

به نام خدا

اعضای گروه :آیسا میاهی نیا 99522149 ، گلبرگ سپهرآرا 99521334

عنوان پروژه : پیاده سازی یک تسریع کننده سخت افزاری برای شبکه های عصبی عمیق از نوع CNN برای پردازش تصویر

با توجه به مواردی که داخل داک پروژه ذکر شده است ، ورودی سخت افزار ما یک عکس است که بیت های آن به صورت آرایه ای از اعداد حقیقی در آمده است ، پس ورودی سخت افزار آرایه ای از اعداد حقیقی میباشد و خروجی آن نیز آرایه ای از اعداد حقیقی میباشد که در واقع پیکسل های فیلتر شده عکس ورودی هستند. در تکه کد زیر ورودی و خروجی سخت افزار به زبان vhdl مشخص شده است.

```
32 --use UNISIM.vcomponents.all;  
33  
34 entity main is  
35     generic( N : integer := 4);  
36     port (picture_in : in pic_type ;  
37           pic_out : out result_type);  
38 end main;  
39
```

شکل کلی سخت افزار :

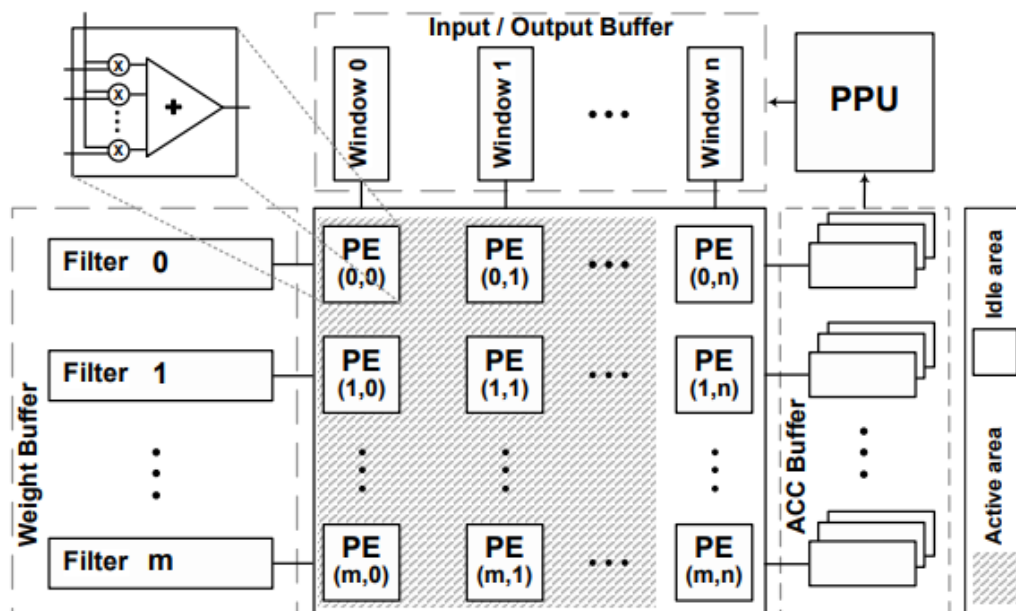


Fig. 1: A tile of basic array-based architecture.

ابتدا به پیاده سازی جزئی ترین بخش این معماری ، یعنی ساخت موجودیت PE میپردازیم :

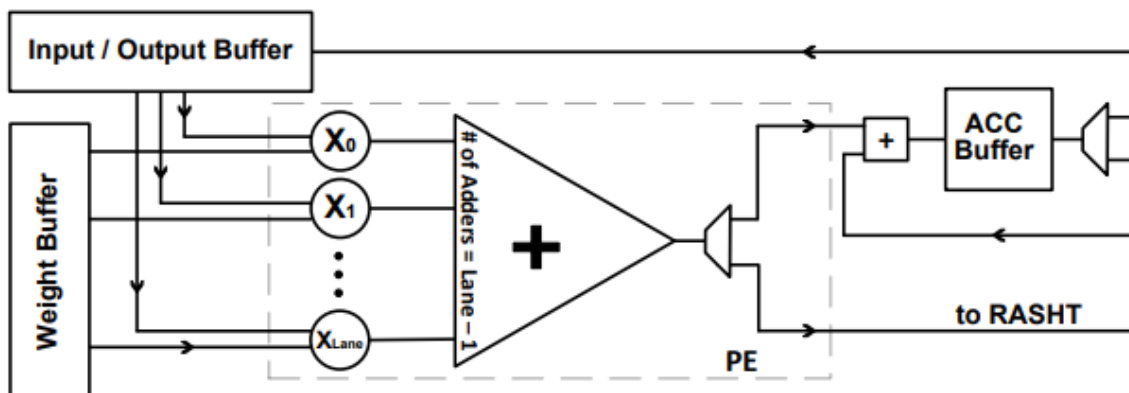


Fig. 3: Proposed PE.

Algorithm 1 Algorithm of convolutional layer execution.

```
procedure CONVOLUTION( $N, F, C, K_x, K_y$ )  
     $\triangleright N$ : the number of input windows  
     $\triangleright F$ : the number of weight filters  
     $\triangleright C$ : the number of channels  
     $\triangleright K_x$  and  $K_y$ : the dimension sizes of channel  
    for  $w = 0; w < N; w += COLUMNS$  do  
        for  $f = 0; f < F; f += ROWS$  do  
            for  $c = 0; c < (C \times K_x \times K_y); c += LANES$  do  
                return  $Output[f][O_x][O_y] +=$   
                     $Input[c][n_x][n_y] \times Weight[f][c][K_x][K_y]$   
            end for  
        end for  
    end for  
end procedure
```

```
36  
37 entity PE is  
38     Port ( weight_filter : in  three_three;  
39           window : in  three_three;  
40           output : out real);  
41 end PE;  
42  
43 architecture Behavioral of PE is  
44  
45 begin  
46     process(weight_filter , window)  
47         variable tmp : real;  
48         begin  
49             for i in 0 to 2 loop  
50                 for j in 0 to 2 loop  
51                     tmp := tmp + (weight_filter(i , j) * window(i , j)) ;  
52                 end loop;  
53             end loop;  
54             if integer(tmp) < 0 then  
55                 tmp:= real(0);  
56             end if;  
57             output<= tmp;  
58         end process;  
59 end Behavioral;  
60
```

همانطور که از کد مشخص است ورودی PE یک فیلتر 3 در 3 و پنجره (window) ای از تصویرمان هست و خروجی آن حاصل انجام عملیات کانولوشن روی پیکسل های متناظر فیلتر و پنجره است که نتیجه یک عدد حقیقی

میباشد.

پس از PE ، داخل main مان باید تصمیم بگیریم که چگونه عکس ورودی را پنجره پنجر و همچنین فیلترهایی که قرار است روی عکس اعمال شود را تولید کنیم . برای این منظور دوتابع GetWindow و generate_random_filter داخل پکیج Mypackage نوشته شده اند.

```
PACKAGE My_Package IS
  type pic_type is array (0 to 127 , 0 to 127) of real;
  type three_three is array (0 to 2 , 0 to 2) of real;
  type result_type is array (0 to 125,0 to 125) of real;
  type array_of_real is array (0 to 3) of real; -- N = 4
  type array_of_filters is array(INTEGER RANGE <>) of three_three;
END My_Package;
```

```
31 END my_package,
32
33 package body My_Package is
34   function GetWindow (N: integer; pic_in: pic_type) return array_of_filters is
35     variable window : three_three ;
36     variable result : array_of_filters(N-1 downto 0);
37   begin
38     for m in 0 to N-1 loop
39       for i in 0 to 125 loop
40         for j in 0 to 125 loop
41           for x in 0 to 2 loop
42             for y in 0 to 2 loop
43               window(x , y) := pic_in(i+x , j+y);
44             end loop;
45           end loop;
46           result(m) := window;
47         end loop;
48       end loop;
49     end loop;
50     return result ;
51   end function ;
52   function generate_random_filter(N : integer) return array_of_filters is
53     variable r : real;
```

```

50     return result ;
51 end function ;
52 function generate_random_filter(N : integer) return array_of_filters is
53     variable r : real;
54     variable o : three_three ;
55     variable intx : integer ;
56     variable seed1 : integer := 0;
57     variable seed2 : integer := 100;
58     variable result : array_of_filters(N-1 downto 0);
59     variable tmp : real;
60     begin
61         for j in 0 to N-1 loop
62             for i in 0 to 2 loop
63                 uniform(seed1 , seed2, r );
64                 intx := integer(floor(r* real(2**3)));
65                 tmp := real(intx);
66                 for j in 0 to 2 loop
67                     o(i , j) := tmp;
68                 end loop;
69             end loop ;
70             result(j) := o;
71         end loop;
72     return result;
73 end function;
74 end My_Package;
--

```

پس از تعریف تابع های نام برده شده ، داخل فایل main مان ، تصویر ورودی را پنجره پنجره کرده و روی هر پنجره ، فیلتر های تولید شده را اثر میدهیم ، که این عمل به وسیله ی generate کردن کامپوننت PE انجام میشود.

در فایل main :

```

81     end function;
82 component PE is
83     Port ( weight_filter : in  three_three;
84           window : in  three_three;
85           output : out real);
86 end component;
87 component Rasht is
88     port (pes: in array_of_real ;
89           result_out : out real);
90 end component;
91 signal pe_out : array_of_pe;
92 signal filters_array : array_of_filters(N-1 downto 0);
93 signal window_array : array_of_filters(N-1 downto 0);
94 signal rasht_out_array : array_of_real;
95 begin
96     filters_array <= generate_random_filter(N);
97     window_array <= GetWindow(N ,picture_in);
98     col_generate : for i in 0 to N-1 generate
99         row_generate : for j in 0 to N-1 generate --architecture due to number of filters
100             PE1 : PE port map(filters_array(j) , window_array(i), pe_out(i)(j));
101         end generate;
--

```

حال باید خروجی های PE های هر ستون را به کامپوننت Rasht بدهیم ، تا به خروجی PE ها را با هم جمع کند . معماری این موجودیت طوری طراحی شده که ورودی ها را به صورت Recursive و همین طور موازی جمع میکند ، در نتیجه منجر به افزایش سرعت میشود .

```
35
36 entity Rasht is
37     port (pes: in array_of_real;
38           result_out : out real);
39 end Rasht;
40
41
42 architecture Behavioral of Rasht is
43     function tree_adder (pes : array_of_real ; start_idx , end_idx : integer) return real is
44         variable left : real;
45         variable right : real;
46         variable mid : integer;
47         variable tmp1 : real;
48         variable tmp2 : real;
49     begin
50         if end_idx - start_idx + 1 = 2 then
51             tmp1:= pes(start_idx);
52             tmp2 := pes(end_idx);
53             return tmp1 + tmp2;
54         else
55             mid := (start_idx + end_idx)/2;
56             left:= tree_adder(pes , start_idx , mid );
57             right := tree_adder(pes , mid + 1 , end_idx);
58             return left + right;
59         end if;
60     end tree_adder;
61 begin
62
63     result_out <= tree_adder(pes , 0 , pes'length -1);
64 end Behavioral;
65
66
```

حال در فایل main ، کامپونت این موجودیت را ایجاد کرده و خروجی PE های هر ستون را به یک Rasht میدهیم. همان طور که در خط 102 تکه کد زیر قابل مشاهده هست ، خروجی هر رشت را داخل آرایه ای به نام rasht_out_array ذخیره کرده و در نهایت در پیکسل های متناظر تصویر خروجی قرار میدهیم. این عملیات در خطوط 106 تا 110 کد زیر انجام شده است.

```

93 signal window_array : array_of_filters(N-1 downto 0);
94 signal rasht_out_array : array_of_real;
95 begin
96   filters_array <= generate_random_filter(N);
97   window_array <= GetWindow(N , picture_in);
98   col_generate : for i in 0 to N-1 generate
99     row_generate : for j in 0 to N-1 generate --architecture due to number of filters
100     PE1 : PE port map(filters_array(j) , window_array(i), pe_out(i)(j));
101   end generate;
102   Rasht1 : Rasht port map (pe_out(i) , rasht_out_array(i));
103 end generate;
104 process is
105   begin
106     for i in 0 to 125 loop
107       for j in 0 to 125 loop
108         pic_out(i , j) <= rasht_out_array( 125 * i + j);
109       end loop;
110     end loop;
111     wait;
112   end process;
113 end Behavioral;
114

```

و این گونه است که سخت افزار خود را طراحی میکنیم .

حال برای تست کردن کارایی این کد ، ابتدا یک فایل تست برای فایل main مان میسازیم . ورودی کامپوننتمان باید آرایه ای از اعداد حقیقی باشد ، پس عکسی را که میخواهیم فیلتر کنیم، ابتدا با کد پایتون به پیکسل های آن که آرایه ای از اعداد حقیقی هست تبدیل کرده و خروجی آن را به فایل تست مان میدهیم .

کد پایتون :

```

C: > classes > CAD > Project > Project (3) > make_input.py
1  from PIL import Image
2  import numpy as np
3  import cv2
4  image = cv2.imread('cat.jpg' , 0)
5  image = cv2.resize(image , (128,128))
6
7  np.savetxt('picture.txt', image , delimiter=',')

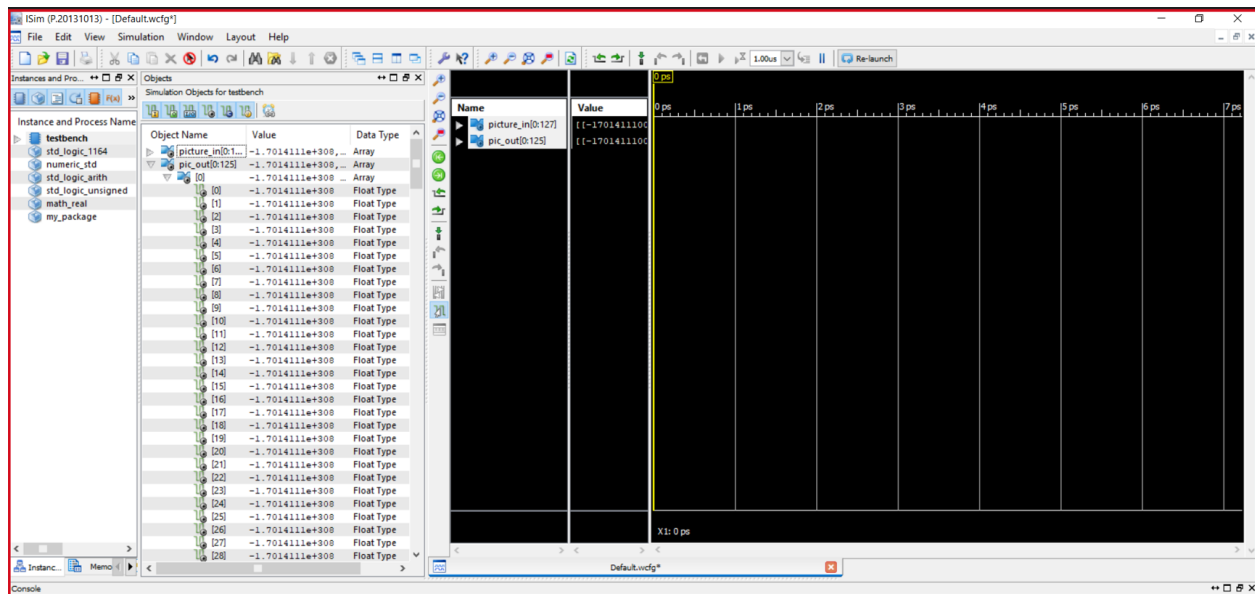
```

عکس مورد نظر و خروجی کد نیز داخل پوشه پروژه قرار داده شده است .

فایل تست :

```
-- Testbench template
2
3
4 LIBRARY ieee;
5 USE ieee.std_logic_1164.ALL;
6 USE ieee.numeric_std.ALL;
7 use Work.My_Package.all;
8
9
10 ENTITY testbench IS
11 END testbench;
12
13 ARCHITECTURE behavior OF testbench IS
14
15 -- Component Declaration
16 COMPONENT main
17 port (picture_in : in pic_type ;
18       pic_out : out result_type);
19 END COMPONENT;
20
21 SIGNAL picture_in : pic_type;
22 SIGNAL pic_out : result_type;
23
24 BEGIN
25
26 -- Component Instantiation
27 uut: main PORT MAP(
28     picture_in => picture_in,
29     pic_out => pic_out
30 );
31
32 -- Test Bench Statements
33 tb : PROCESS
34 BEGIN
35     picture_in<=(1.7800000000000000e+02,1.7300000000000000e+02,1.5600000000000000e+02,1.3300000000000000e+02,9.1000000000000000e+01,6.5000000000000000e+01,8.2000000000000000e+01,1.7000000000000000e+02,1.8100000000000000e+02,1.5800000000000000e+02,1.2500000000000000e+02,9.3000000000000000e+01,6.1000000000000000e+01,8.0000000000000000e+01,1.1000000000000000e+02,1.7400000000000000e+02,1.5900000000000000e+02,1.2400000000000000e+02,8.9000000000000000e+01,6.0000000000000000e+01,7.9000000000000000e+01,1.1000000000000000e+02,1.8100000000000000e+02,1.7600000000000000e+02,1.5800000000000000e+02,1.2000000000000000e+02,9.3000000000000000e+01,6.1000000000000000e+01,8.4000000000000000e+01,1.1100000000000000e+02,1.8000000000000000e+02,1.7700000000000000e+02,1.4900000000000000e+02,1.2100000000000000e+02,8.9000000000000000e+01,5.8000000000000000e+01,5.2000000000000000e+01,1.1700000000000000e+02,1.7900000000000000e+02,1.7500000000000000e+02,1.5500000000000000e+02,1.1800000000000000e+02,9.6000000000000000e+01,5.9000000000000000e+01,8.4000000000000000e+01,1.1500000000000000e+02,1.8200000000000000e+02,1.6300000000000000e+02,1.5700000000000000e+02,1.1900000000000000e+02,8.7000000000000000e+01,5.8000000000000000e+01,8.4000000000000000e+01,1.1400000000000000e+02,1.7700000000000000e+02,1.7000000000000000e+02,1.5400000000000000e+02,1.1900000000000000e+02,8.9000000000000000e+01,5.6000000000000000e+01,8.3000000000000000e+01,1.1400000000000000e+02);
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
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```


خروجی تست :



پایان!