Designing Convolution 2D

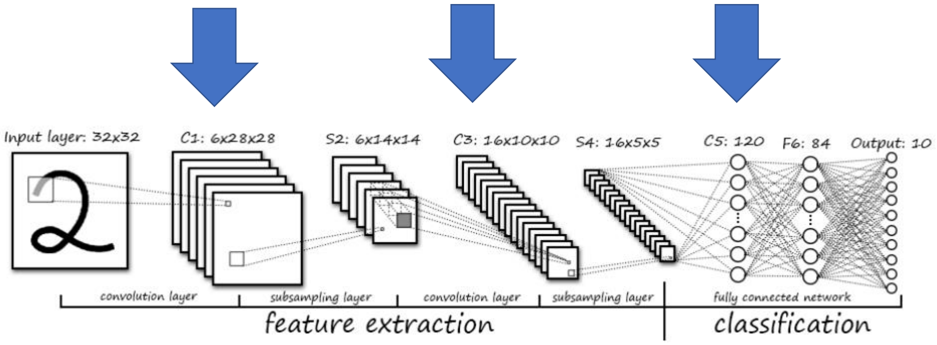
2020

Ando Ki, Ph.D.
adki@future-ds.com

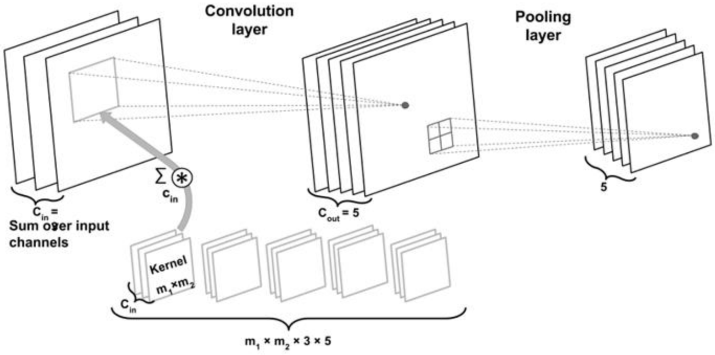
Table of contents

- Convolution layers in LeNet-5
- Typical convolution layer
- Reference 2D convolution in PyTorch
- Reference 2D convolution of Caffe V1
- 2D convolution with multi-channels
- Reference C code
- What to do
- Verification plan using HLS (ideal case)

Convolution layer in LeNet-5



Typical convolution layer



Reference 2D convolution

- PyTorch functional module

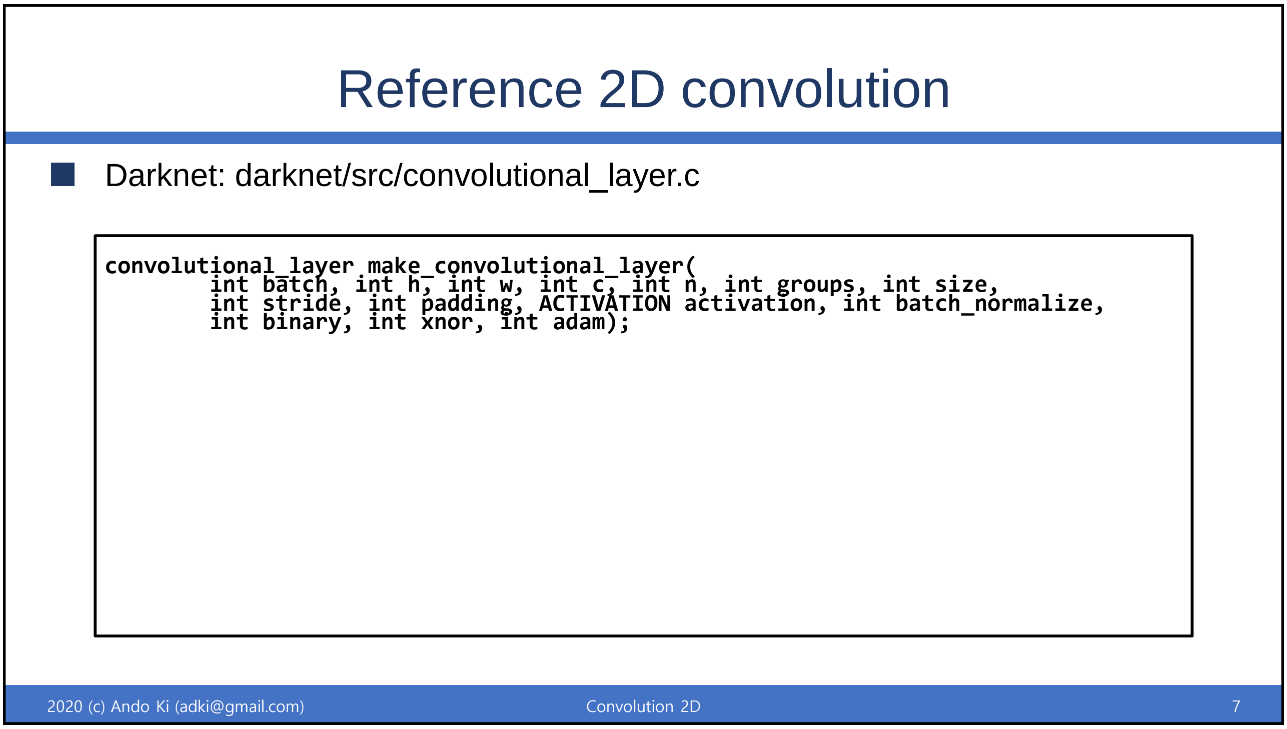
```
torch.nn.functional.conv2d(input, weight, bias=None,
                           stride=1, padding=0, dilation=1, groups=1)
```

- input – input tensor of shape (minibatch,in_channels,iH,iW)
- weight – filters of shape (out_channels,groups,kH,kW)
- bias – optional bias tensor of shape (out_channels)
- stride – the stride of the convolving kernel (s or (sH, sW))
- padding – implicit paddings on both sides of the input (pad or (padH, padW))
- dilation – the spacing between kernel elements (d or (dH, dW))
- groups – split input into groups

Reference 2D convolution

- Caffe V1: src/caffe/proto/caffe.proto

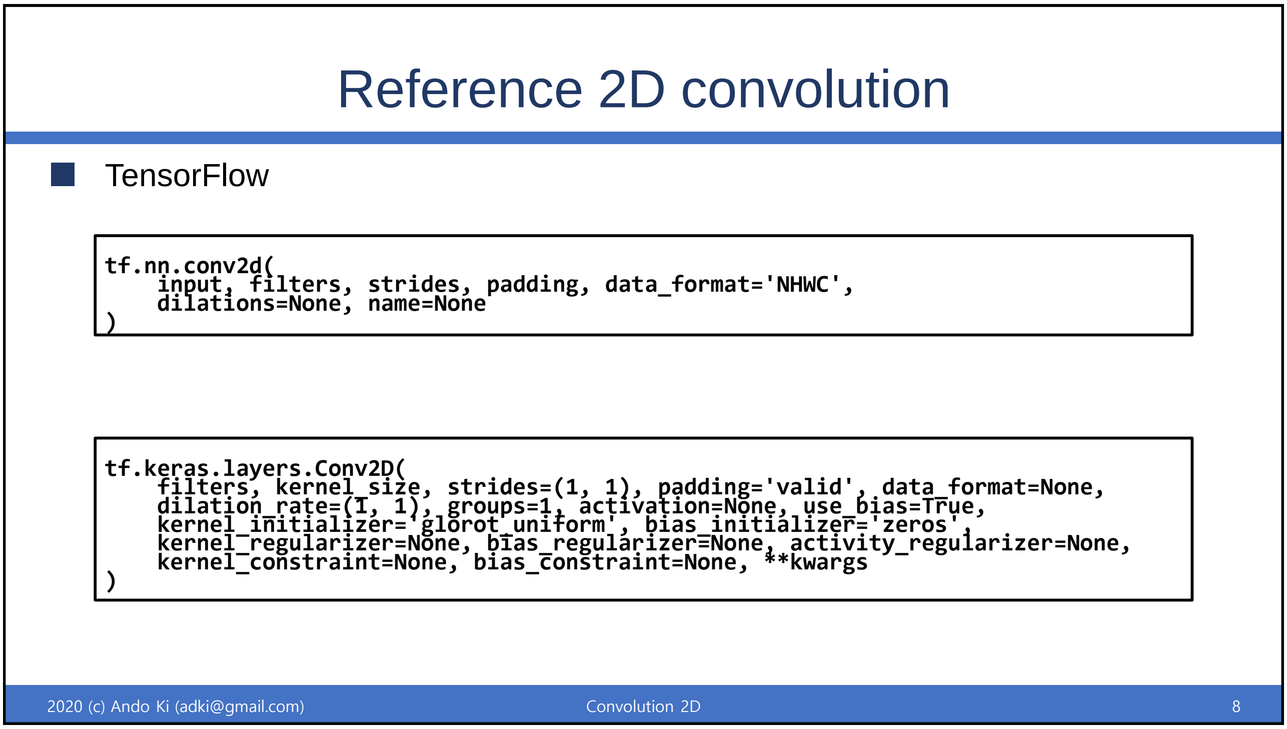
```
message ConvolutionParameter {
  optional uint32 num_output = 1; // The number of outputs for the layer
  optional bool bias_term = 2 [default = true]; // whether to have bias terms

  repeated uint32 pad = 3; // The padding size; defaults to 0
  repeated uint32 kernel_size = 4; // The kernel size
  repeated uint32 stride = 6; // The stride; defaults to 1
  repeated uint32 dilation = 18; // The dilation; defaults to 1

  optional uint32 pad_h = 9 [default = 0]; // The padding height (2D only)
  optional uint32 pad_w = 10 [default = 0]; // The padding width (2D only)
  optional uint32 kernel_h = 11; // The kernel height (2D only)
  optional uint32 kernel_w = 12; // The kernel width (2D only)
  optional uint32 stride_h = 13; // The stride height (2D only)
  optional uint32 stride_w = 14; // The stride width (2D only)

  optional uint32 group = 5 [default = 1]; // The group size for group conv

  ......
}
```



Reference 2D convolution

- Darknet: darknet/src/convolutional_layer.c

```
convolutional_layer make_convolutional_layer(
        int batch, int h, int w, int c, int n, int groups, int size,
        int stride, int padding, ACTIVATION activation, int batch_normalize,
        int binary, int xnor, int adam);
```



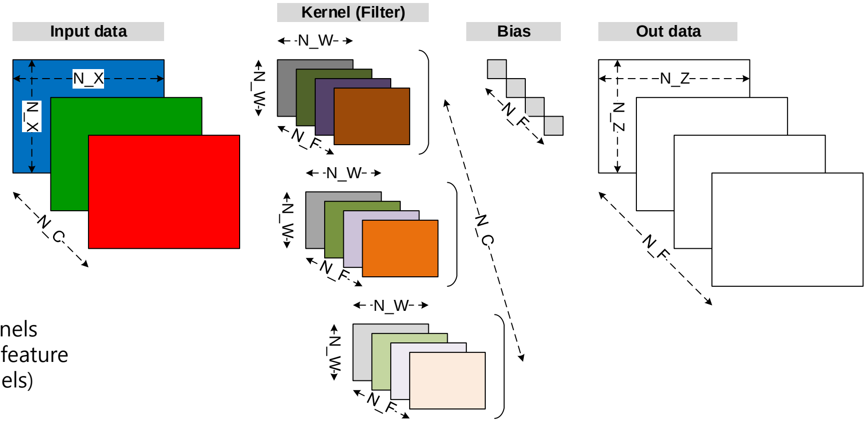
Reference 2D convolution

- TensorFlow

```
tf.nn.conv2d(
    input, filters, strides, padding, data_format='NHWC',
    dilations=None, name=None
)
```

```
tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform', bias_initializer='zeros',
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, bias_constraint=None, **kwargs
)
```

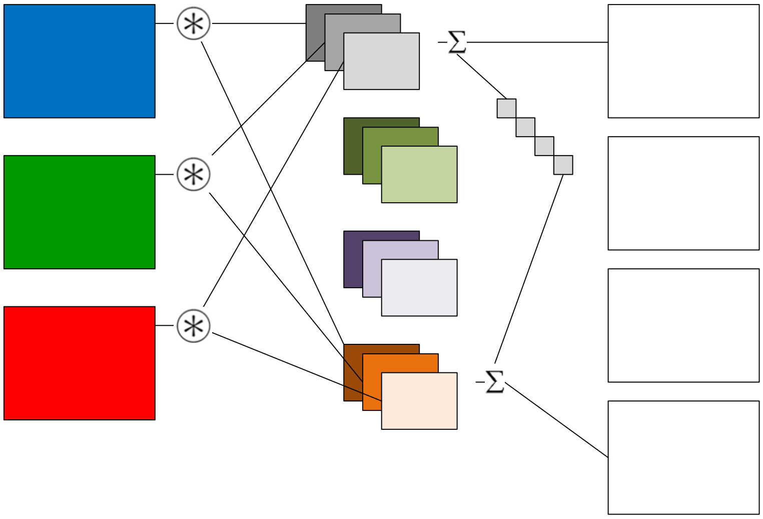
2D convolution with multi-channels



N_C: number of input channels
N_X: width/height of input feature
N_F: number of filters (kernels)
N_S: stride
N_P: padding
N_W: width/height of filter
N_Z: width/height of output

N_Z = (((N_X-N_W+2*N_P)/N_S)+1)

2D convolution with multi-channels



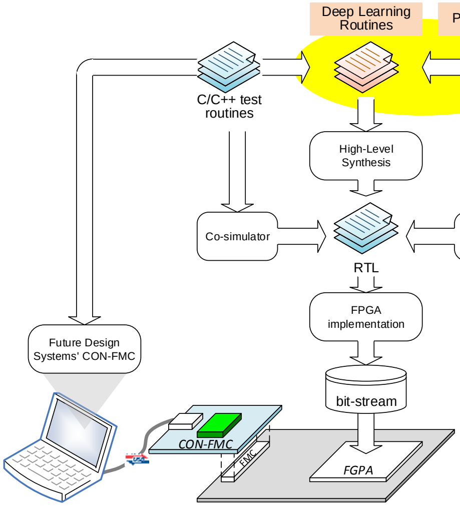
Reference C code

```
void convolution_2d ( const DTYPE   X[N_C][N_X][N_X]
                    , const DTYPE   W[N_C][N_F][N_W][N_W]
                    ,       DTYPE   Z[N_F][N_Z][N_Z]
                    , const DTYPE   bias[N_B] // N_B=N_F
                    , const uint8_t stride=1)
{
    uint8_t  ch, f;
    uint16_t i, j, r, c;
    for (f=0; f<N_F; ++f) {
        for (r = 0; r < N_Z; ++r) {
            for (c = 0; c < N_Z; ++c) {
                Z[f][r][c] = bias[f];
            }
        }
        for (ch=0; ch<N_C; ++ch) {
            for (r=0; r<(N_X - N_W + 1); r += stride) {
                for (c=0, i=0, j=0; c<(N_X - N_W + 1); c += stride) {
                    for (i=0; i<N_W; ++i) {
                        for (j=0; j<N_W; ++j) {
                            Z[f][r][c] += X[ch][r+i][j+c] * W[ch][f][i][j];
                        }
                    }
                }
            }
        }
    }
}
```

What to do

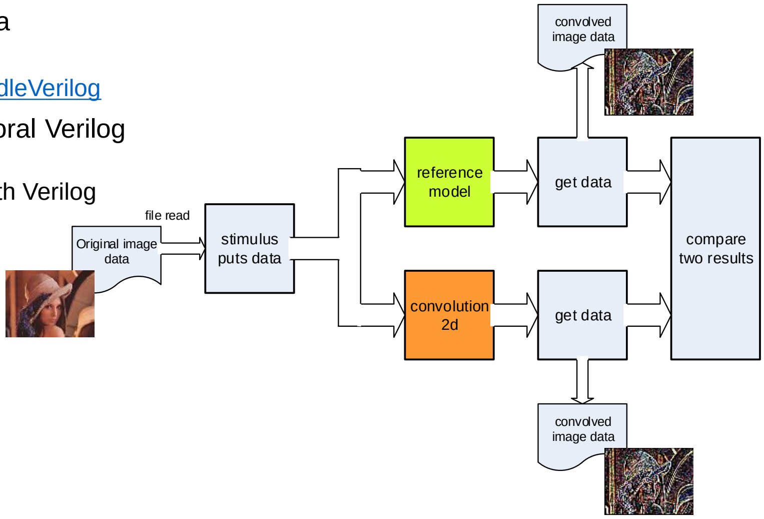
- Plan demonstration environment
  - ▶ 결과 시연 환경을 고려
- Plan how to interface with outside including activation function and pooling
  - ▶ 모듈의 외부와 어떻게 인터페이스 할 것인지 고려
  - ▶ 활성함수와 풀링을 연결할 경우도 고려
- Plan how to verify the functionality.
  - ▶ 기능을 어떻게 검증 할 것인지 고려
- Design ‘convolution_2d’ and its test-bench in parallel using Verilog-HDL, in which all convolution parameters should be parameterized.
  - ▶ ‘convolution_2d’와 ‘test-bench’를 동시에 설계
  - ▶ 콘볼루젼 관련 파라메터들은 변경이 가능하도록 설계
- Use your design as a kind of image filter such as edge-detection or smoothing
  - ▶ 설계를 이미지 필터에 적용해 본다: 윤곽선 검출, 고주파 노이즈 제거, ...

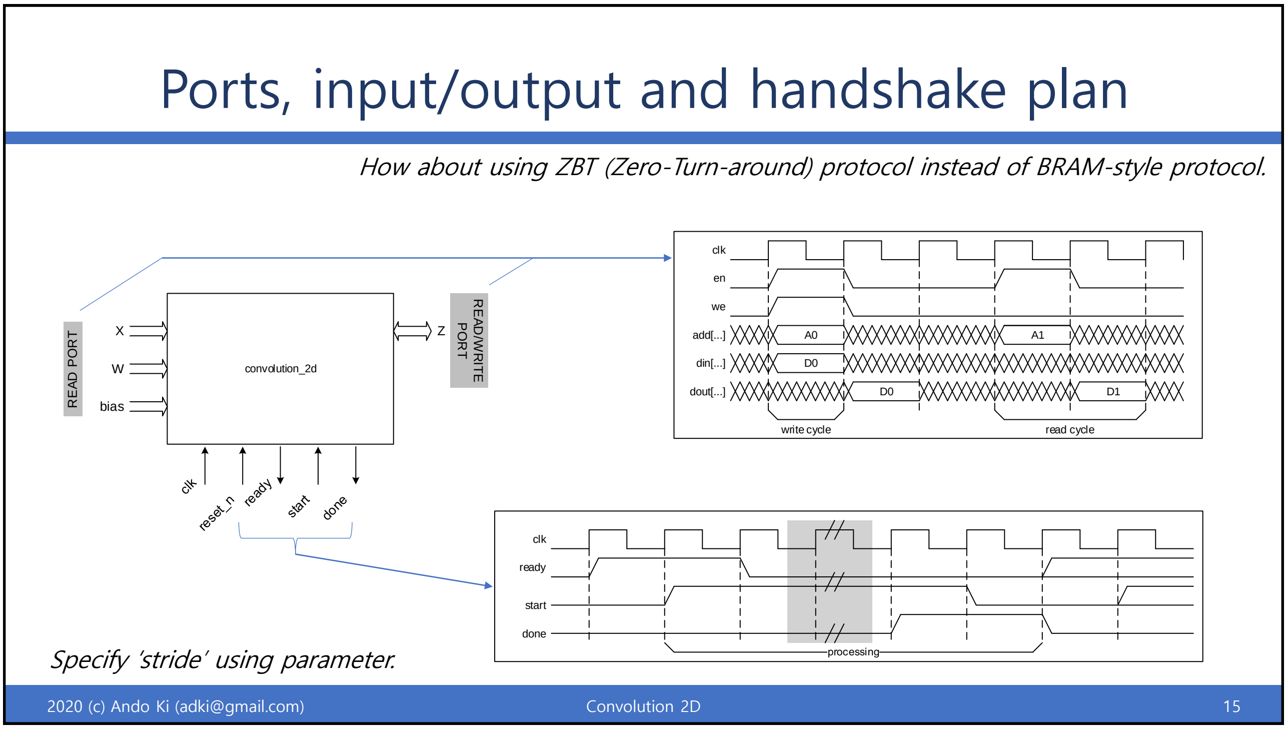
Verification plan using HLS (ideal case)



Verification plan

- Use BMP image file as input data
  - refer to following GitHub page
  - https://github.com/adki/BmpHandleVerilog
- Use reference C code or behavioral Verilog code to check result
  - Use VPI or DPI to interface C with Verilog
- Choose kernel
  - Canny or Sobel edge detector



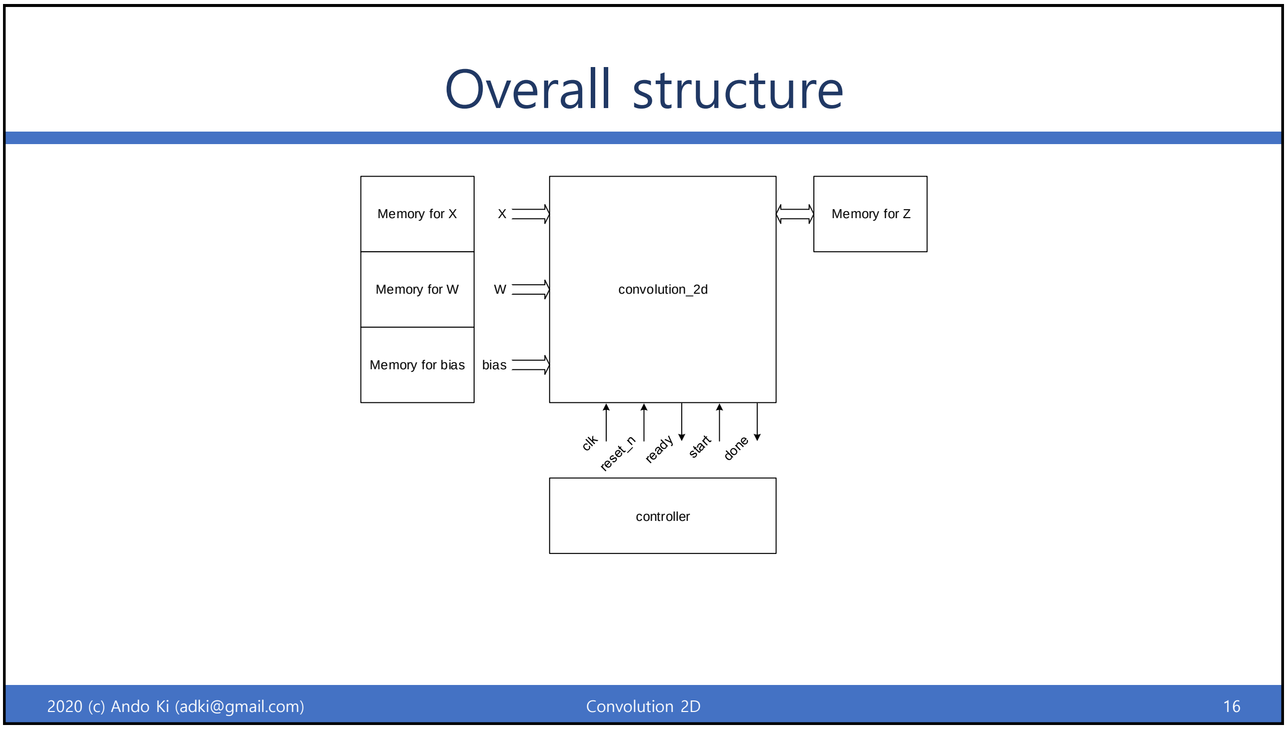


Ports, input/output and handshake plan

How about using ZBT (Zero-Turn-around) protocol instead of BRAM-style protocol.

Specify 'stride' using parameter.

2020 (c) Ando Ki (adki@gmail.com) Convolution 2D 15

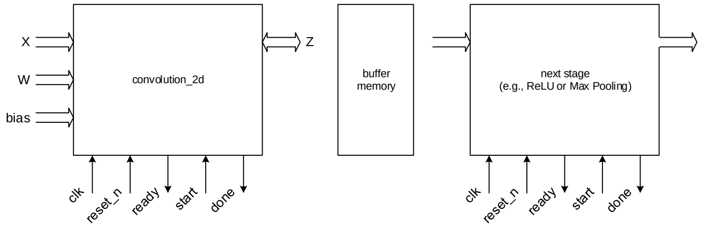


Overall structure

2020 (c) Ando Ki (adki@gmail.com) Convolution 2D 16

System-wide point of view

- How to integrate with other stage?


X
W
bias
convolution_2d
Z
clk
reset_n
ready
start
done
buffer
memory
next stage
(e.g., ReLU or Max Pooling)
clk
reset_n
ready
start
done