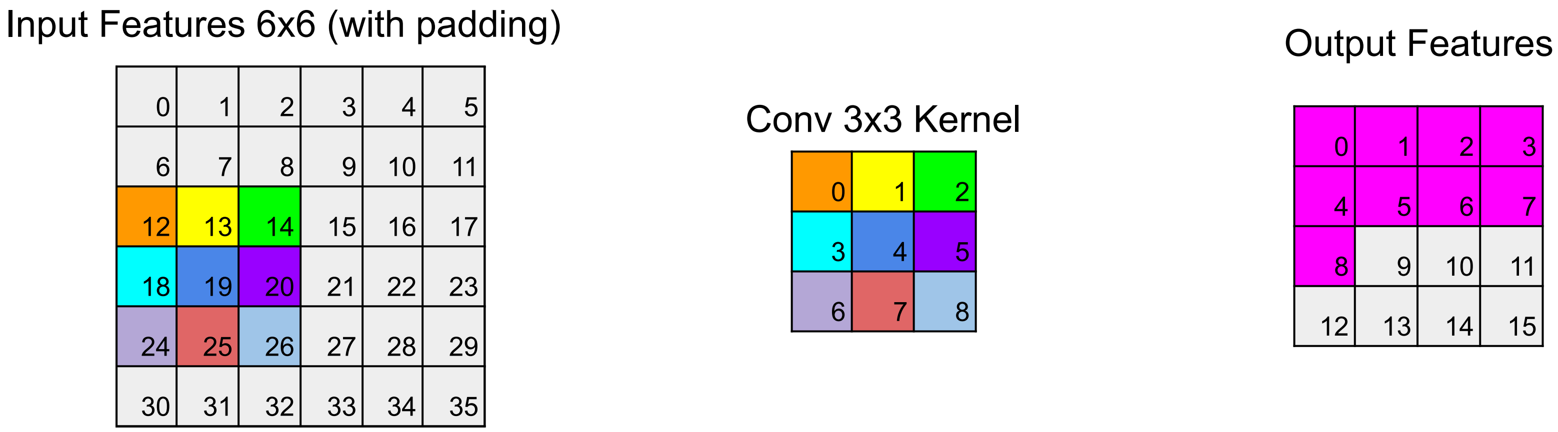
$Nij_{out} = 6$


Input Features 6x6 (with padding)
Conv 3x3 Kernel
Output Features

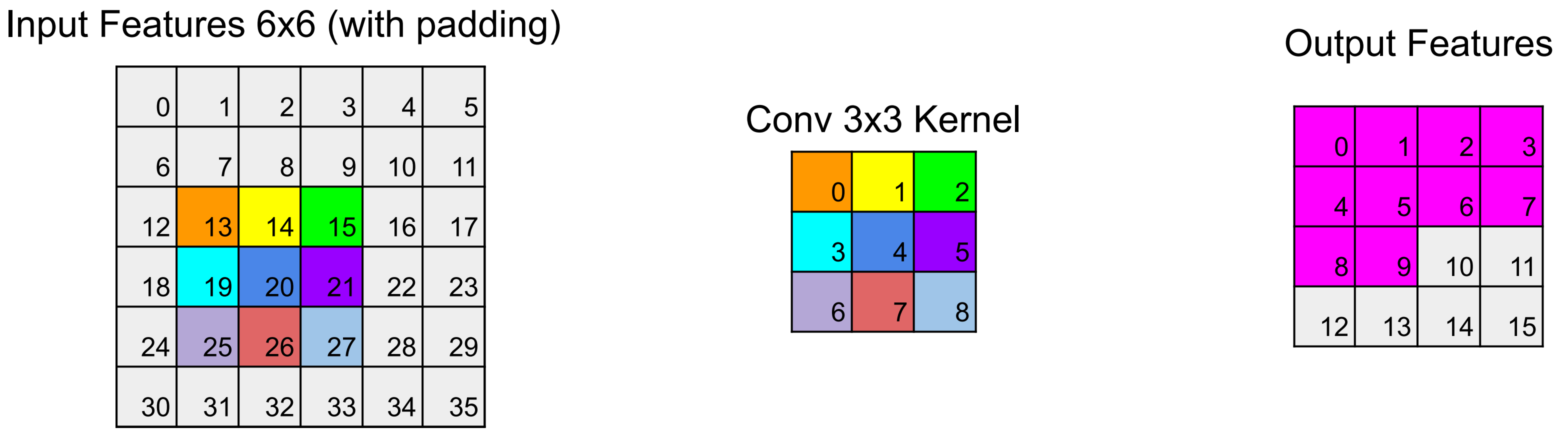
$Nij_{out} = 7$


Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

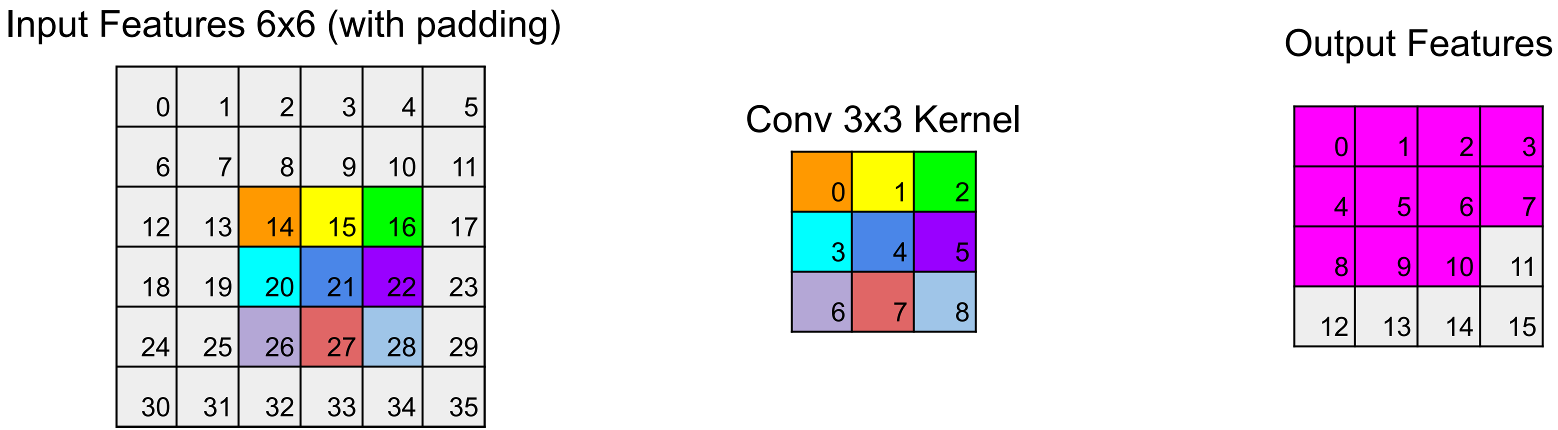
$Nij_{out} = 8$


Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

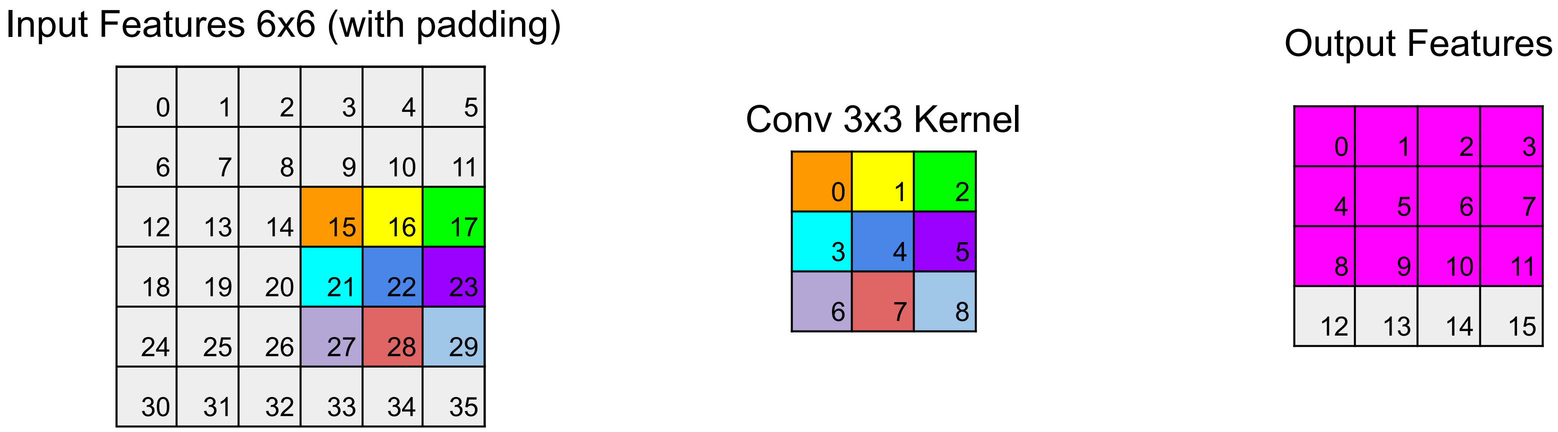
$Nij_{out} = 9$


Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

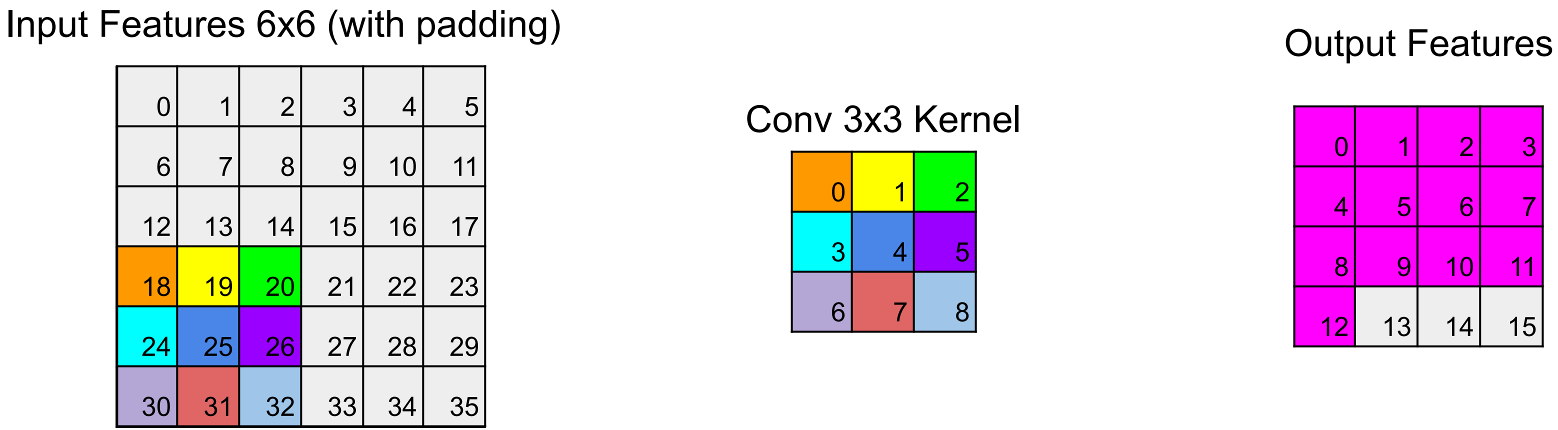
$Nij_{out} = 10$


Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

$Nij_{out} = 11$

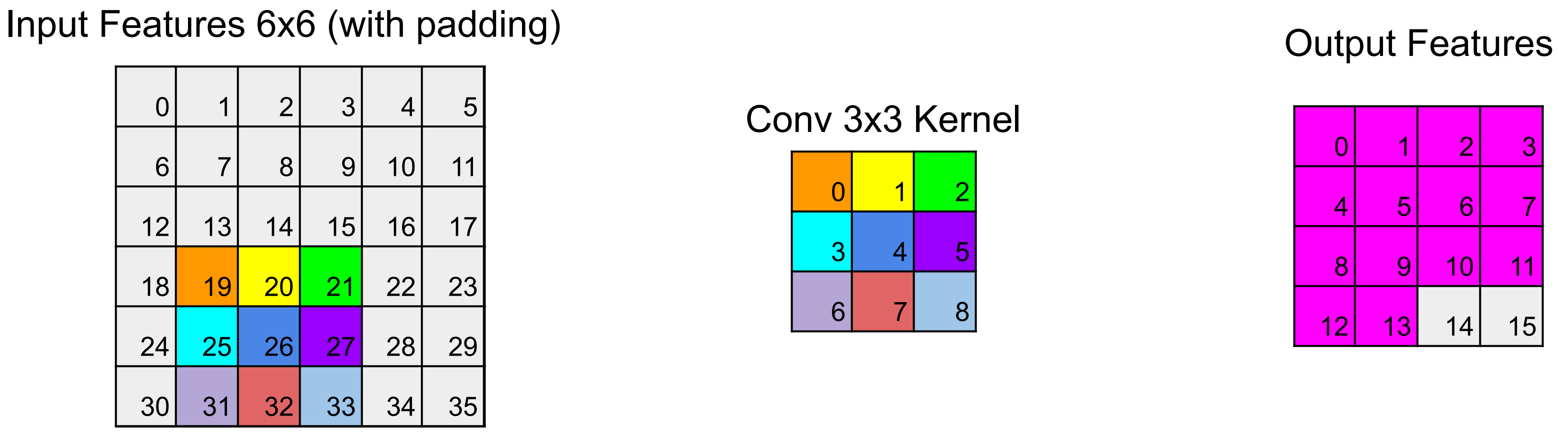

Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

$Nij_{out} = 12$



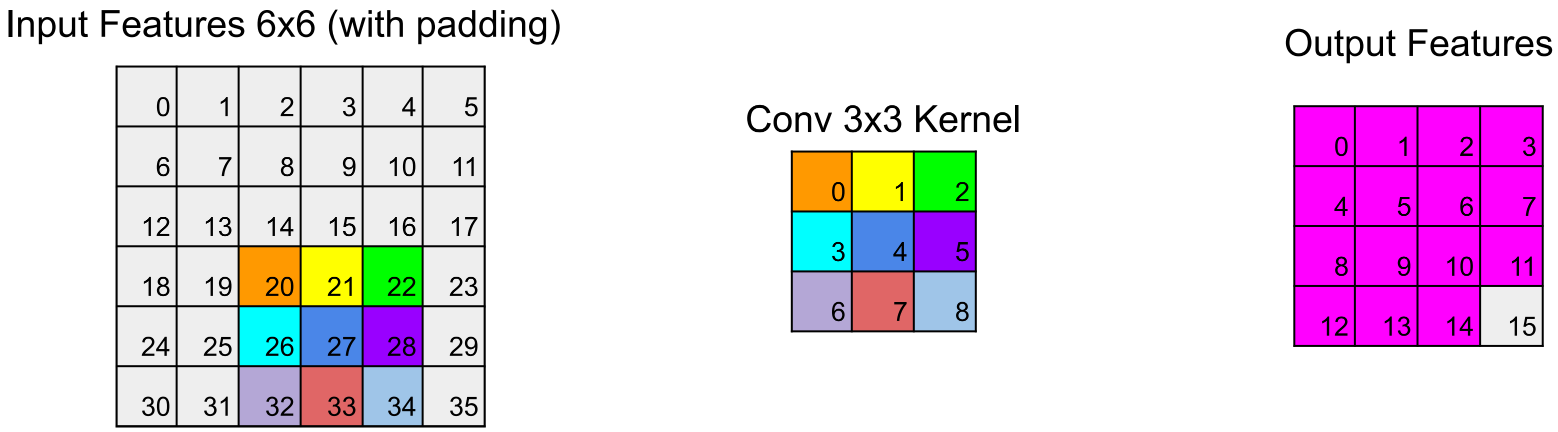
Input Features 6x6 (with padding)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
Conv 3x3 Kernel
0
1
2
3
4
5
6
7
8
Output Features
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

$Nij_{out} = 13$



Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

$Nij_{out} = 14$

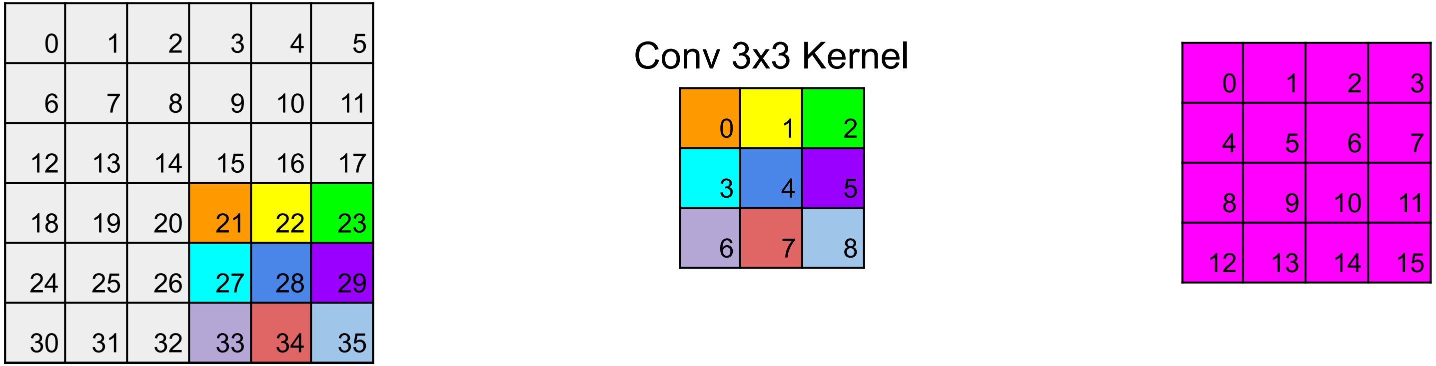

Input Features 6x6 (with padding)
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
Conv 3x3 Kernel
0 1 2
3 4 5
6 7 8
Output Features
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

$Nij_{out} = 15$

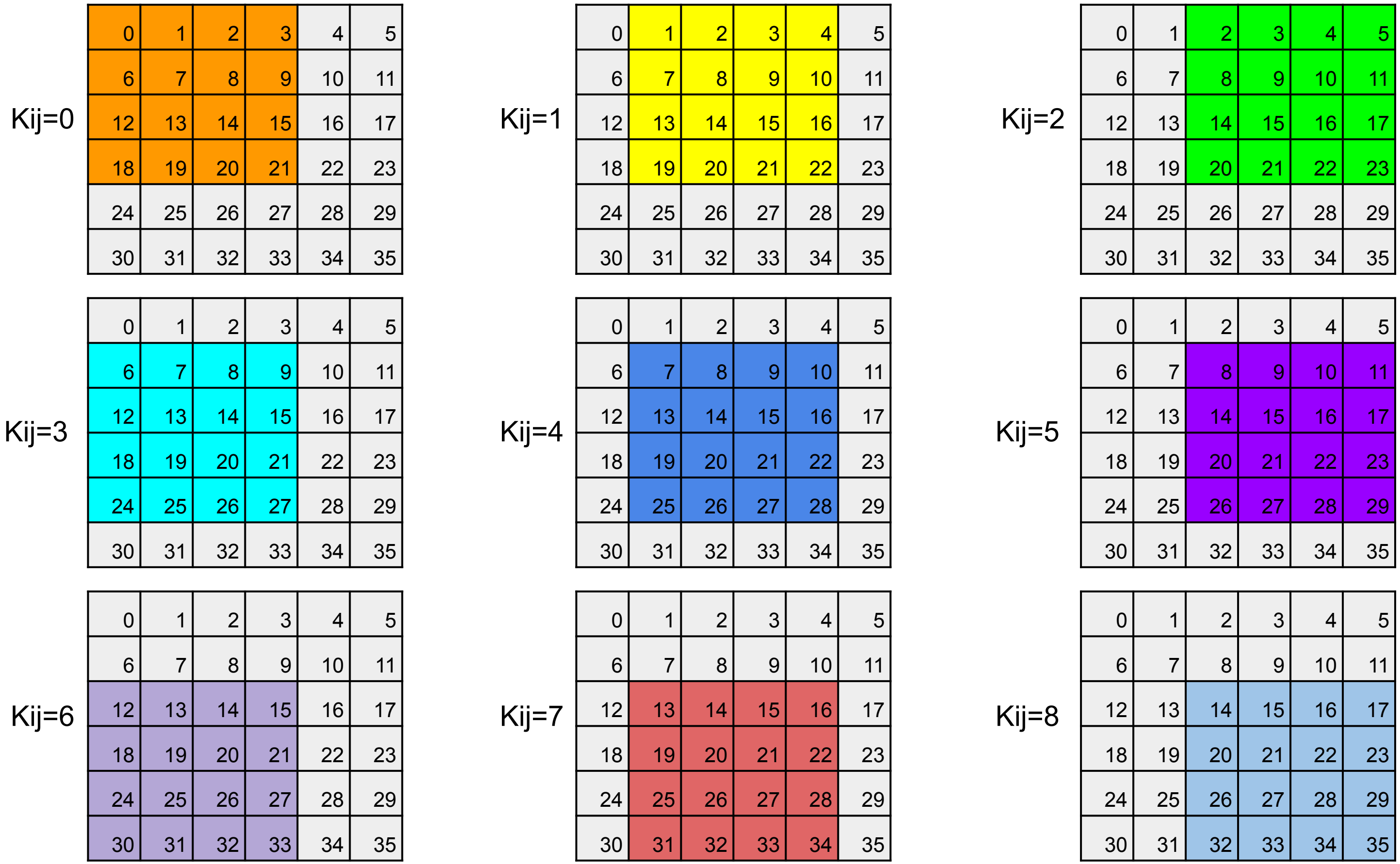
Input Features 6x6 (with padding)

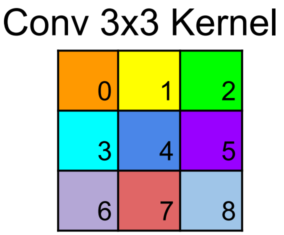
Conv 3x3 Kernel

Output Features



Input features utilisation mapped to Kernel Elements

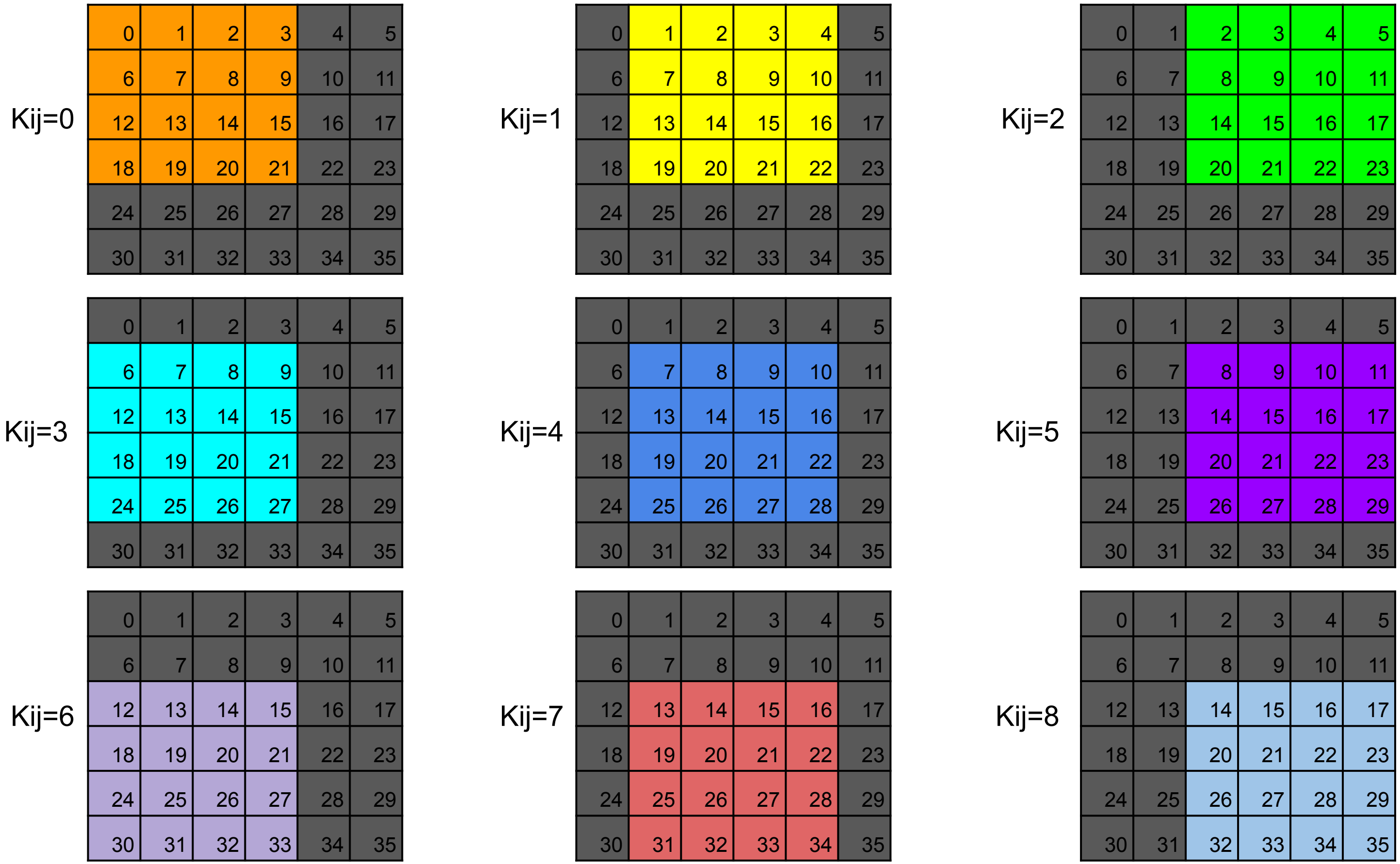


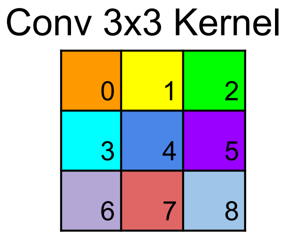


Observation:

White colored inputs are not used in final output.

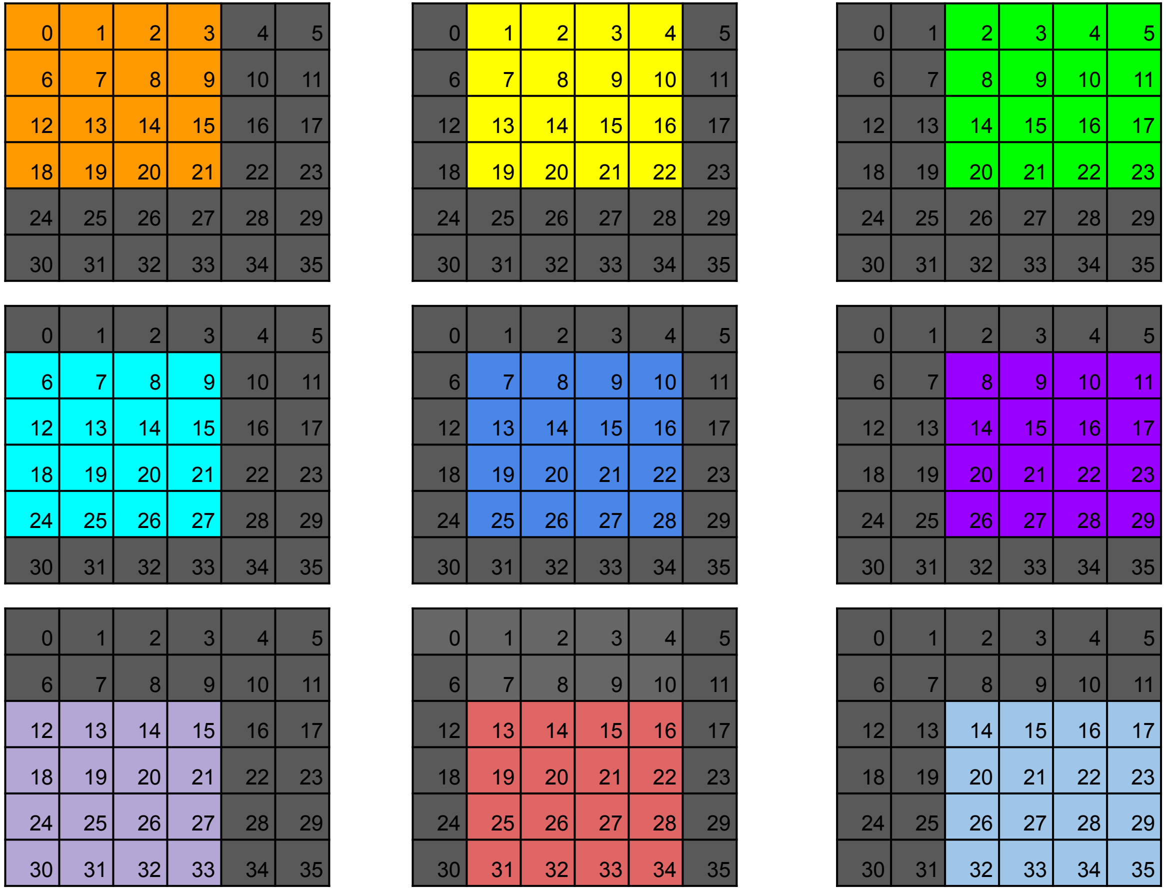
Alpha 1 (Sieving) : Input features utilisation mapped to Kernel Elements



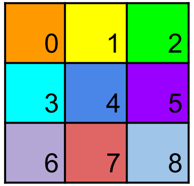


So we created a mathematical equation to sieve only the colored inputs from activation SRAM before loading them to L0

Modifying the Input features sent to L0



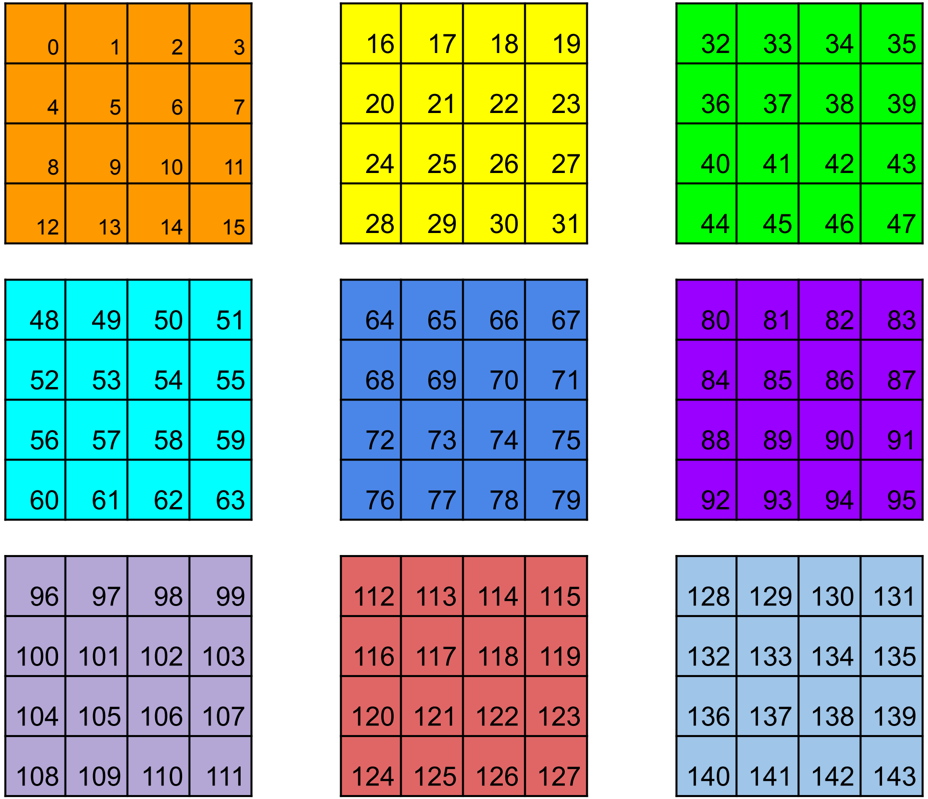
Conv 3x3 Kernel




```
for (kij=0 ; kij<9; kij++)
    for (nij=0 ; nij <16 ; nij++)
        nij_prime = int(nij/4)*6 + nij%4;
        kij_prime = int(kij/3)*6 + kij%3;
        data_L0 = x_in[ kij_prime + nij_prime];
```


```
for (kij=0 ; kij<9; kij++)
    for (nij=0 ; nij <16 ; nij++)
        nij_prime = int(nij/4)*6 + nij%4;
        kij_prime = int(kij/3)*6 + kij%3;
        data_L0 = x_in[ kij_prime + nij_prime];
```

Alpha 1 (Sieving) : Generated Psums and their indexing



Benefit of the Idea

- Lesser computations in Systolic Array
- Lesser memory required to store the Psums

There will be lower latency and lower energy usage due to both

	Computations in systolic array	Memory Required
Vanilla	324	324 words
Alpha 1: Sieving the inputs	144	144 words

Alpha 2

For $Nij_{out} = 0$

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

32	33	34	35
36	37	38	39
40	41	42	43
44	45	46	47

48	49	50	51
52	53	54	55
56	57	58	59
60	61	62	63

64	65	66	67
68	69	70	71
72	73	74	75
76	77	78	79

80	81	82	83
84	85	86	87
88	89	90	91
92	93	94	95

96	97	98	99
100	101	102	103
104	105	106	107
108	109	110	111

112	113	114	115
116	117	118	119
120	121	122	123
124	125	126	127

128	129	130	131
132	133	134	135
136	137	138	139
140	141	142	143

Alpha 2

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

32	33	34	35
36	37	38	39
40	41	42	43
44	45	46	47

48	49	50	51
52	53	54	55
56	57	58	59
60	61	62	63

64	65	66	67
68	69	70	71
72	73	74	75
76	77	78	79

80	81	82	83
84	85	86	87
88	89	90	91
92	93	94	95

96	97	98	99
100	101	102	103
104	105	106	107
108	109	110	111

112	113	114	115
116	117	118	119
120	121	122	123
124	125	126	127

128	129	130	131
132	133	134	135
136	137	138	139
140	141	142	143

For $Nij_{out} = 0$

- Symmetric relative addressing between the psums for output computation, so simple logic to call the psum to the SFU.

Similar pattern for rest of the Nij_{out} also

- Opportunity to make architectural changes to the SFU unit.

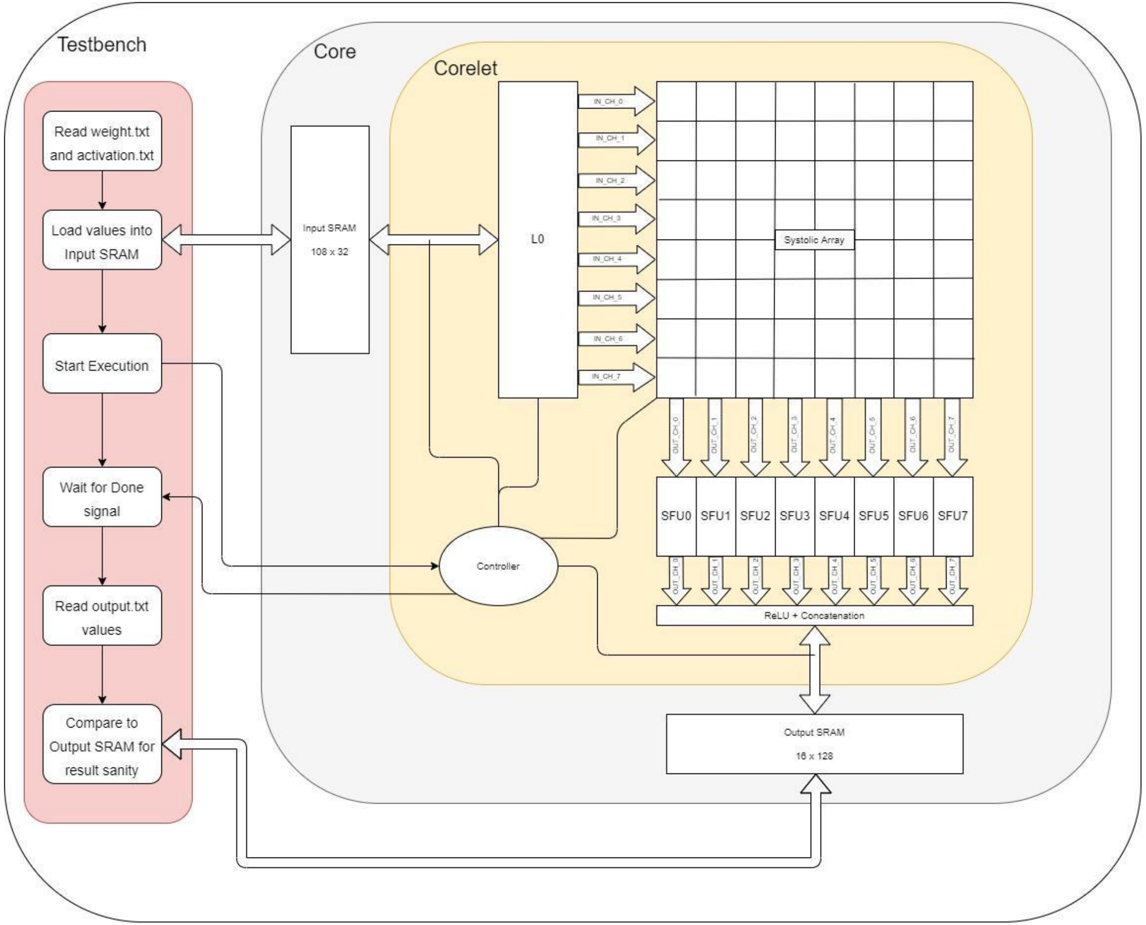
Alpha2: Modified SFU

After every kij iteration, instead of sending the data to psum SRAM, we are doing in-place accumulation

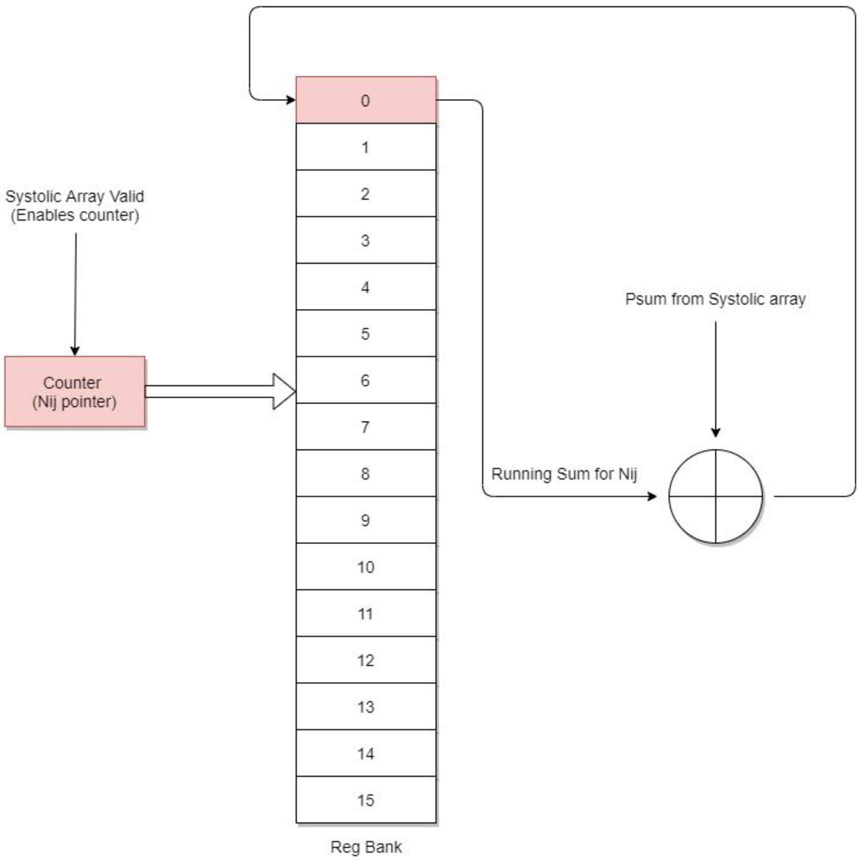
Architectural Changes

- Removed the OFIFO
- Replaced by 16 sets of registers sitting in the SFU to store values for Nij_{out} in the form of a running sum

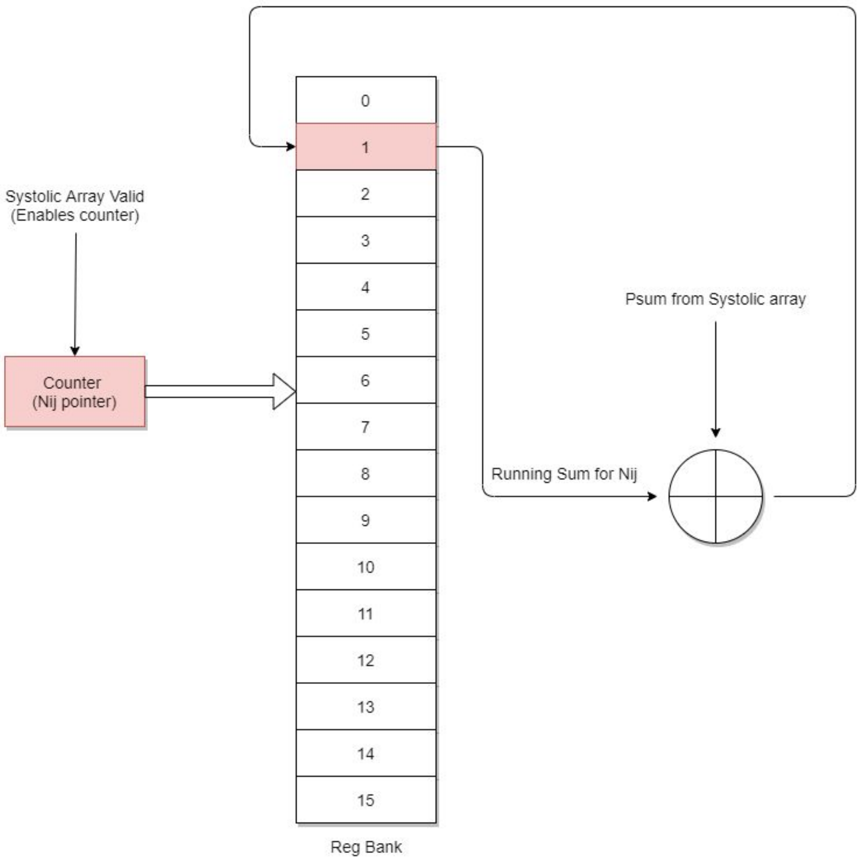
System Architecture



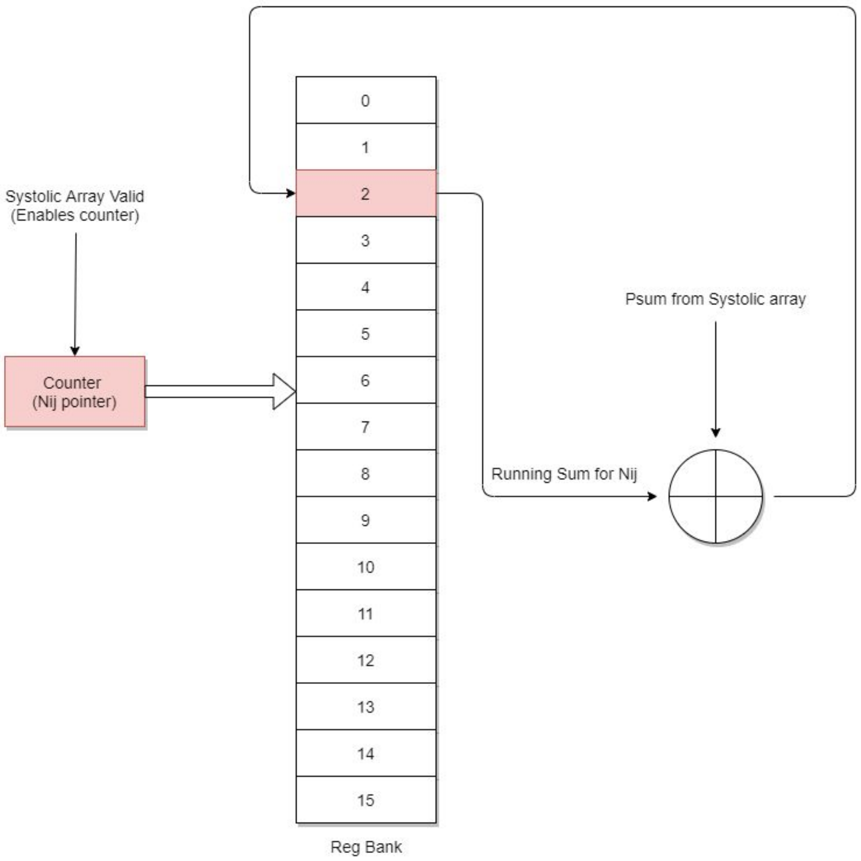
Alpha 2 (Modified SFU) working



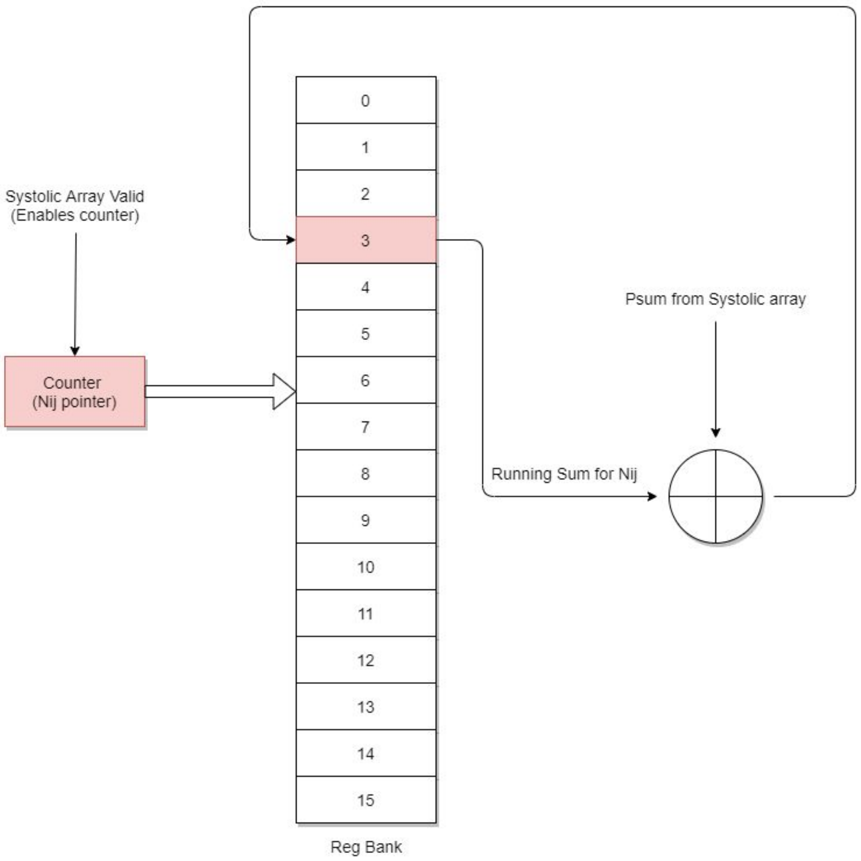
Alpha 2 (Modified SFU) working



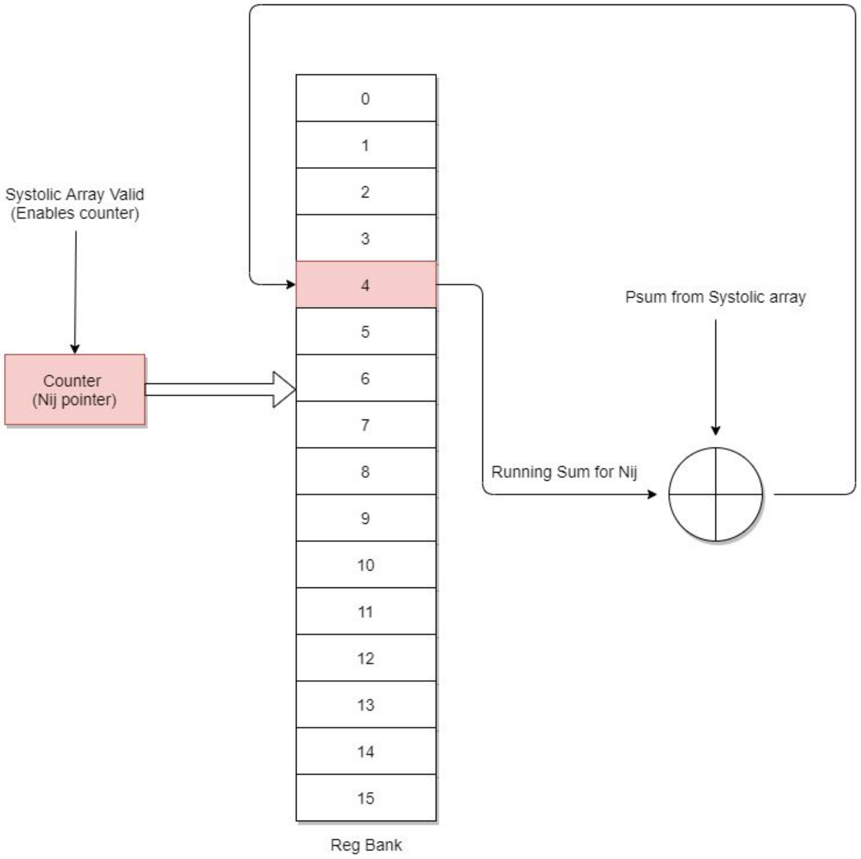
Alpha 2 (Modified SFU) working



Alpha 2 (Modified SFU) working



Alpha 2 (Modified SFU) working



….and so on

Alpha 2 (Modified SFU) Benefits

- Able to achieve computation in place without requirement of unnecessary SRAM reads and writes.
  - Zero added latency : Final outputs generated 1 clock cycle after the systolic array finishes execution of kij=8

- Requirement of huge SRAM to store intermediate psums removed
  - We do not need to store psums for each kij separately
  - Memory requirement reduced by a **factor of 9!!** (144 rows after alpha 1 vs only 16 rows after alpha2)

Results

1. Frequency: 123.85 MHz

2. Core Dynamic Power: 16.94mW

3. Verification: a) Core’s Output perfectly matches the one we got from Pytorch

   b) The difference between the computed output and the next layer’s input is ~ 10^(-7)

```
r=nn.ReLU()
next_layer_in_computed = r(output_recovered)
next_layer_in_ref = save_output.outputs[9][0]
difference = abs( next_layer_in_computed - next_layer_in_ref)
print(difference.mean())
```

```
tensor(1.3314e-07, device='cuda:0', grad_fn=<MeanBackward0>)
```

Slow 1200mV 100C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	123.85 MHz	123.85 MHz	clk	

Power Analyzer Status	Successful - Sat Nov 27 00:02:02 2021
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	ECE284-Final_Project
Top-level Entity Name	corelet
Family	Cyclone IV GX
Device	EP4CGX150DF31I7AD
Power Models	Final
Total Thermal Power Dissipation	256.48 mW
Core Dynamic Thermal Power Dissipation	16.94 mW
Core Static Thermal Power Dissipation	119.24 mW
I/O Thermal Power Dissipation	120.30 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

```
[pgangwar@ieng6-ece-03]:sim:683$ iveri filelist
[pgangwar@ieng6-ece-03]:sim:684$ irun
VCD info: dumpfile core_tb.vcd opened for output.
Checking the weights written into the I-SRAM from weight.txt
 0-th read data is b99a999c --- Data matched
 1-th read data is 1c307f7d --- Data matched
 2-th read data is 992a0497 --- Data matched
 3-th read data is 27bb9d46 --- Data matched
 4-th read data is 3c92a9f7 --- Data matched
 5-th read data is 7d091e4c --- Data matched
 6-th read data is 75dcee69 --- Data matched
 7-th read data is 179e79aa --- Data matched
 8-th read data is d9999cc9 --- Data matched
 9-th read data is 1342c774 --- Data matched
10-th read data is ef5da1df --- Data matched
11-th read data is 31dec6a4 --- Data matched
12-th read data is 7dc29c75 --- Data matched
13-th read data is 3b71fa46 --- Data matched
14-th read data is 7cdfae79 --- Data matched
15-th read data is 6000710d --- Data matched
16-th read data is 9999cb99 --- Data matched
17-th read data is 37d095be --- Data matched
18-th read data is 44a5a9e5 --- Data matched
19-th read data is 4acca591 --- Data matched
20-th read data is 3d67913d --- Data matched
21-th read data is 094adde7 --- Data matched
22-th read data is 7b11e979 --- Data matched
23-th read data is 29cfd21c --- Data matched
24-th read data is 999dbdca --- Data matched
25-th read data is 1ca77b23 --- Data matched
26-th read data is 1d5101b4 --- Data matched
27-th read data is f750c54d --- Data matched
28-th read data is 0fcf1077 --- Data matched
29-th read data is 4d4e309b --- Data matched
30-th read data is a703d551 --- Data matched
31-th read data is d7f3e52d --- Data matched
32-th read data is 9c99d999 --- Data matched
33-th read data is 73e7211f --- Data matched
34-th read data is 737143e6 --- Data matched
35-th read data is 277617f7 --- Data matched
36-th read data is 400b6d73 --- Data matched
37-th read data is d077e092 --- Data matched
38-th read data is 2401d04f --- Data matched
39-th read data is 0300760d --- Data matched
40-th read data is aa999a9a --- Data matched
41-th read data is 7097d5ce --- Data matched
42-th read data is 7507ecc7 --- Data matched
43-th read data is 9c949417 --- Data matched
44-th read data is 3f7a773f --- Data matched
45-th read data is 9977c056 --- Data matched
46-th read data is 1e70e979 --- Data matched
47-th read data is feca46f0 --- Data matched
48-th read data is 99bb99dc --- Data matched
49-th read data is 9bc76bde --- Data matched
50-th read data is 2374f0bb --- Data matched
51-th read data is f790900d --- Data matched
52-th read data is 1eddd0f7 --- Data matched
53-th read data is 3ad6979b --- Data matched
54-th read data is e61da775 --- Data matched
55-th read data is 792df7c9 --- Data matched
56-th read data is 999b9d99 --- Data matched
57-th read data is ca356be1 --- Data matched
58-th read data is 73795e10 --- Data matched
59-th read data is b7047f23 --- Data matched
60-th read data is 20cc5735 --- Data matched
61-th read data is 7f2707cf --- Data matched
```

```
60-th read data is 20cc5735 --- Data matched
61-th read data is 7f2707cf --- Data matched
62-th read data is ad46b640 --- Data matched
63-th read data is 797057d2 --- Data matched
64-th read data is b9999999 --- Data matched
65-th read data is 39b6be3d --- Data matched
66-th read data is 7309abd0 --- Data matched
67-th read data is b4241ad6 --- Data matched
68-th read data is ee1d72d2 --- Data matched
69-th read data is f234d77d --- Data matched
70-th read data is b0469f6e --- Data matched
71-th read data is 093c5776 --- Data matched
Checking the activations written into the I-SRAM from activation.txt
 0-th read data is 00000000 --- Data matched
 1-th read data is 00000000 --- Data matched
 2-th read data is 00000000 --- Data matched
 3-th read data is 00000000 --- Data matched
 4-th read data is 00000000 --- Data matched
 5-th read data is 00000000 --- Data matched
 6-th read data is 00000000 --- Data matched
 7-th read data is 02330000 --- Data matched
 8-th read data is 32260041 --- Data matched
 9-th read data is 31350132 --- Data matched
10-th read data is 10120040 --- Data matched
11-th read data is 00000000 --- Data matched
12-th read data is 00000000 --- Data matched
13-th read data is 03020020 --- Data matched
14-th read data is 15030071 --- Data matched
15-th read data is 34520252 --- Data matched
16-th read data is 02200251 --- Data matched
17-th read data is 00000000 --- Data matched
18-th read data is 00000000 --- Data matched
19-th read data is 00000000 --- Data matched
20-th read data is 21000050 --- Data matched
21-th read data is 30520140 --- Data matched
22-th read data is 00110140 --- Data matched
23-th read data is 00000000 --- Data matched
24-th read data is 00000000 --- Data matched
25-th read data is 02120011 --- Data matched
26-th read data is 02020040 --- Data matched
27-th read data is 01530052 --- Data matched
28-th read data is 01220050 --- Data matched
29-th read data is 00000000 --- Data matched
30-th read data is 00000000 --- Data matched
31-th read data is 00000000 --- Data matched
32-th read data is 00000000 --- Data matched
33-th read data is 00000000 --- Data matched
34-th read data is 00000000 --- Data matched
35-th read data is 00000000 --- Data matched
Comparing the outputs written into the O-SRAM by the Systolic Array with the output.txt
 0-th read data is 0000006d009f0000005a0047002c0000 --- Data matched
 1-th read data is 000000ac0079000000b5007d00150000 --- Data matched
 2-th read data is 000e00ed003b001c00ab0056000a0000 --- Data matched
 3-th read data is 0000009c000000300038004500000000 --- Data matched
 4-th read data is 0000005f001a00250000005000380000 --- Data matched
 5-th read data is 001600ce001700c7000000120004e0000 --- Data matched
 6-th read data is 006f0136001d00b9006b000000520000 --- Data matched
 7-th read data is 000f00a90000004d0039000000000000 --- Data matched
 8-th read data is 00000070000001e000000000200000 --- Data matched
 9-th read data is 000000fc001800a80000000000470000 --- Data matched
10-th read data is 00000115004900 4e0084000000 9f0000 --- Data matched
11-th read data is 0000009a000000000039000000490000 --- Data matched
12-th read data is 000000510013001000 0a001d00040000 --- Data matched
13-th read data is 000000c8004800a500000000000a0000 --- Data matched
14-th read data is 000000d6006b00760042000000580000 --- Data matched
15-th read data is 0000006f000000270023000000 6b0000 --- Data matched
[pgangwar@ieng6-ece-03]:sim:685$
```

Summarizing Novelty

1. Sending only relevant data to the Systolic Array

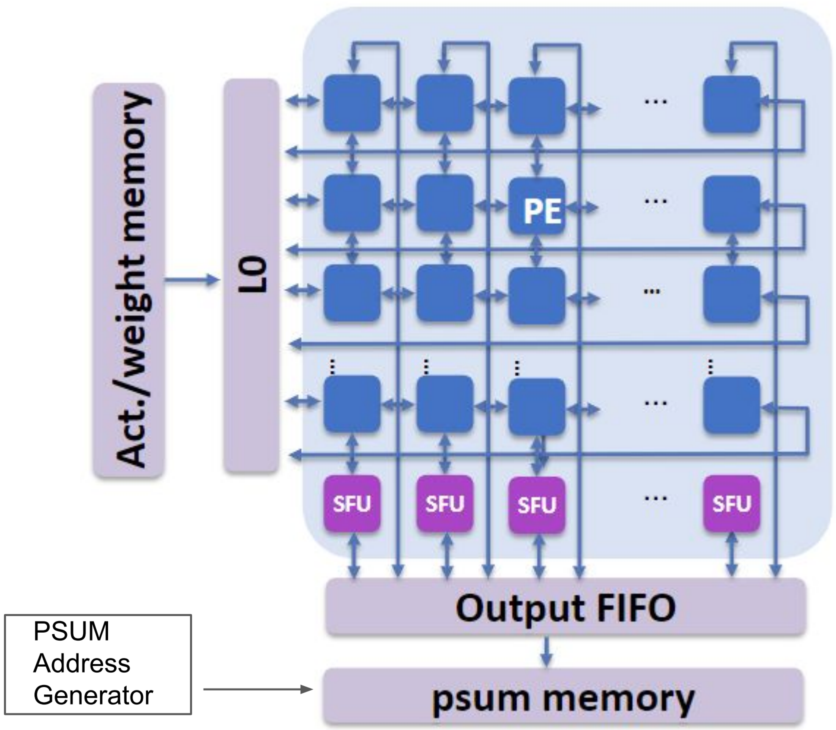
2. Doing in-place accumulation for Psum in SFU

3. Designed Controller for the whole design

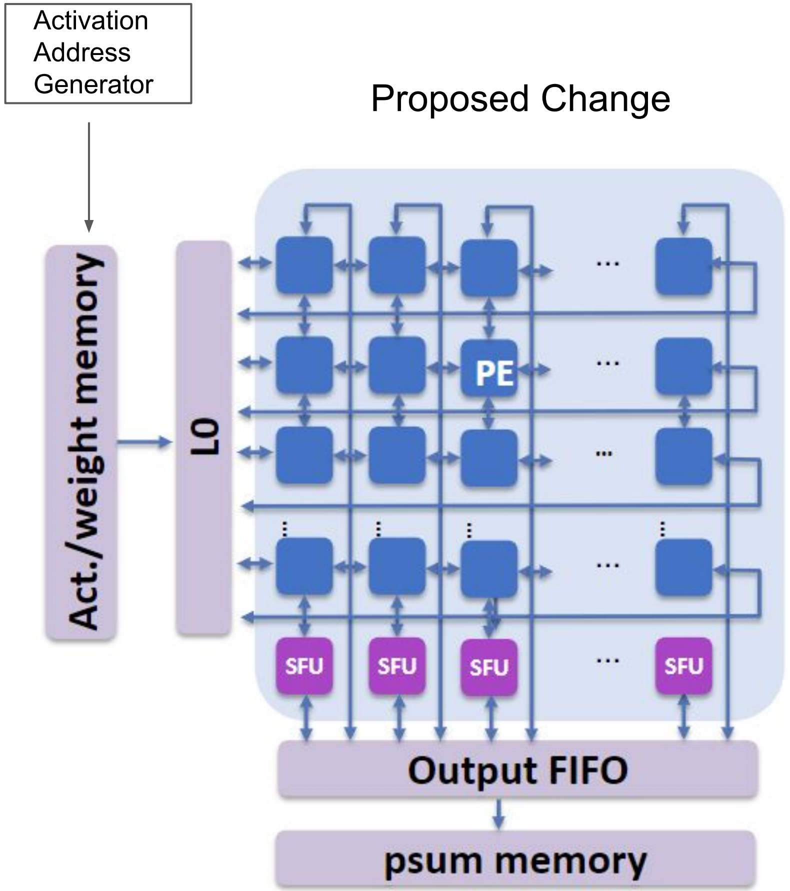
Thank You

Hardware Consideration

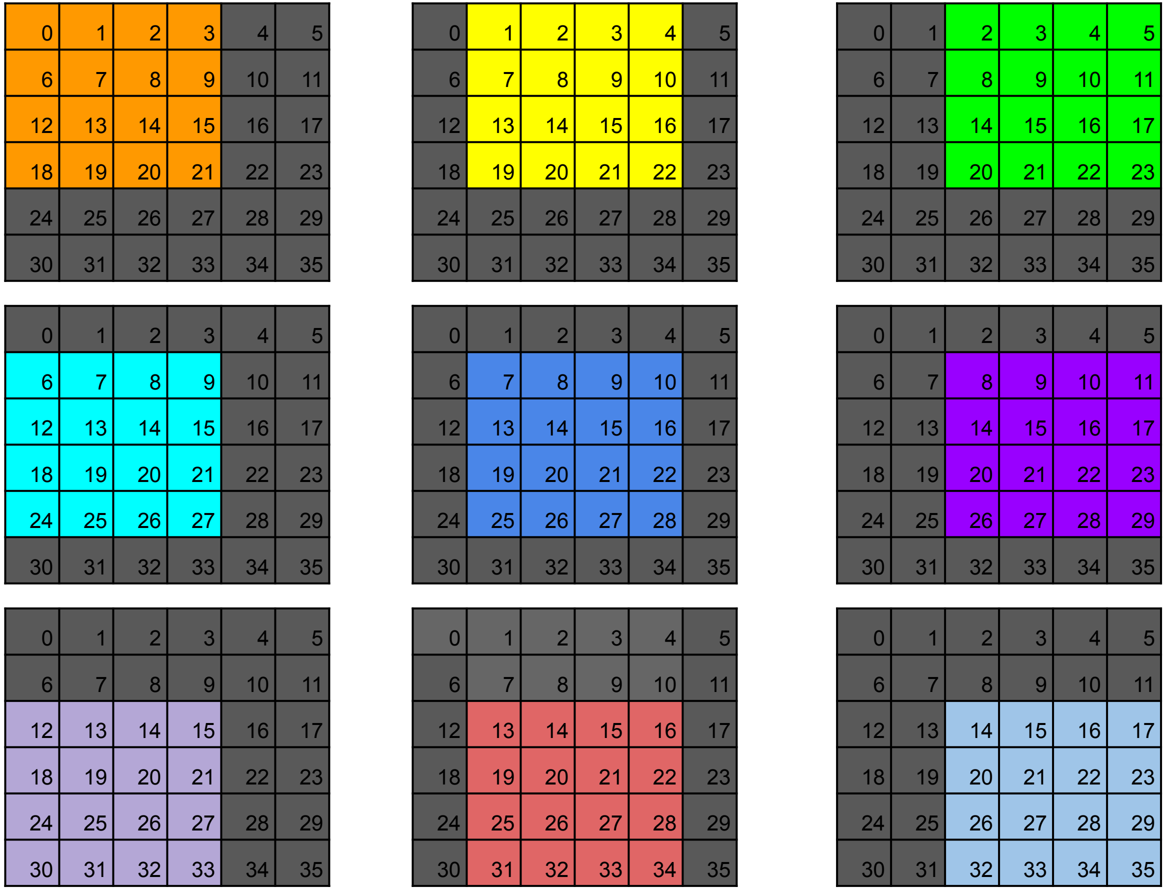
Baseline Design



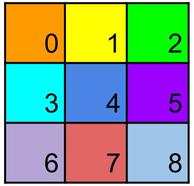
Proposed Change



Modifying the Input features sent to L0



Conv 3x3 Kernel




```
for (kij=0 ; kij<9; kij++)
    for (nij=0 ; nij <16 ; nij++)
        nij_prime = int(nij/4)*6 + nij%4;
        kij_prime = int(kij/3)*6 + kij%3;
        data_L0 = x_in[ kij_prime + nij_prime];
```

```
for (kij=0 ; kij<9; kij++)
    for (nij=0 ; nij <16 ; nij++)
        nij_prime = int(nij/4)*6 + nij%4;
        kij_prime = int(kij/3)*6 + kij%3;
        data_L0 = x_in[ kij_prime + nij_prime];
```

Benefits of Idea

- Cycles for loading Activations in L0 = 9(Kij) *16(Nij) = **144 cycles** vs 9(Kij)*36(Nij in Reference) = **324 cycles**
- Cycles for computing Psum for Kij=0 = [16 (Nij) + 8 (Last row staggered offset) + 8 (reaching last PE)] = 32 cycles

  Cycles for computing all Psums = 32*9(Kij) = **288 cycles** vs 9(Kij)*[36(Nij Reference) +8 +8] = **468 cycles**
- Cycles Saved in Computation = 792 - 432 = **360 cycles**

- Cycles for loading weights in L0 = 9(Kij) * 8 (Nij_{out}) = **72 cycles**
- Cycles for loading weights in Systolic Array = 9(Kij) * [8 (Last Row Staggered offset) + 8 (Out Ch) * 2(Cycles/wt loading)] = **216 cycles**
- **Percentage of saved cycles in Computation** = 360/1080 = **33%**

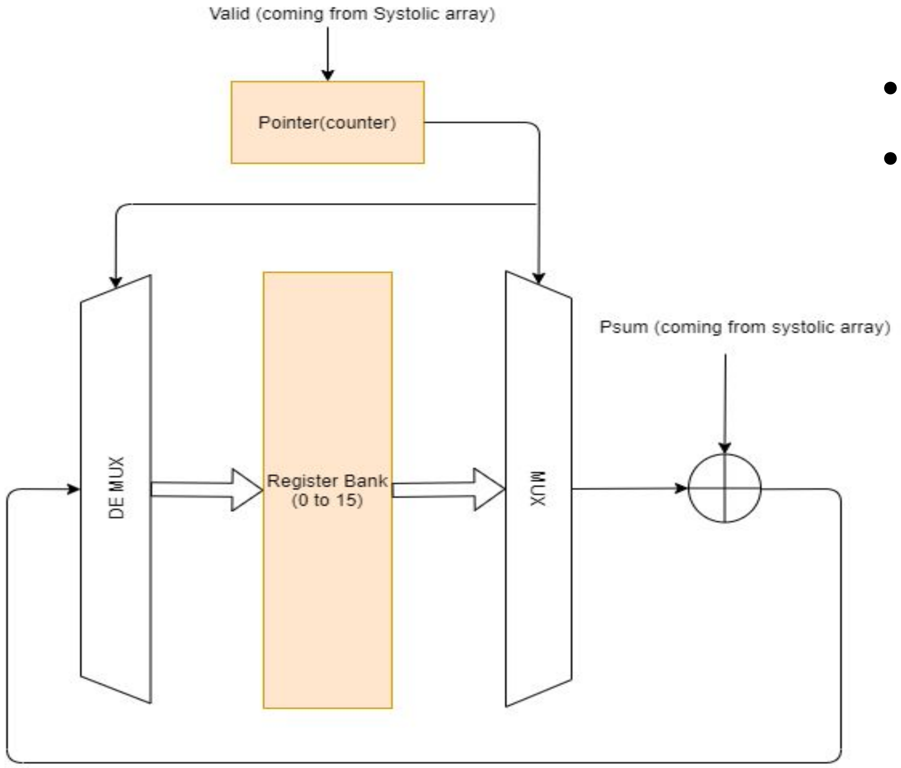
Benefits of Idea

Assuming 16 bit * 8(Out Ch) = 128 bits can be written in the SRAM in one clock cycle.

Number of such 128 bit transactions = 36 (In Ch) * 9 (Kij) = 324 cycles

With In-place Psum accumulation, **324 transactions** of 128 bits to the expensive SRAM were saved.

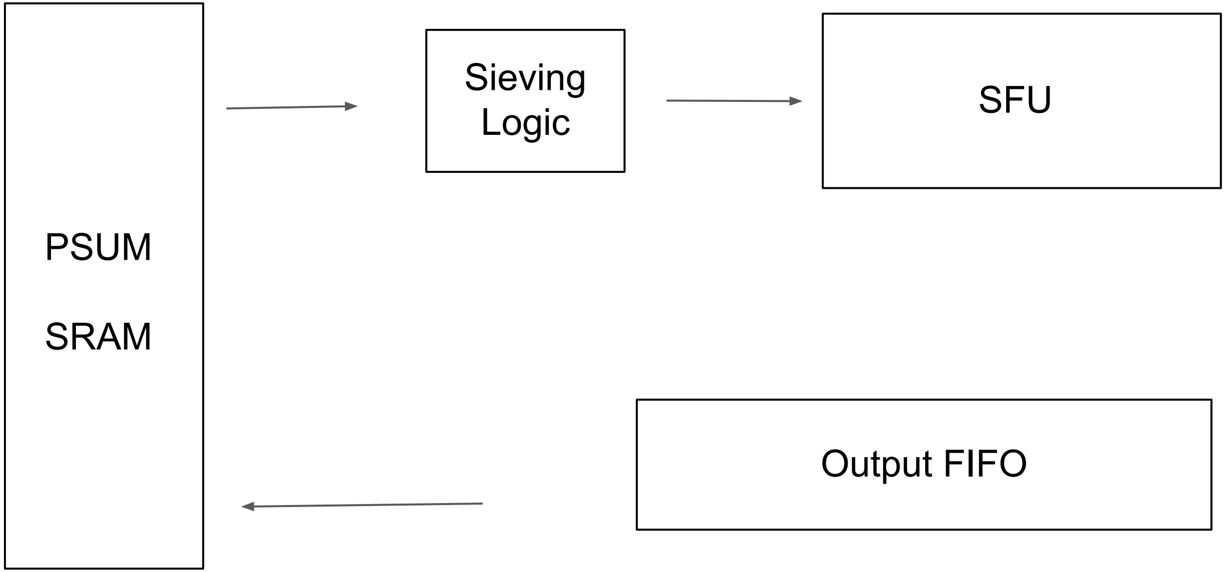
Modified SFU



- This is the modified version of the SFU that will directly compute the final output without the need to store it in scratch pad memory.
- We have removed the need for scratch pad memory (16x16x9) and O-FIFO (16x8 minimum).
- Instead, we have added registers (16x16) and a counter (4bits) to achieve the output calculation.

Hardware Consideration

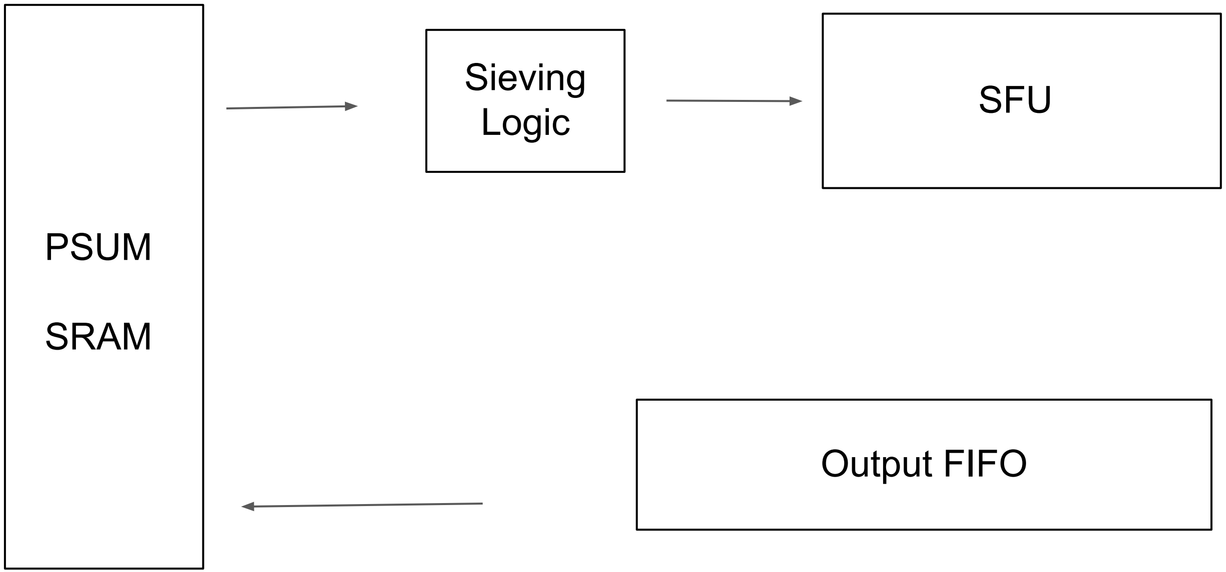
Placement of Sieving Logic (1)



Hardware Consideration

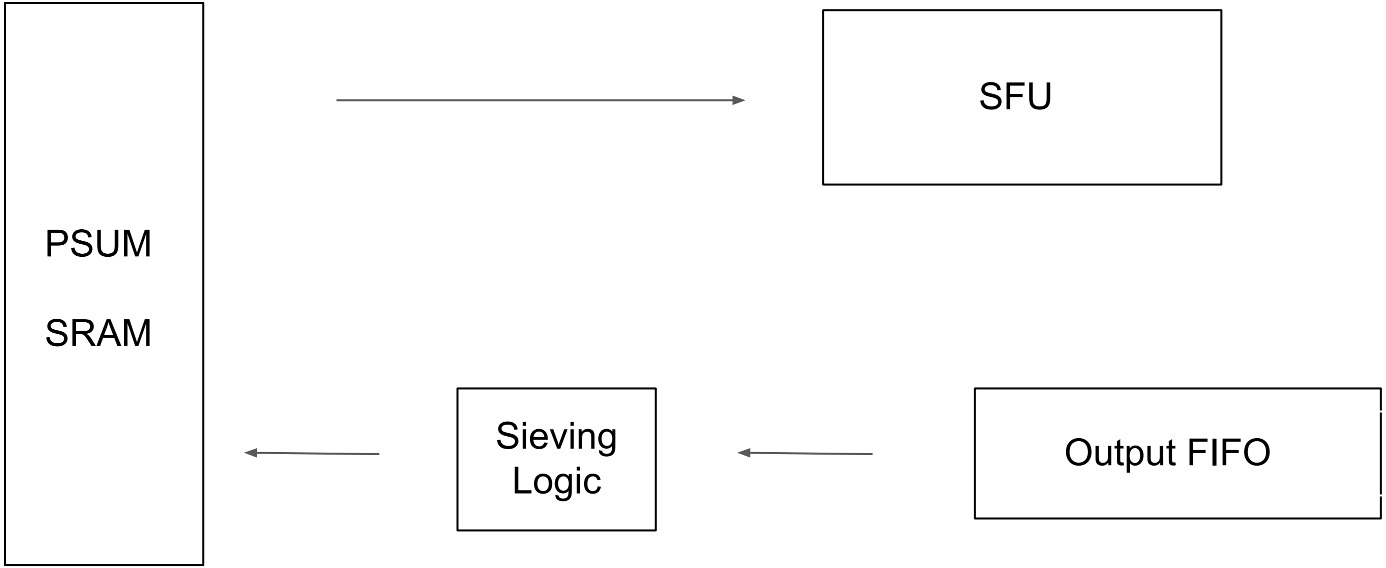
Can we do better?

Placement of Sieving Logic (1)



Hardware Consideration

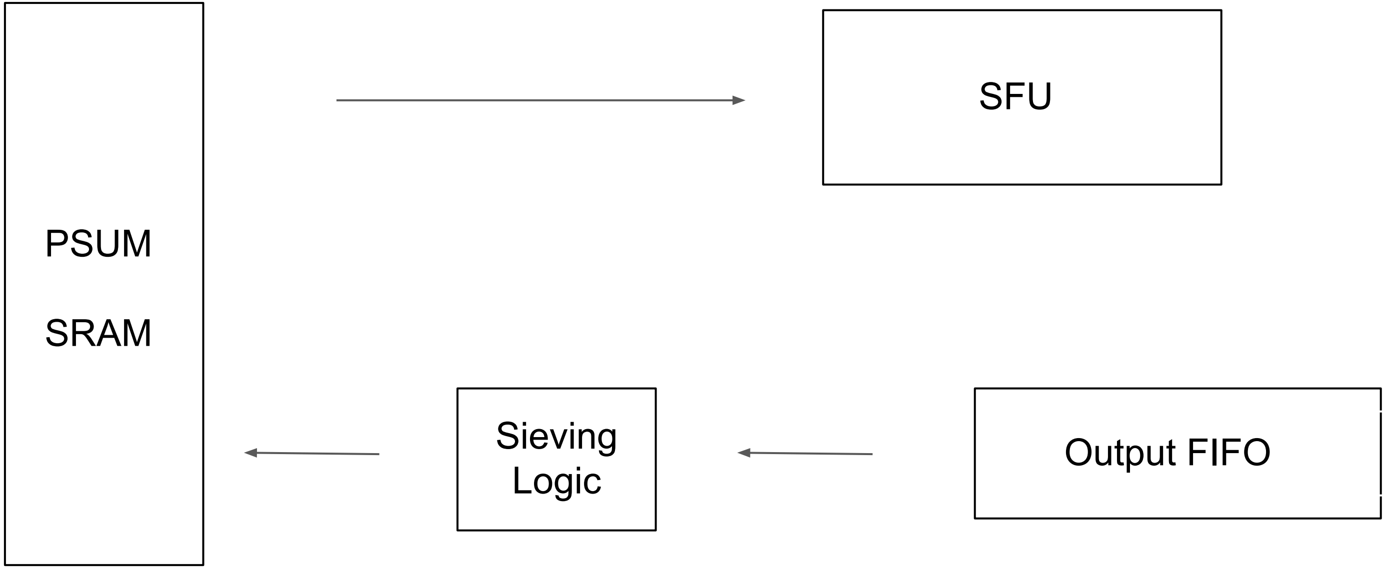
Placement of Sieving Logic (2)



Hardware Consideration

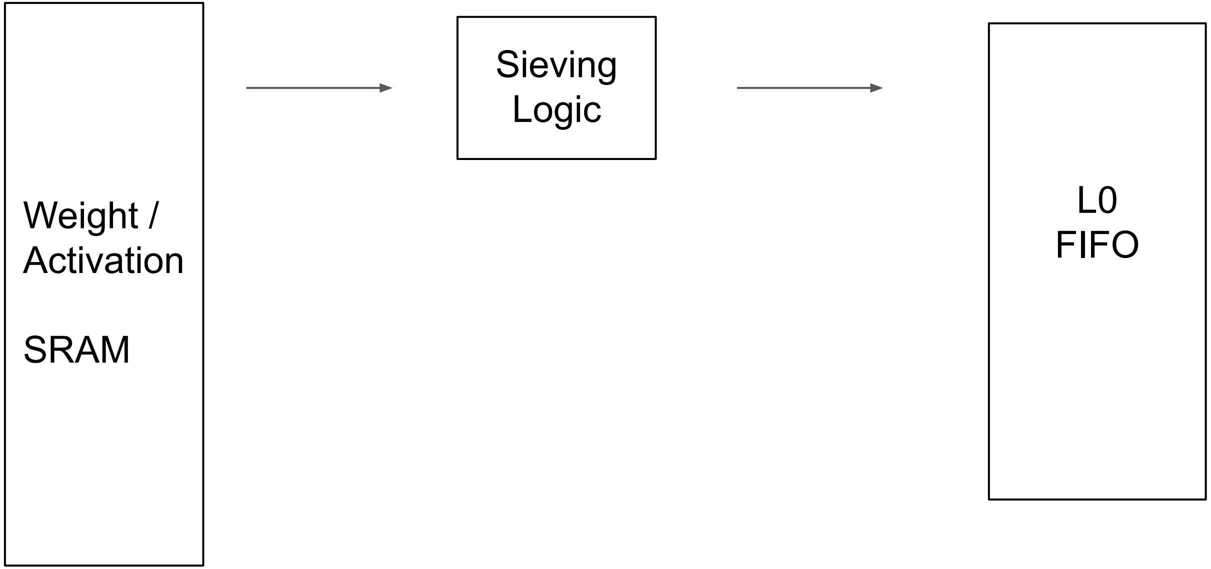
Can we do even better?

Placement of Sieving Logic (2)

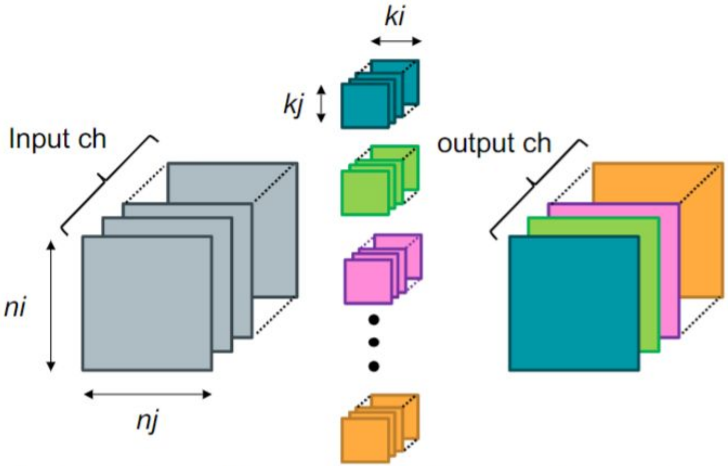


Hardware Consideration

Placement of Sieving Logic (3) : Proposed Design

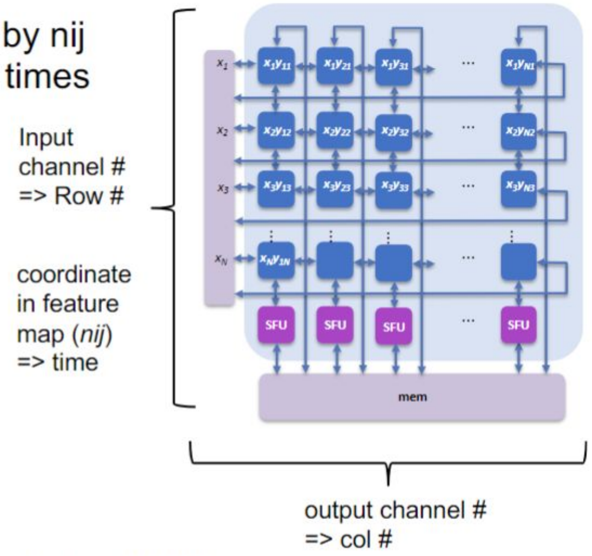


Algorithm to Hardware Mapping (Conv to 2D array)



Assumption: 3X3 kernel, 16X16 input feature map
64 in / out channels

Weight is reused by nij
(49 – thousands) times



Matrix multiplication

for kij = 0:8 (time, renew all the weights in registers)
  for out_ch = 0:63 (col #)
    for in_ch = 0:63 (row #)
      for nij = 0:255 (time for horizontal input)
        psum(out_ch, kij, nij) += w(out_ch, in_ch, kij) * x(in_ch, nij)

Accumulation (SFU)

for nij = 0:255 (output index)
  for kij = 0:8 (time)
    output (out_ch, nij) += psum(out_ch, kij, nij')

- Note nij' = f(nij, kij) is shifted index of nij for conv.
- Matmult and acc can be processed simultaneously.

Psum indexing : After passing the inputs into Systolic array

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59
60	61	62	63	64	65
66	67	68	69	70	71

72	73	74	75	76	77
78	79	80	81	82	83
84	85	86	87	88	89
90	91	92	93	94	95
96	97	98	99	100	101
102	103	104	105	106	107

108	109	110	111	112	113
114	115	116	117	118	119
120	121	122	123	124	125
126	127	128	129	130	131
132	133	134	135	136	137
138	139	140	141	142	143

144	145	146	147	148	149
150	151	152	153	154	155
156	157	158	159	160	161
162	163	164	165	166	167
168	169	170	171	172	173
174	175	176	177	178	179

180	181	182	183	184	185
186	187	188	189	190	191
192	193	194	195	196	197
198	199	200	201	202	203
204	205	206	207	208	209
210	211	212	213	214	215

216	217	218	219	220	221
222	223	224	225	226	227
228	229	230	231	232	233
234	235	236	237	238	239
240	241	242	243	244	245
246	247	248	249	250	251

252	253	254	255	256	257
258	259	260	261	262	263
264	265	266	267	268	269
270	271	272	273	274	275
276	277	278	279	280	281
282	283	284	285	286	287

288	289	290	291	292	293
294	295	296	297	298	299
300	301	302	303	304	305
306	307	308	309	310	311
312	313	314	315	316	317
318	319	320	321	322	323

Conv 3x3 Kernel

0	1	2
3	4	5
6	7	8

```
for (nij=0 ; nij<16; nij++)
    output[nij]=0
    for (kij=0 ; kij <9 ; kij++)
        nij_prime = int(nij/4)*6 + nij%4;
        kij_prime = int(kij/3)*114 + kij%3 * 37;
        output[nij] += psum[ kij_prime + nij_prime];
```


```
for (nij=0 ; nij<16; nij++)
    output[nij]=0
    for (kij=0 ; kij <9 ; kij++)
        nij_prime = int(nij/4)*6 + nij%4;
        kij_prime = int(kij/3)*114 + kij%3 * 37;
        output[nij] += psum[ kij_prime + nij_prime];
```

Sieve (ie nij_prime and kij_prime)

```
for (nij=0 ; nij<16; nij++)
    output[nij]=0
    for (kij=0 ; kij <9 ; kij++)
        nij_prime = int(nij/4)*6 + nij%4;
        kij_prime = int(kij/3)*114 + kij%3 * 37;
        output[nij] += psum[ kij_prime + nij_prime];
```

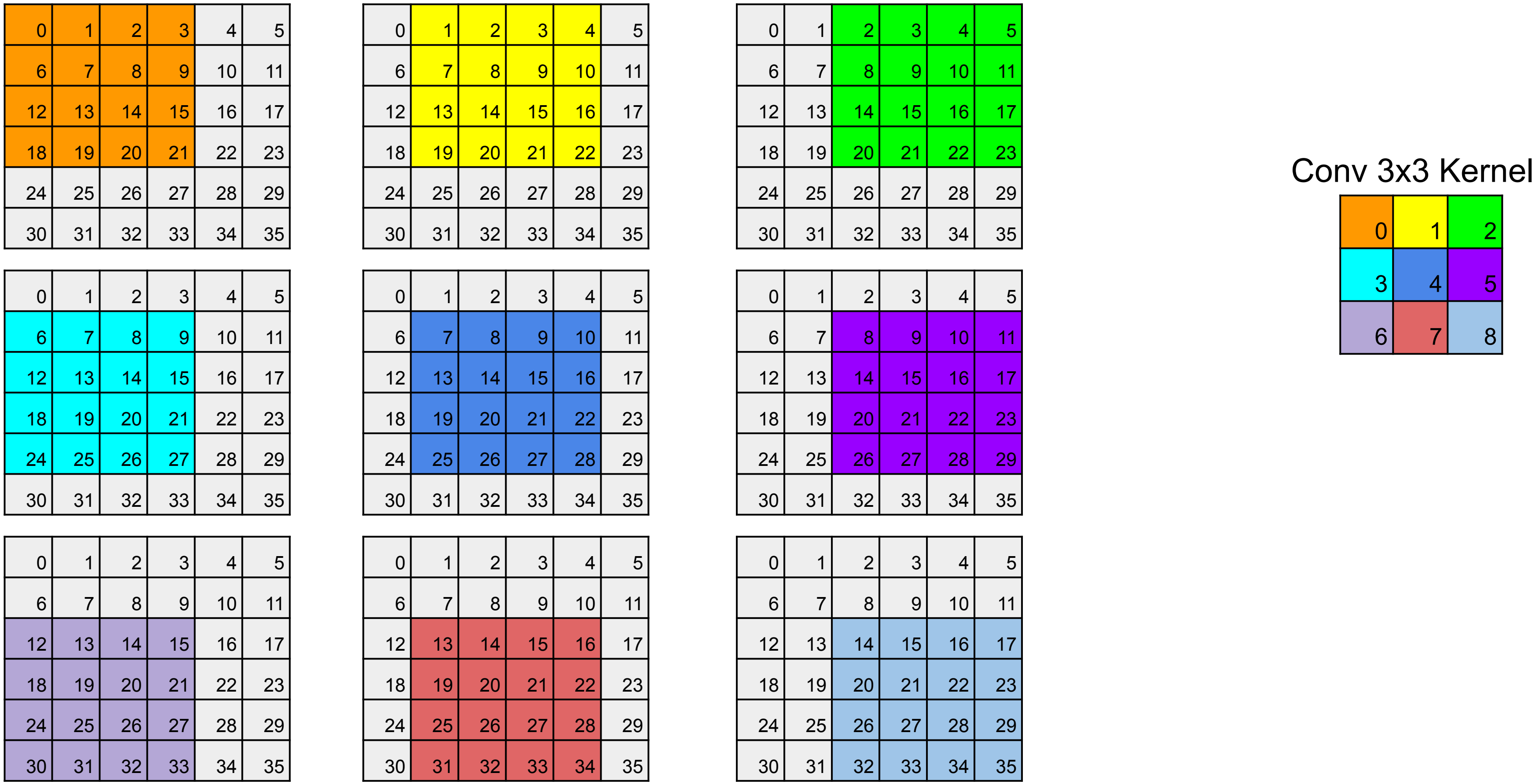

```
for (nij=0 ; nij<16; nij++)
    output[nij]=0
    for (kij=0 ; kij <9 ; kij++)
        nij_prime = int(nij/4)*6 + nij%4;
        kij_prime = int(kij/3)*114 + kij%3 * 37;
        output[nij] += psum[ kij_prime + nij_prime];
```

- Un-needed psums computed => (n+2)*(n+2) - n^2 = 4n+4
- The extra computations will take energy, storage and increase latency.

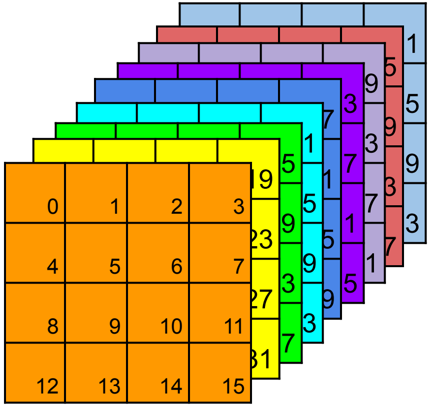
Can we do something about this?

How about we shift the **Sieve** to the activation input fetching stage??

Recap: Input features utilisation mapped to Kernel



Modified Store for Psums and SFU



- Utilizing the natural alignment of the psums, we can add them up in place to produce the output.
- For this we will need to have a running counter that will have to keep track of the nij index.
- We can either use registers or a dual port SRAM for storing the running sum. We implemented using register for proof of concept, but an implementation with SRAM will be more scalable with increasing input dimension size. This logic will replace the O-FIFO

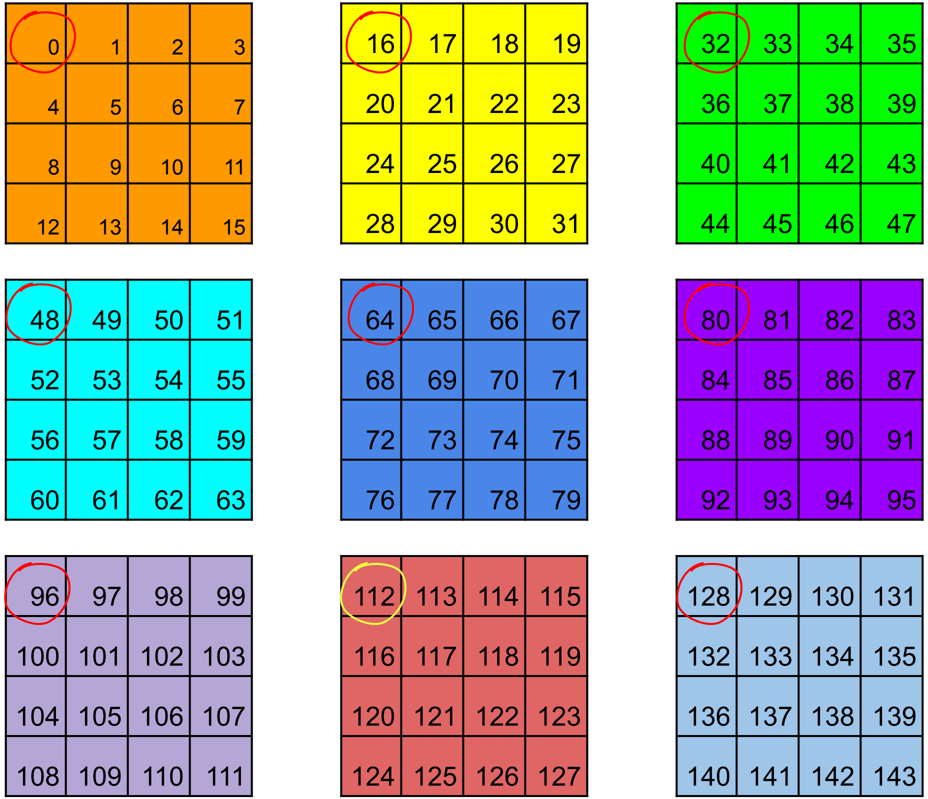
```
// reset to zero
for (nij=0 ; nij<16; nij++)
    output[ : ]=0

//SFU Computation
for (kij=0 ; kij <9 ; kij++)
    for (nij=0 ; nij<16; nij++)
        output[nij] += psum[ kij*16 + nij];
```


```
// reset to zero
for (nij=0 ; nij<16; nij++)
    output[ : ]=0

//SFU Computation
for (kij=0 ; kij <9 ; kij++)
    for (nij=0 ; nij<16; nij++)
        output[nij] += psum[ kij*16 + nij];
```

Modified Psums and their indexing



- Symmetric relative addressing between the psums for output computation, so simple logic to call the psum to the SFU
- Lesser memory used to store the psums.
- Opportunity to make architectural changes to the SFU unit.

```
for (nij=0 ; nij<16; nij++)
    output[nij]=0
    for (kij=0 ; kij <9 ; kij++)
        output[nij] += psum[ kij*16 + nij];
```


```
for (nij=0 ; nij<16; nij++)
    output[nij]=0
    for (kij=0 ; kij <9 ; kij++)
        output[nij] += psum[ kij*16 + nij];
```