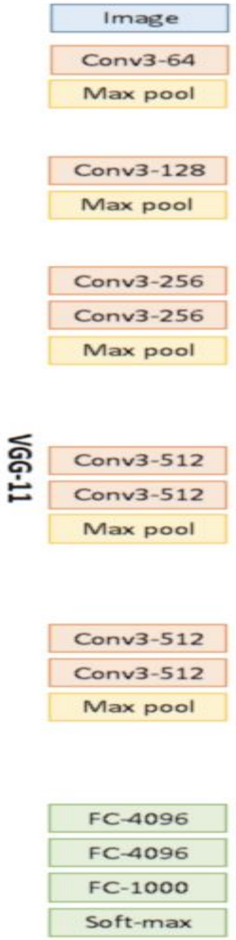
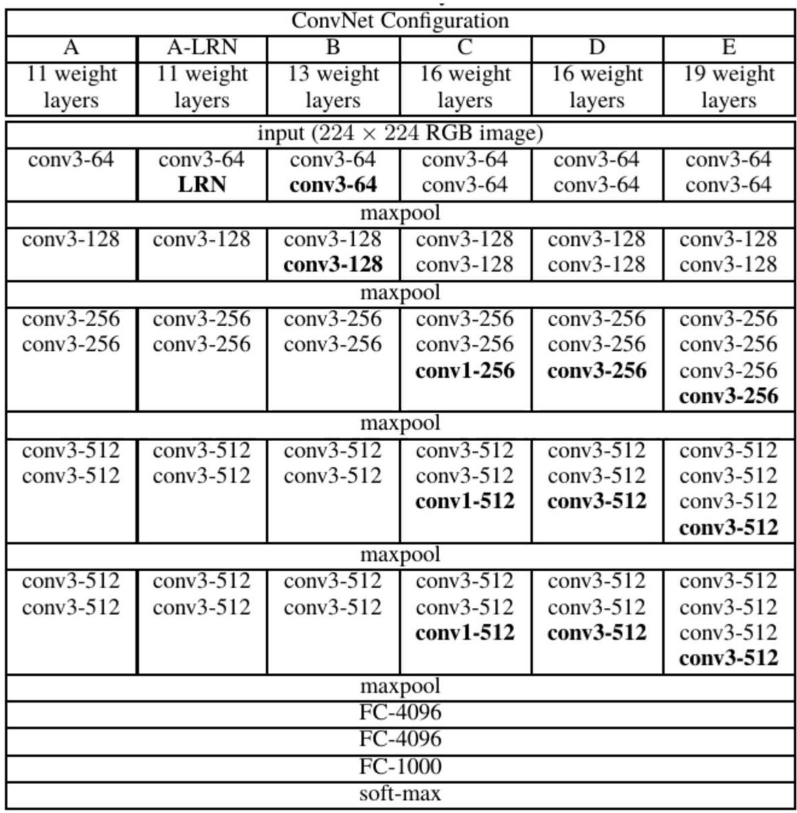
Convolutional Neural Network

Yanchen Liu yl4189
Minghui Zhao mz2866

Overview

Applying float point 16 modified VGG11 network on the de1-Soc.



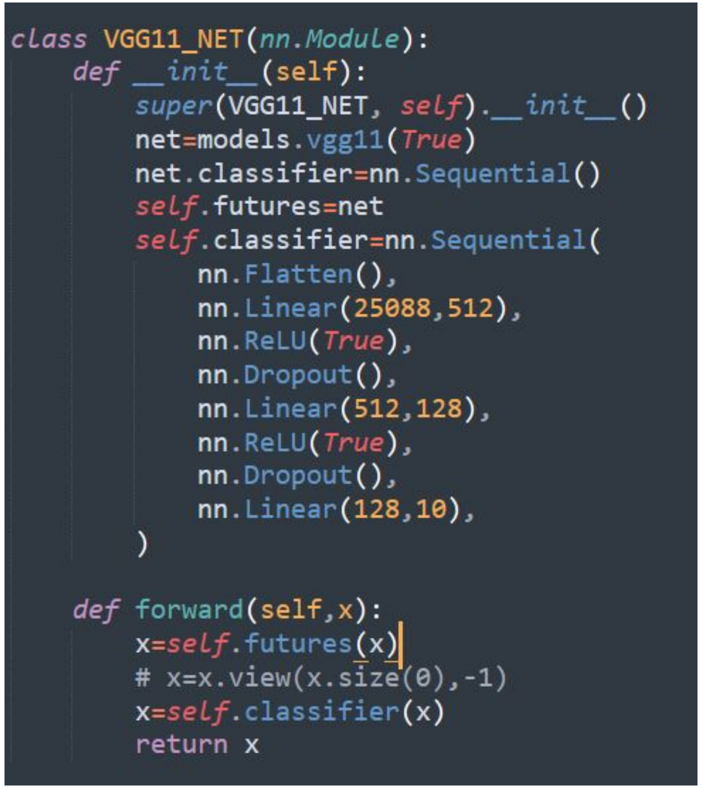


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Neural Network Architecture

VGG11 Network in pytorch.

When forwarding, conv2d, pooling and Relu are applied.

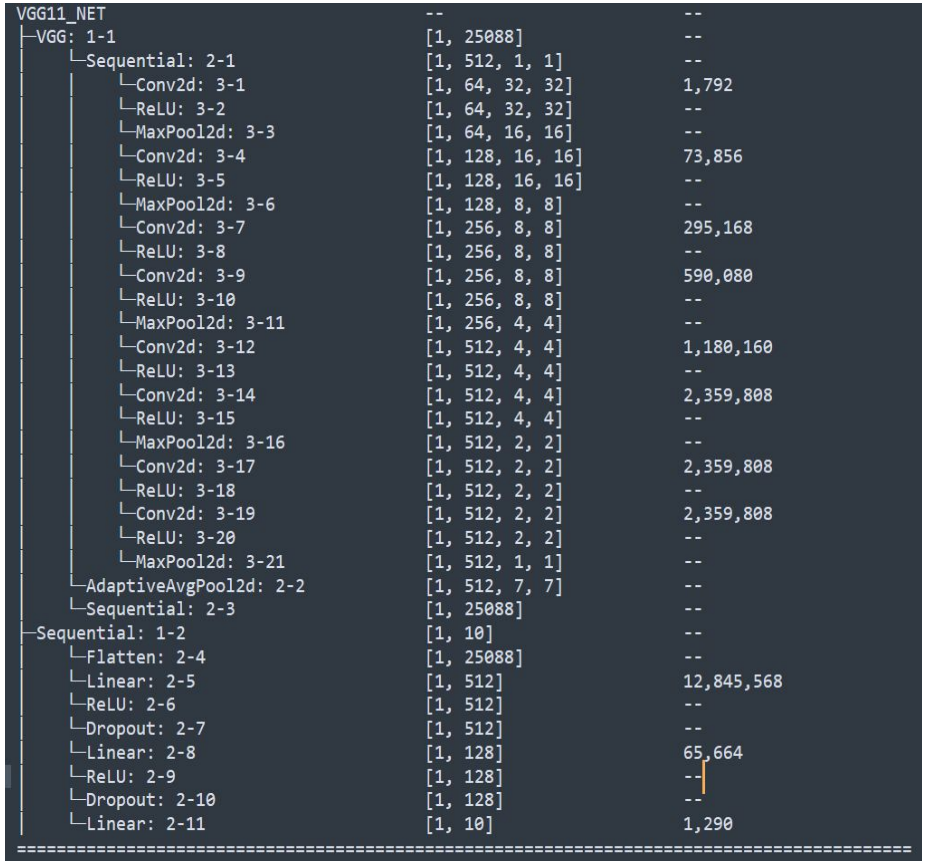


```
class VGG11_NET(nn.Module):
    def __init__(self):
        super(VGG11_NET, self).__init__()
        net=models.vgg11(True)
        net.classifier=nn.Sequential()
        self.futures=net
        self.classifier=nn.Sequential(
            nn.Flatten(),
            nn.Linear(25088,512),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(512,128),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(128,10),
        )

    def forward(self,x):
        x=self.futures(x)
        # x=x.view(x.size(0),-1)
        x=self.classifier(x)
        return x
```

Neural Network Architecture

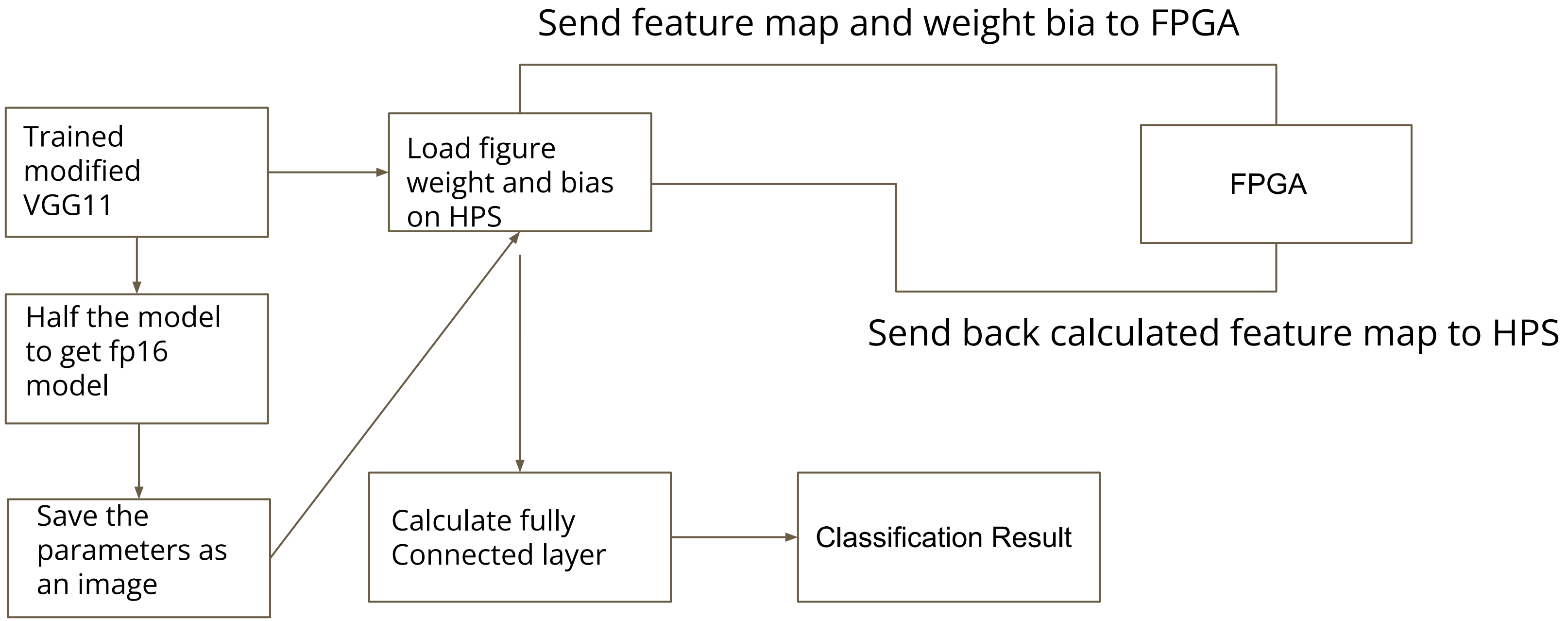
Architecture of the VGG11 provided by the pytorch official

Modify the classification class from 1000 to 10 and input image size from 224 by 224 to 32 by 32.



```
VGG11_NET                                         --                        --
├─VGG: 1-1                                        [1, 25088]                --
│    └─Sequential: 2-1                            [1, 512, 1, 1]            --
│    │    └─Conv2d: 3-1                           [1, 64, 32, 32]           1,792
│    │    └─ReLU: 3-2                             [1, 64, 32, 32]           --
│    │    └─MaxPool2d: 3-3                        [1, 64, 16, 16]           --
│    │    └─Conv2d: 3-4                           [1, 128, 16, 16]          73,856
│    │    └─ReLU: 3-5                             [1, 128, 16, 16]          --
│    │    └─MaxPool2d: 3-6                        [1, 128, 8, 8]            --
│    │    └─Conv2d: 3-7                           [1, 256, 8, 8]            295,168
│    │    └─ReLU: 3-8                             [1, 256, 8, 8]            --
│    │    └─Conv2d: 3-9                           [1, 256, 8, 8]            590,080
│    │    └─ReLU: 3-10                            [1, 256, 8, 8]            --
│    │    └─MaxPool2d: 3-11                       [1, 256, 4, 4]            --
│    │    └─Conv2d: 3-12                          [1, 512, 4, 4]            1,180,160
│    │    └─ReLU: 3-13                            [1, 512, 4, 4]            --
│    │    └─Conv2d: 3-14                          [1, 512, 4, 4]            2,359,808
│    │    └─ReLU: 3-15                            [1, 512, 4, 4]            --
│    │    └─MaxPool2d: 3-16                       [1, 512, 2, 2]            --
│    │    └─Conv2d: 3-17                          [1, 512, 2, 2]            2,359,808
│    │    └─ReLU: 3-18                            [1, 512, 2, 2]            --
│    │    └─Conv2d: 3-19                          [1, 512, 2, 2]            2,359,808
│    │    └─ReLU: 3-20                            [1, 512, 2, 2]            --
│    │    └─MaxPool2d: 3-21                       [1, 512, 1, 1]            --
│    └─AdaptiveAvgPool2d: 2-2                     [1, 512, 7, 7]            --
│    └─Sequential: 2-3                            [1, 25088]                --
├─Sequential: 1-2                                 [1, 10]                   --
│    └─Flatten: 2-4                               [1, 25088]                --
│    └─Linear: 2-5                                [1, 512]                  12,845,568
│    └─ReLU: 2-6                                  [1, 512]                  --
│    └─Dropout: 2-7                               [1, 512]                  --
│    └─Linear: 2-8                                [1, 128]                  65,664
│    └─ReLU: 2-9                                  [1, 128]                  --
│    └─Dropout: 2-10                              [1, 128]                  --
│    └─Linear: 2-11                               [1, 10]                   1,290
==========================================================================================
```

Software system

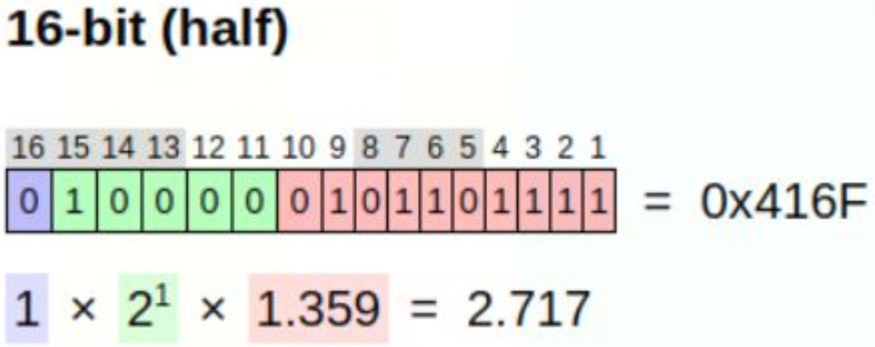


Send feature map and weight bia to FPGA
Trained modified VGG11
Load figure weight and bias on HPS
FPGA
Half the model to get fp16 model
Send back calculated feature map to HPS
Save the parameters as an image
Calculate fully Connected layer
Classification Result

Float point 16

Not using float point 32 due to the memory limitation.

Not using integer 8 due to the loss of the accuracy and dynamic bias.



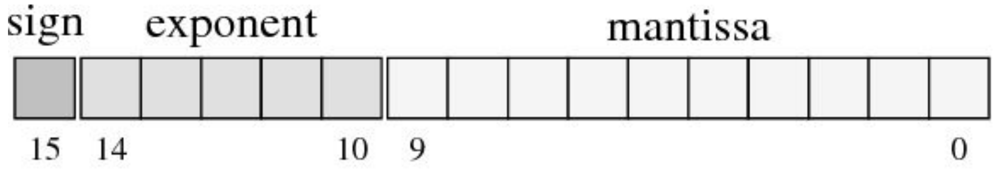
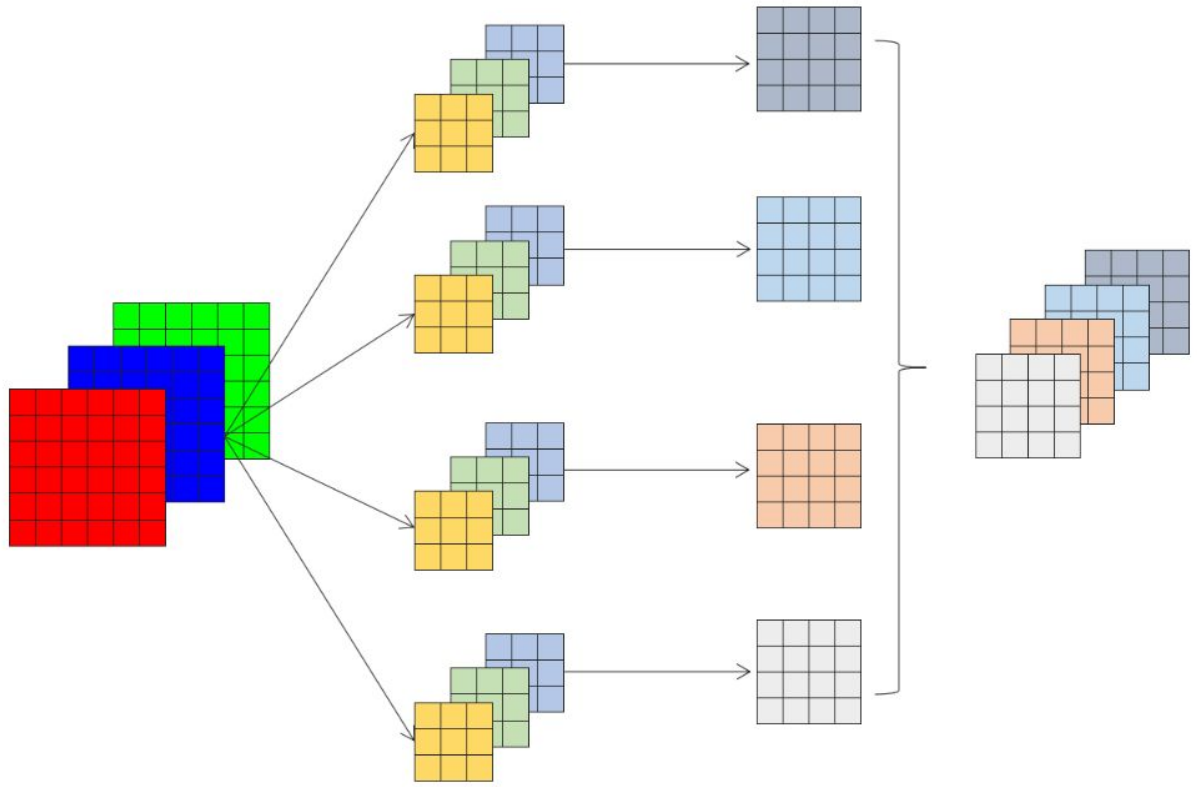


Fig. 1: IEEE 754 binary16 format

Conv2d: an example



An example of conv2d with in channel equal to 3 and out channel equal to 4, with a 3 by 3 kernel and no zero padding.

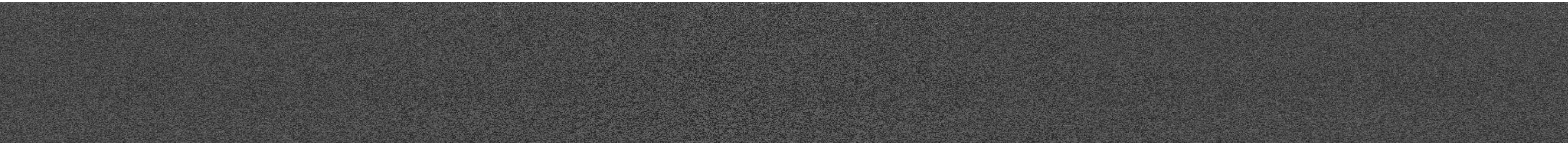
Operators in HPS: Maxpooling, Relu, adaptive avg pooling

Relu only need to check the first bit of a fp16 pixel.

All the max pooling 2d in VGG11 is 2 by 2 with a stride of 2. Since we are using float point 16 to represent the feature map and pooling is after Relu, the comparison could only the exponent part.

Adaptive average pooling is a pytorch unique method, in the structure provided by the official, is simply expand (1,1) to (7,7)

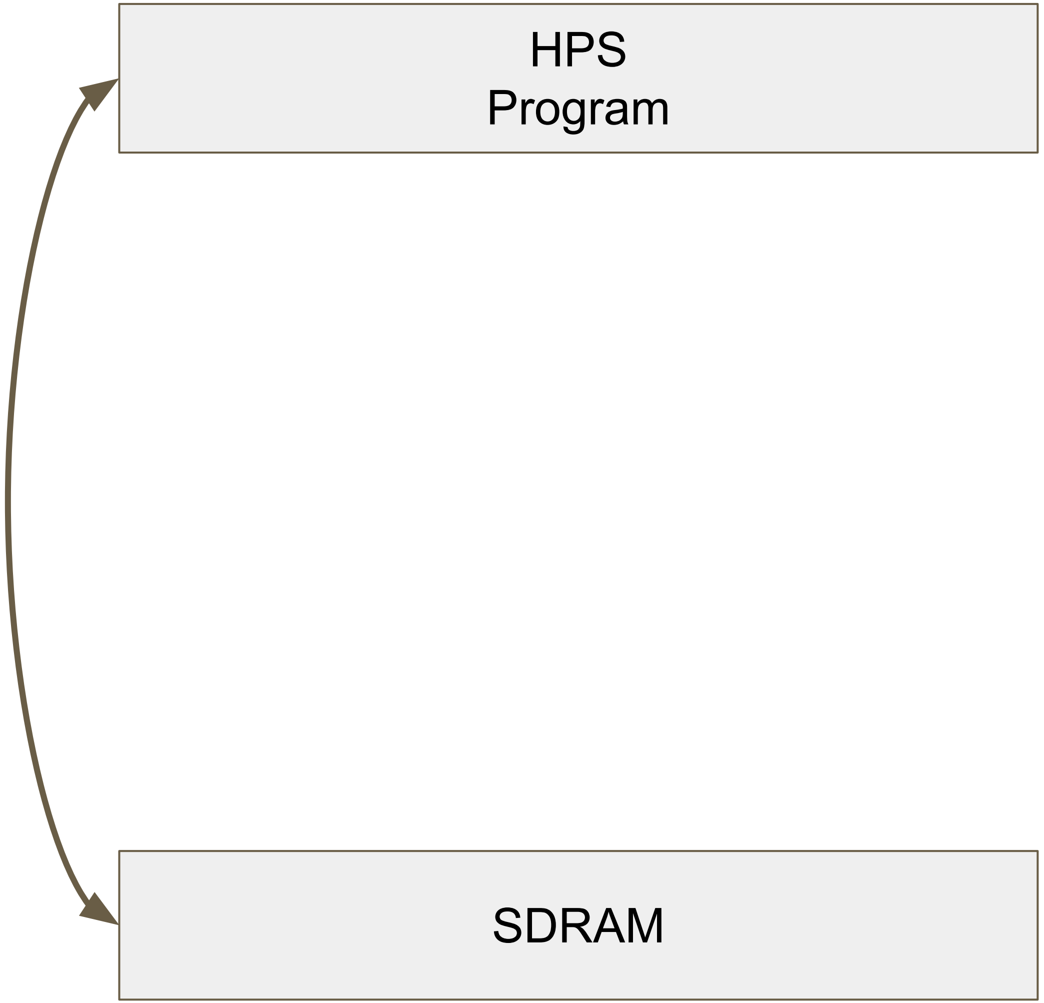
System Design



The figure below is the weight and bias for the conv2d part. Each float16 is divided into 2 uint8 and the figure's size is 1280 by 14407.

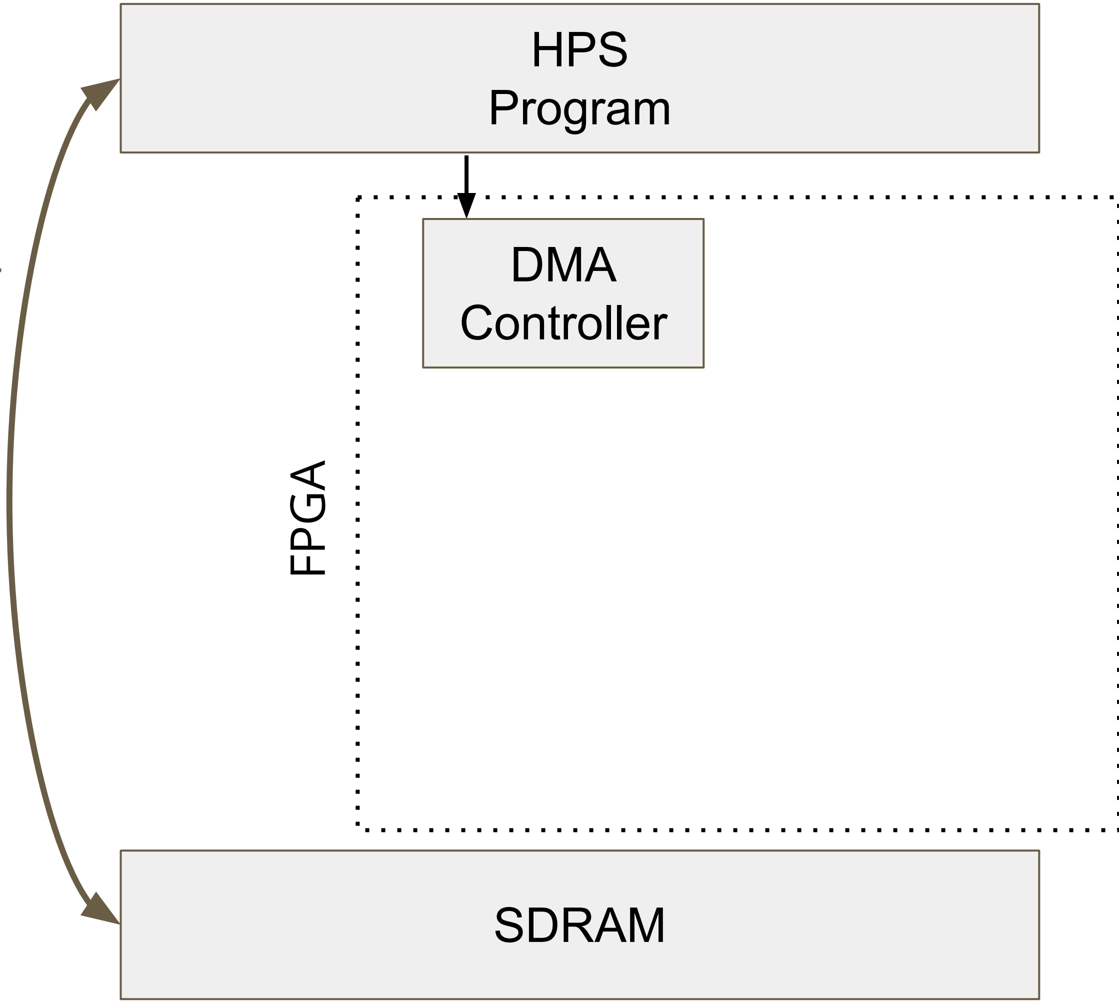
Hardware System

HPS store and retrieve data through SDRAM



Hardware System

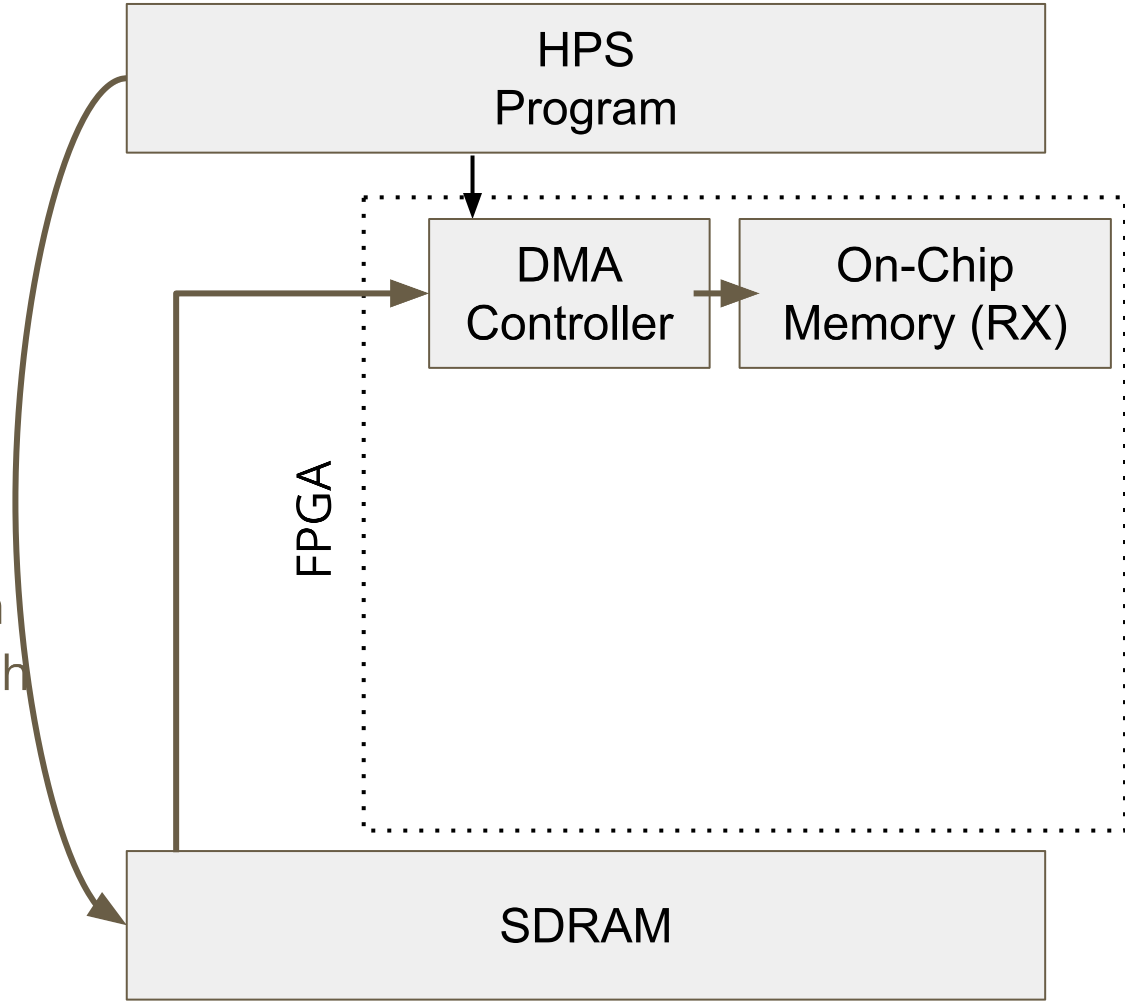
HPS connects to DMA Controller through light-weight AXI bridge



Hardware System

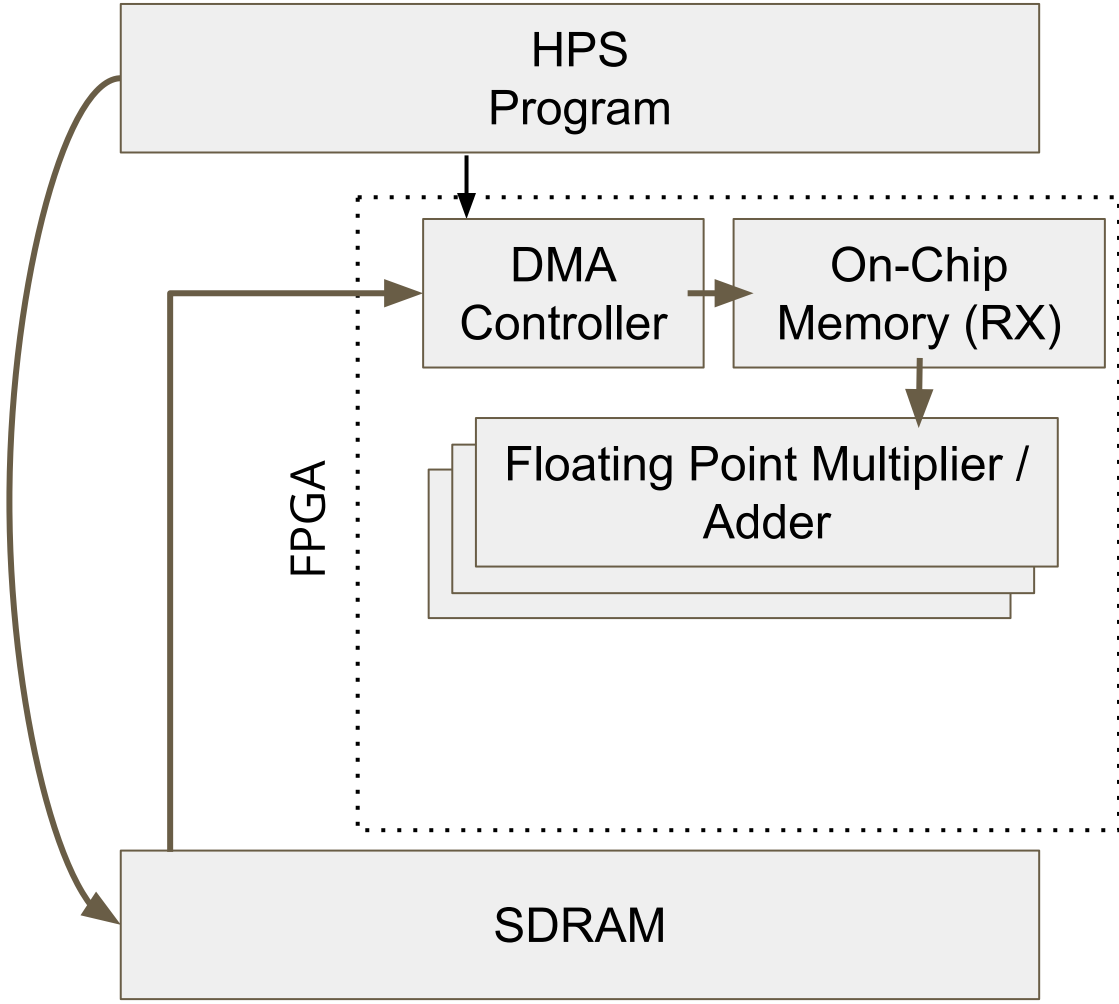
Data loaded onto FPGA through DMA controller and avalon memory-mapped slave

DMA Controller reads data from SDRAM into FPGA's OCM through AXI Bus



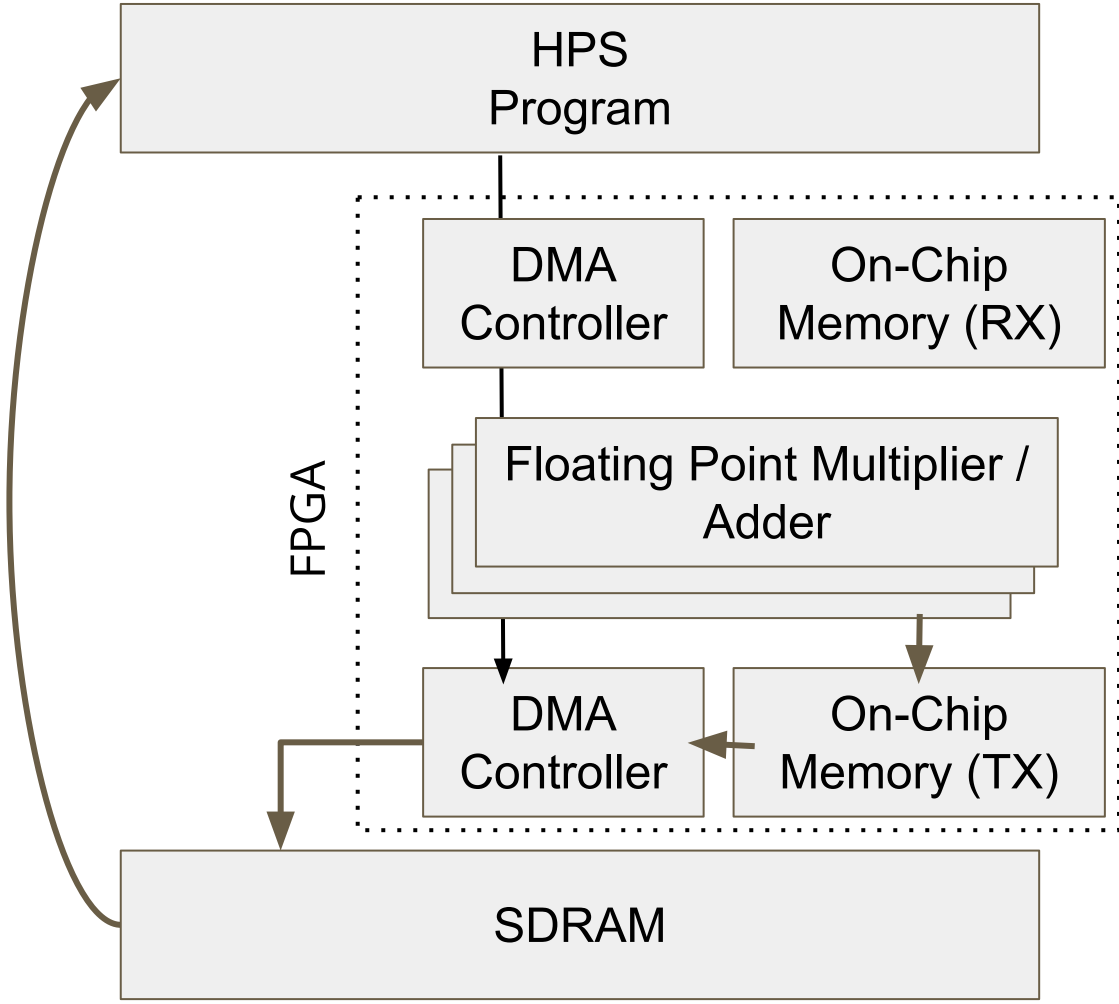
Hardware System

FPGA implements circuits to perform arithmetic operations for convolution



Hardware System

Data sent back through another set of OCM -> DMA -> SDRAM

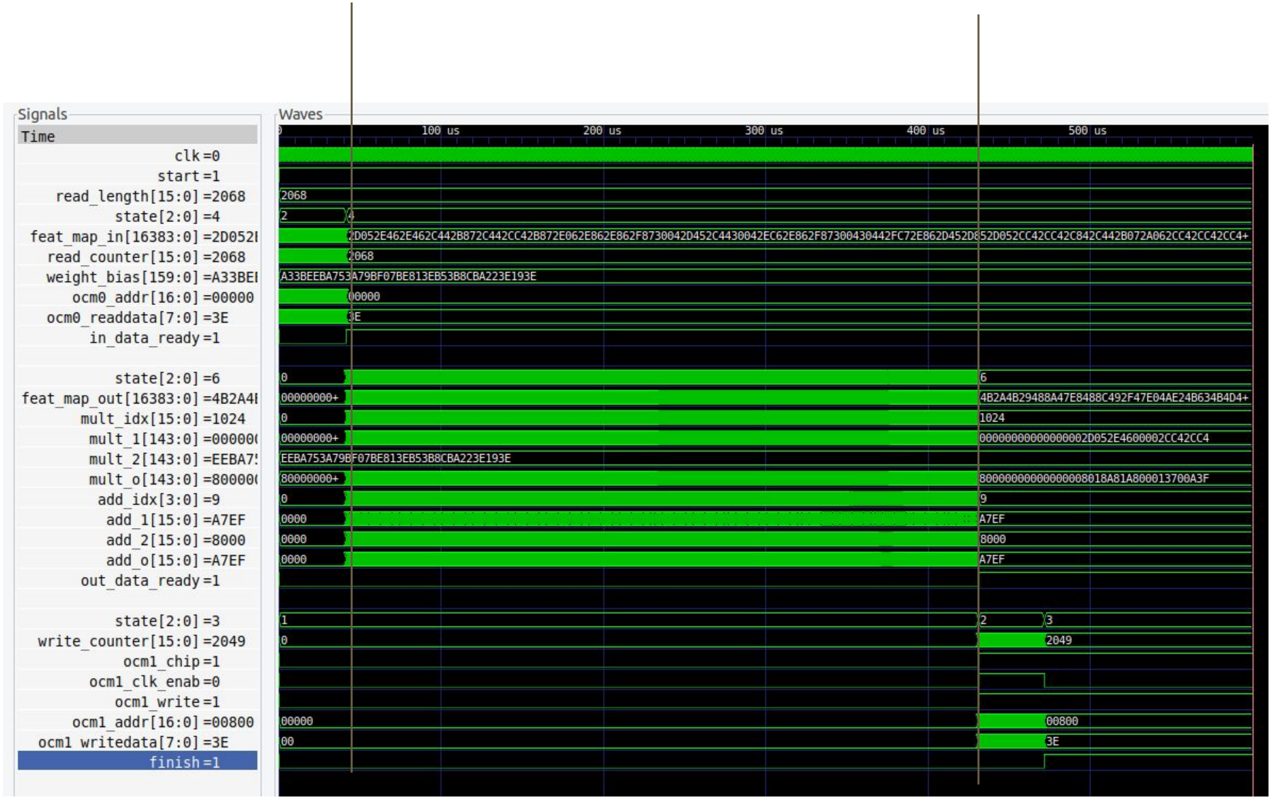


Version 1: 472us

Loading 32*32: 40us

Computation: 390us

Sending 32*32: 40us



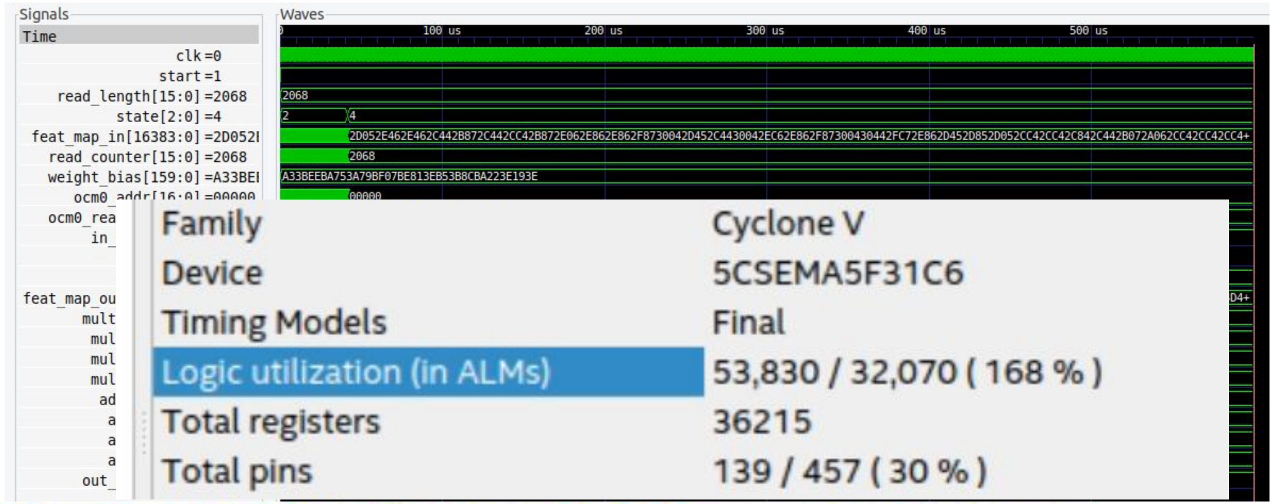
```
Time
                 clk=0
               start=1
   read_length[15:0]=2068
          state[2:0]=4
feat_map_in[16383:0]=2D052I
  read_counter[15:0]=2068
  weight_bias[159:0]=A33BEI
    ocm0_addr[16:0]=00000
  ocm0_readdata[7:0]=3E
       in_data_ready=1

          state[2:0]=6
feat_map_out[16383:0]=4B2A4I
     mult_idx[15:0]=1024
      mult_1[143:0]=00000(
      mult_2[143:0]=EEBA7!
      mult_o[143:0]=80000(
        add_idx[3:0]=9
        add_1[15:0]=A7EF
        add_2[15:0]=8000
        add_o[15:0]=A7EF
      out_data_ready=1

          state[2:0]=3
 write_counter[15:0]=2049
           ocm1_chip=1
       ocm1_clk_enab=0
          ocm1_write=1
    ocm1_addr[16:0]=00800
 ocm1_writedata[7:0]=3E
              finish=1
```

Version 1 - Not synthesizable

Loading 32*32: 40us

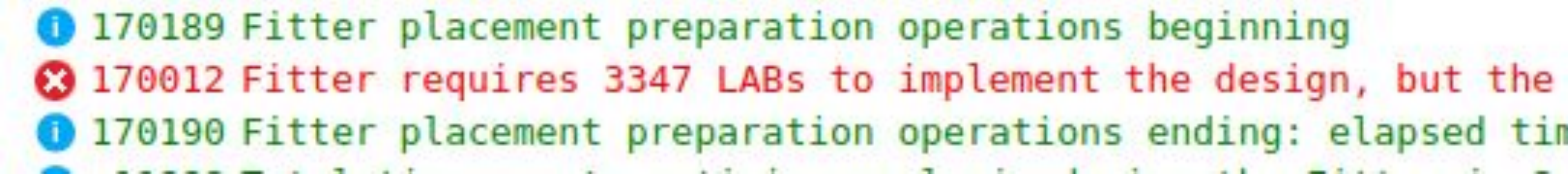
Computation: 390us

Sending 32*32: 40us



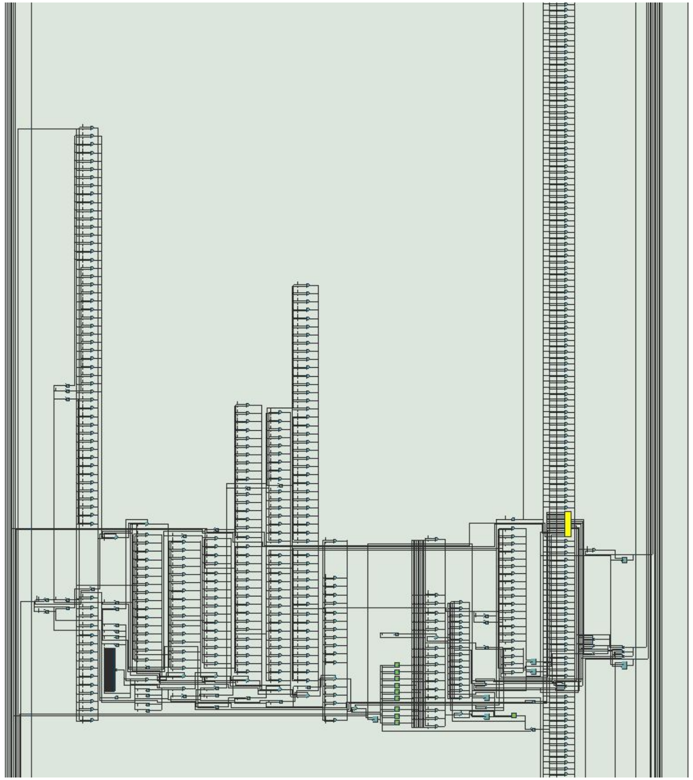
```
Signals                      Waves
Time                         0        100 us     200 us     300 us     400 us     500 us
                clk=0
              start=1
  read_length[15:0]=2068     2068
         state[2:0]=4        2    4
feat_map_in[16383:0]=2D052I       2D052E462E462C442B872C442CC42B872E062E862E862F8730042D452C4430042EC62E862F87300430442FC72E862D452D852D052CC42CC42C842C442B072A062CC42CC42CC4+
 read_counter[15:0]=2068          2068
 weight_bias[159:0]=A33BEI   A33BEEBA753A79BF07BE813EB53B8CBA223E193E
     ocm0_addr[16:0]=00000        00000
        ocm0_rea
              in_
        feat_map_ou                                                                                    D4+
               mult
                mul
                mul
                mul
                 ad
                  a
                  a
                  a
               out_
```

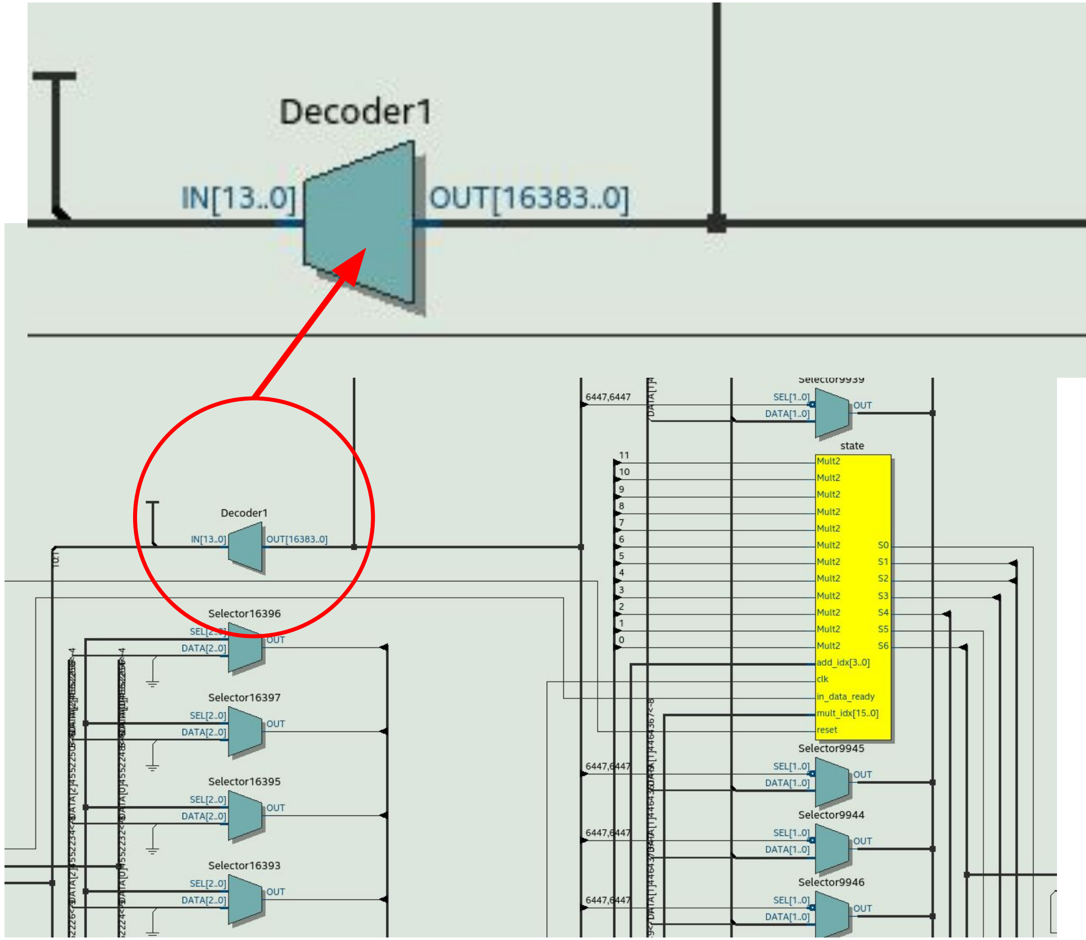
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	53,830 / 32,070 (168 %)
Total registers	36215
Total pins	139 / 457 (30 %)



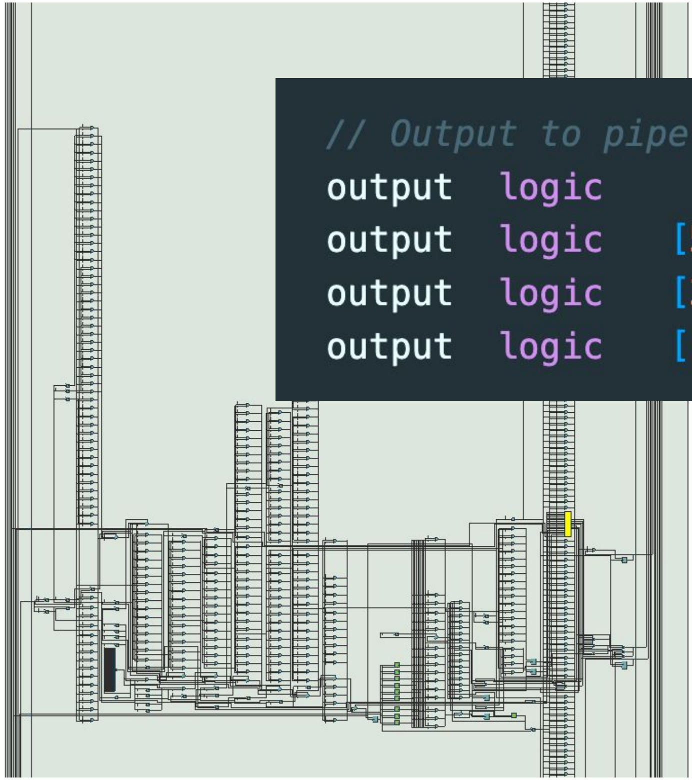
```
170189 Fitter placement preparation operations beginning
170012 Fitter requires 3347 LABs to implement the design, but the
170190 Fitter placement preparation operations ending: elapsed tin
```

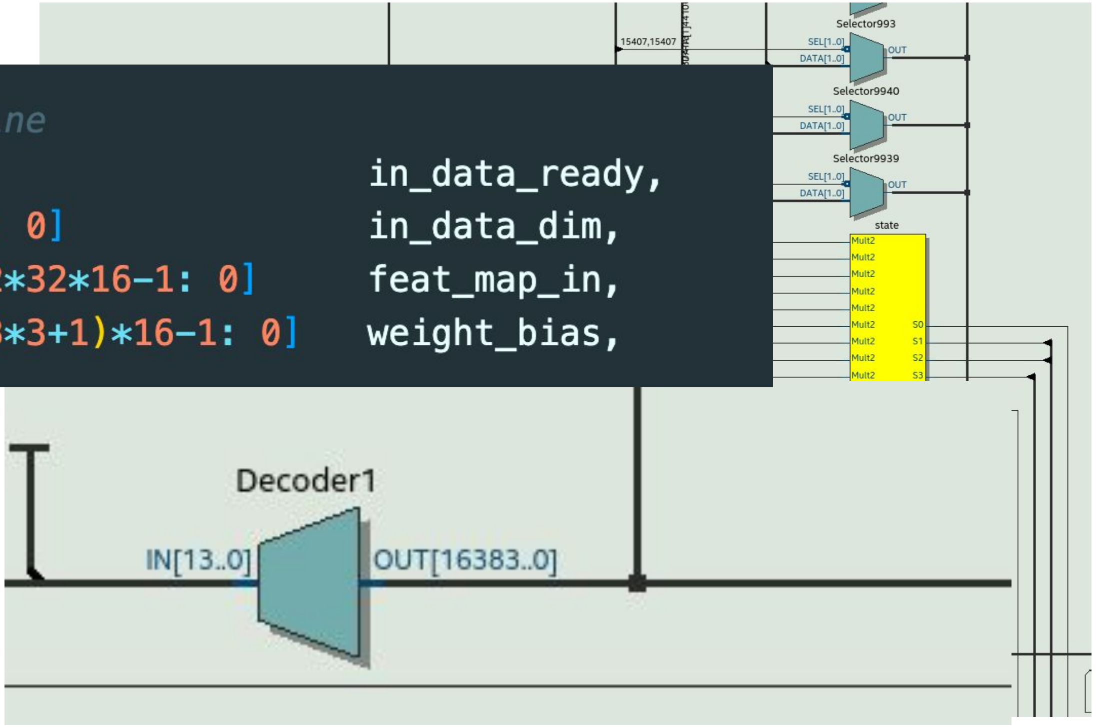
Version 1




Decoder1
IN[13..0]
OUT[16383..0]
Selector16396
Selector16397
Selector16395
Selector16393
SEL[2..0]
DATA[2..0]
OUT
Selector9939
Selector9945
Selector9944
Selector9946
SEL[1..0]
DATA[1..0]
state
Mult2
S0
S1
S2
S3
S4
S5
S6
add_idx[3..0]
clk
in_data_ready
mult_idx[15..0]
reset
6447,6447

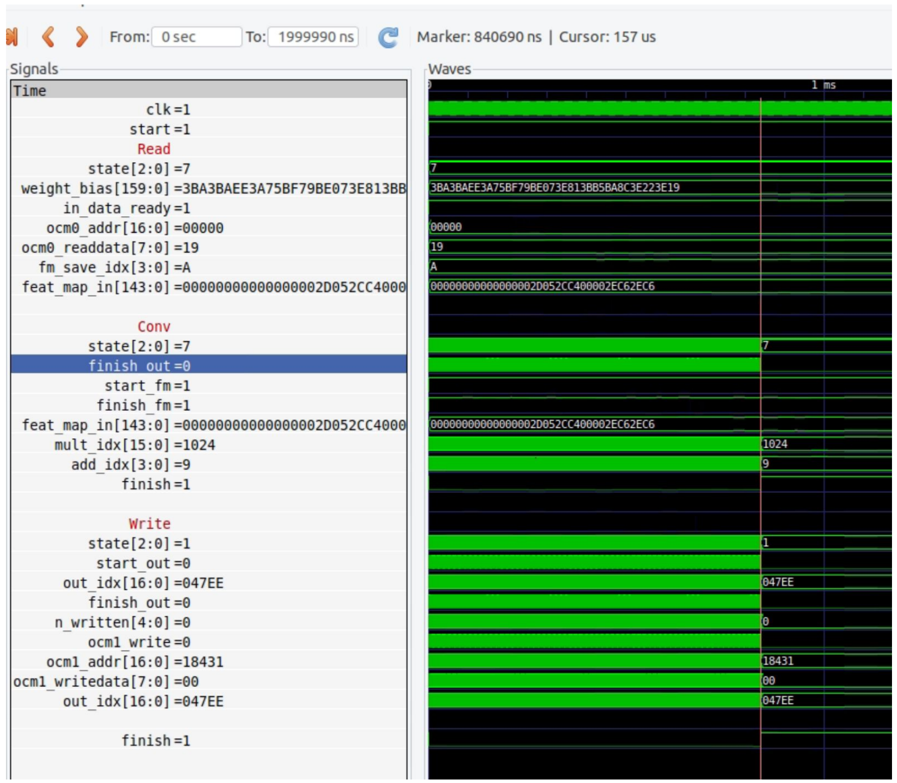
Version 1





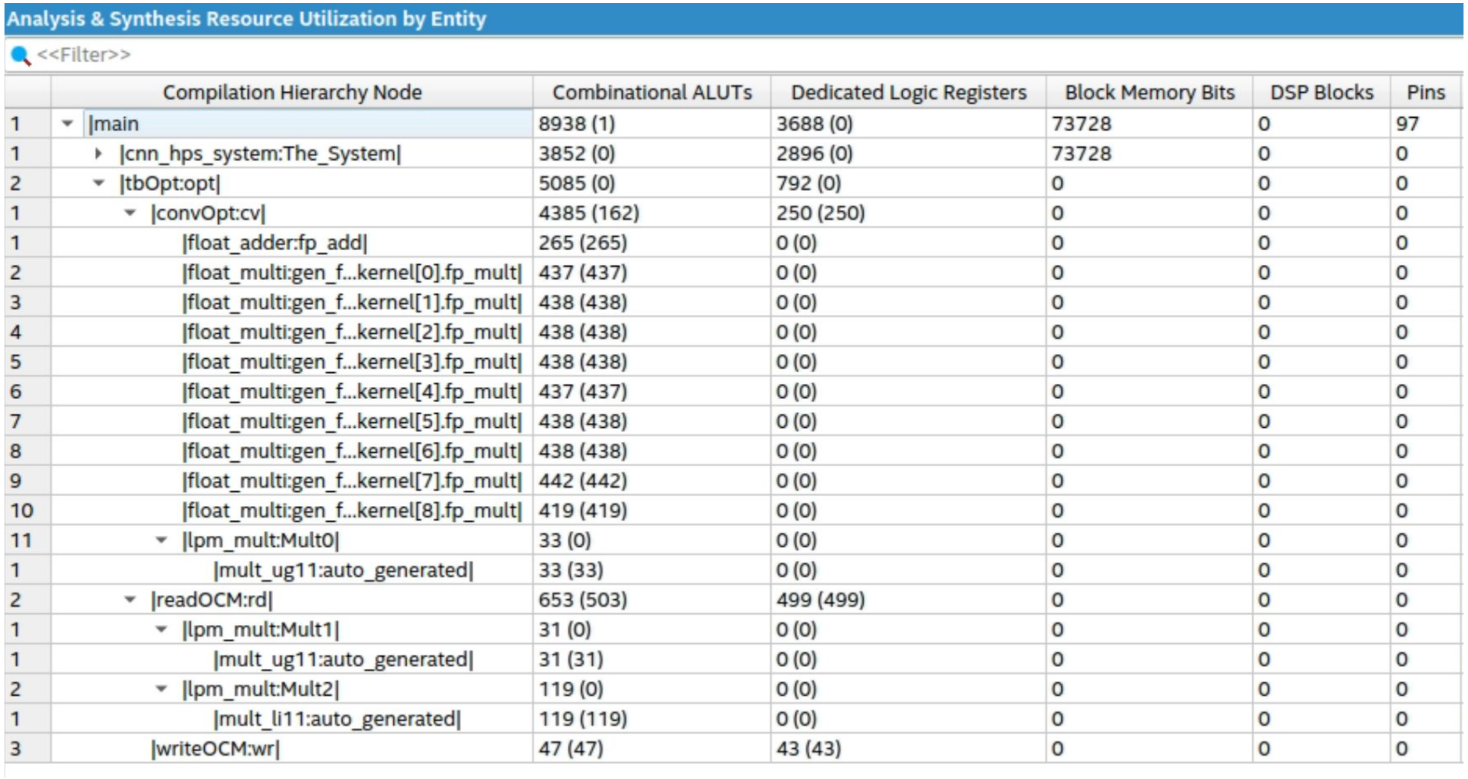
```
// Output to pipeline
output  logic                        in_data_ready,
output  logic   [5: 0]               in_data_dim,
output  logic   [32*32*16-1: 0]      feat_map_in,
output  logic   [(3*3+1)*16-1: 0]    weight_bias,
```

Time Consumption: 840us


1 ms
7
3BA3BAEE3A75BF79BE073E813BB5BA8C3E223E19
00000
19
A
00000000000000002D052CC400002EC62EC6
7
00000000000000002D052CC400002EC62EC6
1024
9
1
047EE
0
18431
00
047EE

```
From: 0 sec  To: 1999990 ns   Marker: 840690 ns | Cursor: 157 us

Signals
Time
                clk=1
              start=1
               Read
         state[2:0]=7
 weight_bias[159:0]=3BA3BAEE3A75BF79BE073E813BB
      in_data_ready=1
    ocm0_addr[16:0]=00000
 ocm0_readdata[7:0]=19
   fm_save_idx[3:0]=A
 feat_map_in[143:0]=00000000000000002D052CC4000

               Conv
         state[2:0]=7
         finish_out=0
           start_fm=1
          finish_fm=1
 feat_map_in[143:0]=00000000000000002D052CC4000
     mult_idx[15:0]=1024
       add_idx[3:0]=9
             finish=1

              Write
         state[2:0]=1
          start_out=0
      out_idx[16:0]=047EE
         finish_out=0
     n_written[4:0]=0
         ocm1_write=0
    ocm1_addr[16:0]=18431
ocm1_writedata[7:0]=00
      out_idx[16:0]=047EE

             finish=1
```

Utilization



Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins
1	|main	8938 (1)	3688 (0)	73728	0	97
1	|cnn_hps_system:The_System|	3852 (0)	2896 (0)	73728	0	0
2	|tbOpt:opt|	5085 (0)	792 (0)	0	0	0
1	|convOpt:cv|	4385 (162)	250 (250)	0	0	0
1	|float_adder:fp_add|	265 (265)	0 (0)	0	0	0
2	|float_multi:gen_f...kernel[0].fp_mult|	437 (437)	0 (0)	0	0	0
3	|float_multi:gen_f...kernel[1].fp_mult|	438 (438)	0 (0)	0	0	0
4	|float_multi:gen_f...kernel[2].fp_mult|	438 (438)	0 (0)	0	0	0
5	|float_multi:gen_f...kernel[3].fp_mult|	438 (438)	0 (0)	0	0	0
6	|float_multi:gen_f...kernel[4].fp_mult|	437 (437)	0 (0)	0	0	0
7	|float_multi:gen_f...kernel[5].fp_mult|	438 (438)	0 (0)	0	0	0
8	|float_multi:gen_f...kernel[6].fp_mult|	438 (438)	0 (0)	0	0	0
9	|float_multi:gen_f...kernel[7].fp_mult|	442 (442)	0 (0)	0	0	0
10	|float_multi:gen_f...kernel[8].fp_mult|	419 (419)	0 (0)	0	0	0
11	|lpm_mult:Mult0|	33 (0)	0 (0)	0	0	0
1	|mult_ug11:auto_generated|	33 (33)	0 (0)	0	0	0
2	|readOCM:rd|	653 (503)	499 (499)	0	0	0
1	|lpm_mult:Mult1|	31 (0)	0 (0)	0	0	0
1	|mult_ug11:auto_generated|	31 (31)	0 (0)	0	0	0
2	|lpm_mult:Mult2|	119 (0)	0 (0)	0	0	0
1	|mult_li11:auto_generated|	119 (119)	0 (0)	0	0	0
3	|writeOCM:wr|	47 (47)	43 (43)	0	0	0

Challenges

1. Huge feature map size. Although using vgg11 instead of vgg16, most layers’ feature map size is over the limit of the on chip memory.
2. Float point 16 multiplication and addition are slow.
3. Debugging needs a lot of test bench code.

Future Work

1. Add more floating point calculation module to parallelize the process.
2. Make the AXI buffer to the largest size to reduce the times of transmission.
3. Try to build int8 neural network instead of the fp16.