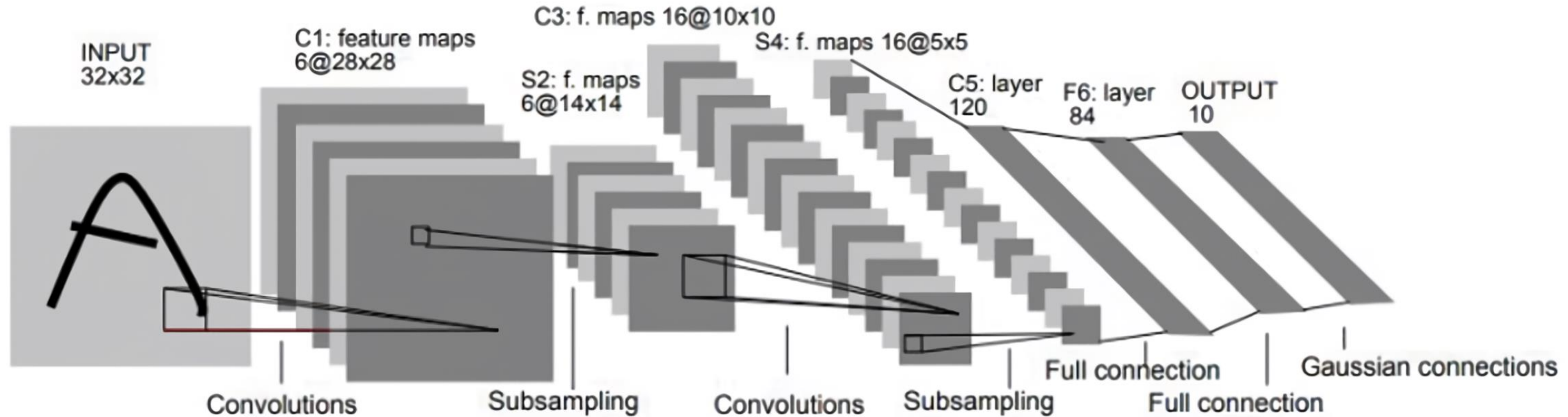


# LeNet-5 on ZYBO Z7-10 FPGA

# Index

- LeNet-5 Introduction
  - Components of LeNet-5
  - Parameters of LeNet-5
- Optimization for FPGA Implementation (Model)
  - Activation function
  - Model lightening & Parametes of Lite LeNet-5
- Description of each Layer
  - 1<sup>st</sup> Convolution
  - 1<sup>st</sup> Pooling
  - 2<sup>nd</sup> Convolution
  - 2<sup>nd</sup> Pooling
  - 3<sup>rd</sup> Convolution
  - 1<sup>st</sup> Fully Connected
  - 2<sup>nd</sup> Fully Connected
- Overview of Optimized LeNet-5 (Light Light LeNet-5)
- Implementation of LeNet-5 using Python
- Implementation of LeNet-5 using C/C++
  - Fixed-Point vs. Floating-Point
  - Original LeNet-5 with C/C++
  - Light LeNet-5 with C/C++ (Floating-Point)
  - Light LeNet-5 with C/C++ (Fixed-Point)
- Implementation of LeNet-5 using HLS
  - Light LeNet-5 with HLS (Stream Interface)
  - Light LeNet-5 with HLS (Partially Parallel)
  - Light LeNet-5 with HLS (Optimize)
- Overview of Implementation of LeNet-5 using C/C++ and HLS
- HW Design of LeNet-5 (Programmable Logic)
- SW Driver of LeNet-5 (Processing System)
- Performance Analysis, ARM Core vs. Predict IP
- Overview of HW Design & SW Driver & Perf.
- References

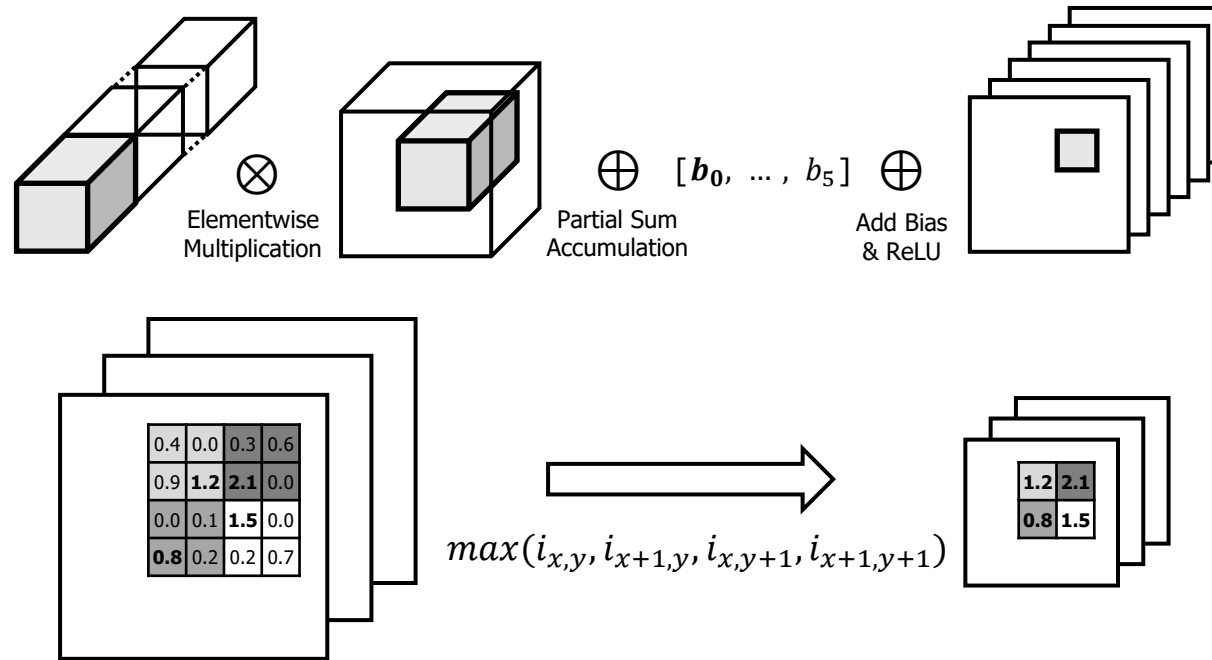
# LeNet-5 Introduction



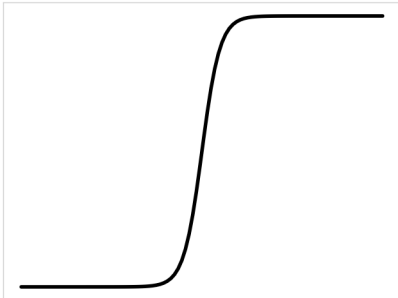
- LeNet-5 is a convolutional neural network (CNN) architecture.
- One of the first neural network architectures to achieve high accuracy on the MNIST handwritten digit recognition.
- Foundational architecture in the development of deep learning.

# Components of LeNet-5

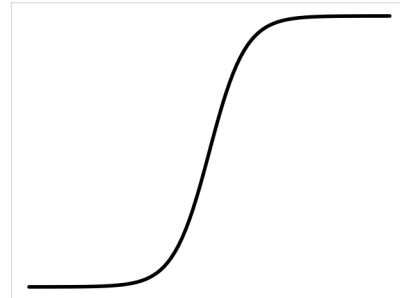
- Convolution
- Pooling
- Fully connected
- Activation functions
  - *ReLU*
  - *sigmoid*
  - *tanh*



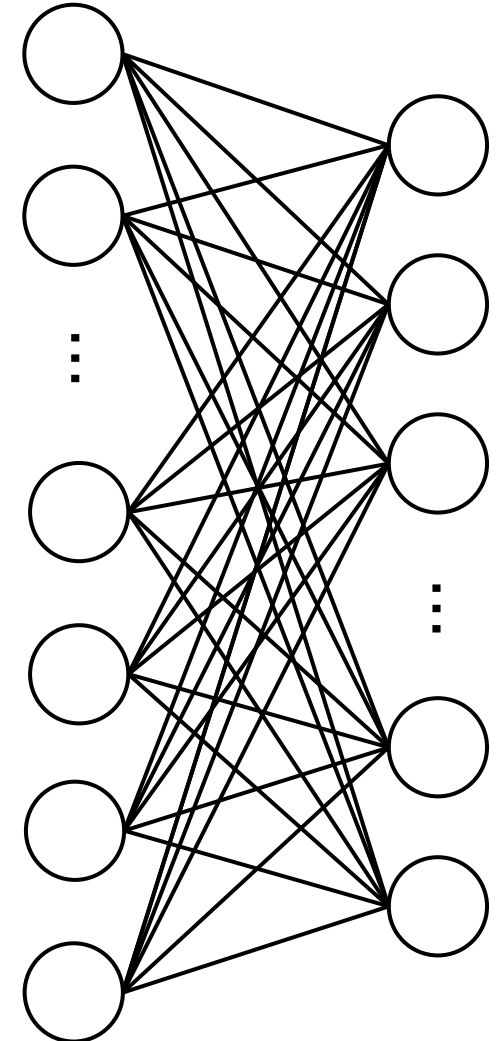
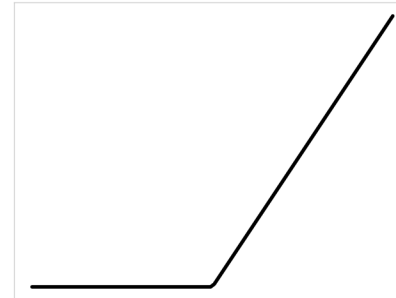
$$\tanh(x) = \frac{1 - \exp(-x)}{1 + \exp(-x)}$$



$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



$$\text{ReLU}(x) = \max(x, 0)$$



# Parameters of Original LeNet-5

- ZYBO Z7-10 board's resources are not enough to implement Original LeNet-5.
- ZYBO Z7-10 has far fewer resources than the ZYBO Z7-20.

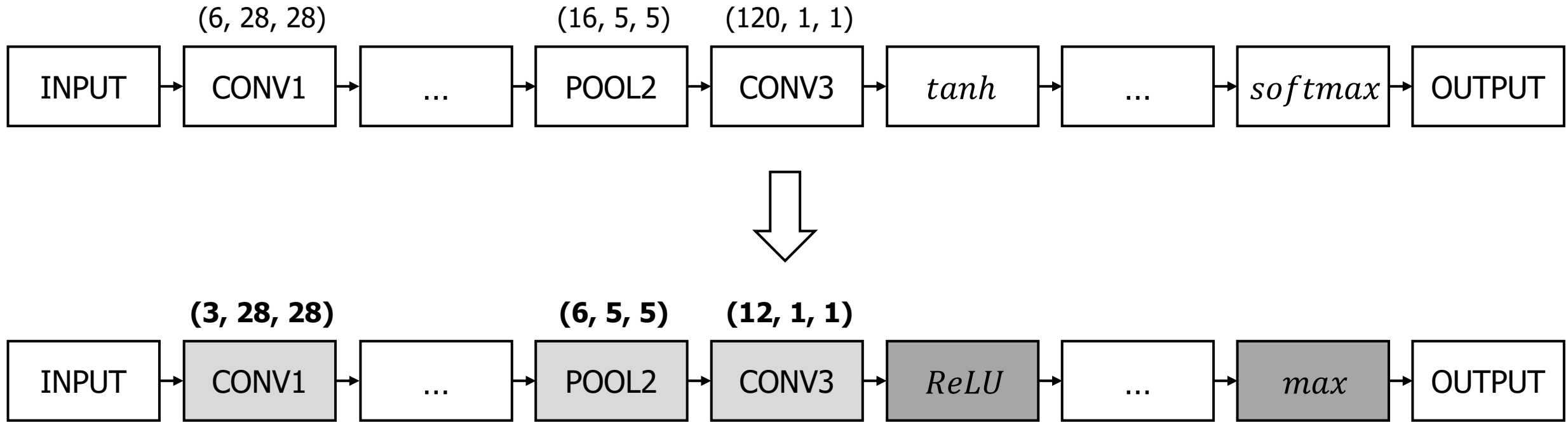
Layer	C	W x H	Kernel	Stride	Activation	Weight + Bias	Feature map
INPUT	1	32 x 32	-	-	-	-	1024
CONV1	6	28 x 28	5x5	1	<i>tanh</i>	$150 + 6 = 156$	4704
A.POOL1	6	14 x 14	2x2	2	<i>sigmoid</i>	-	1176
CONV2	16	10 x 10	5x5	1	<i>tanh</i>	$2400 + 16 = 2416$	1600
A.POOL2	16	5 x 5	2x2	2	<i>sigmoid</i>	-	400
CONV3	120	1 x 1	5x5	1	<i>tanh</i>	$48000 + 120 = 48120$	120
FC1	-	120	-	-	<i>tanh</i>	$10080 + 84 = 10164$	84
FC2	-	84	-	-	<i>softmax</i>	$840 + 10 = 850$	10
<b>Total</b>	-	-	-	-	-	<b>61706</b>	<b>9118</b>

	<b>ZYBO Z7-10</b>
LUT	17,600
Flip-Flop	35,200
Block RAM	270 KB
DSP	80
Slice	4,400

	<b>ZYBO Z7-20</b>
LUT	53,200
Flip-Flop	106,400
Block RAM	630 KB
DSP	220
Slice	13,300

# Optimization for FPGA Implementation (Model-level)

- Model-level optimization of LeNet-5 from a hardware accelerator perspective.
  - Activation function
  - Model lightening



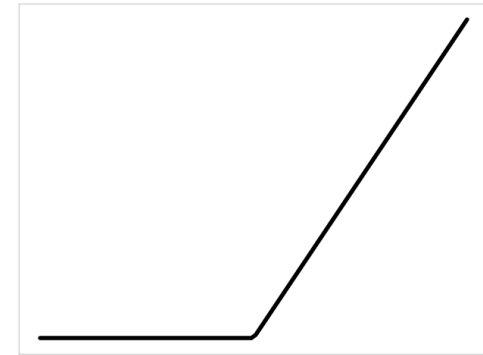
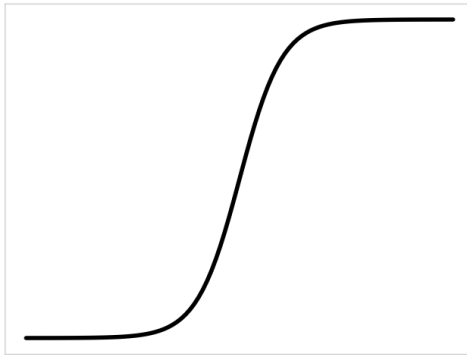
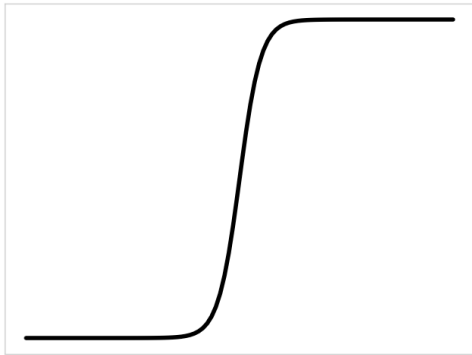
# Activation Function (*ReLU* vs. *tanh*, *sigmoid*)

- $\tanh(x)$  or  $\text{sigmoid}(x)$  function in hardware requires additional resources to perform exponentiation and division operations.
- $\text{ReLU}(x)$  function can be implemented using basic logical and arithmetic operations, resulting in minimal hardware complexity.
- $\text{ReLU}(x)$  helps alleviate the vanishing gradient problem that occurs in activation functions such as  $\tanh(x)$  or  $\text{sigmoid}(x)$ .

$$\tanh(x) = \frac{1 - \exp(-x)}{1 + \exp(-x)}$$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

$$\text{ReLU}(x) = \max(x, 0)$$



# Activation Function (*max* vs. *softmax*), Cont'd

- $softmax(\vec{x})$  function in hardware also requires additional resources to perform exponentiation and division operations.
- $max(\vec{x})$  function also can be implemented using basic logical and arithmetic operations, resulting in minimal hardware complexity.
- For both  $softmax(\vec{x})$  and  $max(\vec{x})$  functions, using the maximum value is the same, so the  $max(\vec{x})$  function was used in consideration of efficiency.

$$softmax(\vec{x}) = \frac{\exp(x_i)}{\sum_{j=1}^k \exp(x_j)}, (for\ i = 1, 2, \dots, k)$$

$$max(\vec{x}) = max(x_0, x_1, \dots, x_k)$$

[0.96 -0.45 ... 0.41 -0.5 0.12]

↓  $softmax(\vec{x})$

[0.97 0.0 ... 0.02 0.0 0.01]

↓  $max(\vec{x})$  &  $getIndex(x_{max})$

[0]

[0.96 -0.45 ... 0.41 -0.5 0.12]

↓  $max(\vec{x})$  &  $getIndex(x_{max})$

[0]



# Model lightening & Parameters of Light LeNet-5

- Adjust the parameters of each layer to make the model smaller.
- *sigmoid*, *tanh* are replaced by *ReLU*.
- Weights and biases were reduced by **96%**, and feature maps were reduced by **48%**.

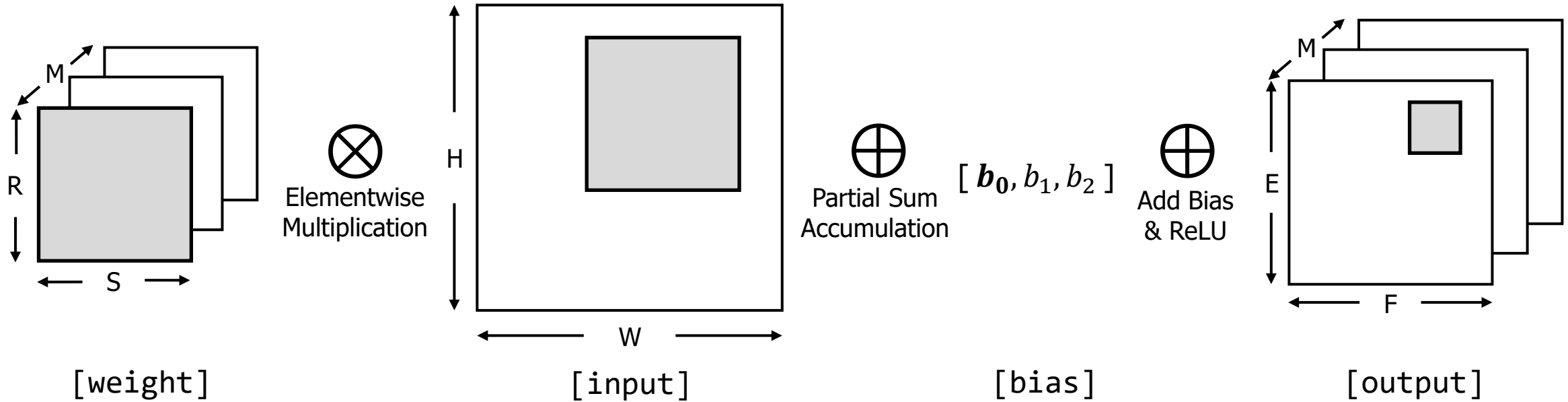
Layer	C	W x H	Kernel	Stride	Activation	Weight + Bias	Feature map
INPUT	1	32 x 32	-	-	-	-	1024
CONV1	3	28 x 28	5x5	1	<i>ReLU</i>	$75 + 3 = 78$	2352
M.POOL1	3	14 x 14	2x2	2	-	-	588
CONV2	6	10 x 10	5x5	1	<i>ReLU</i>	$450 + 6 = 456$	600
M.POOL2	6	5 x 5	2x2	2	-	-	150
CONV3	12	1 x 1	5x5	1	<i>ReLU</i>	$1800 + 12 = 1812$	12
FC1	-	120	-	-	<i>ReLU</i>	$120 + 10 = 130$	10
FC2	-	84	-	-	<i>Max</i>	$100 + 10 = 110$	10
<b>Total</b>	-	-	-	-	-	<b>2586</b>	<b>4746</b>

	<b>ZYBO Z7-10</b>
LUT	17,600
Flip-Flop	35,200
Block RAM	270 KB
DSP	80
Slice	4,400

	<b>ZYBO Z7-20</b>
LUT	53,200
Flip-Flop	106,400
Block RAM	630 KB
DSP	220
Slice	13,300

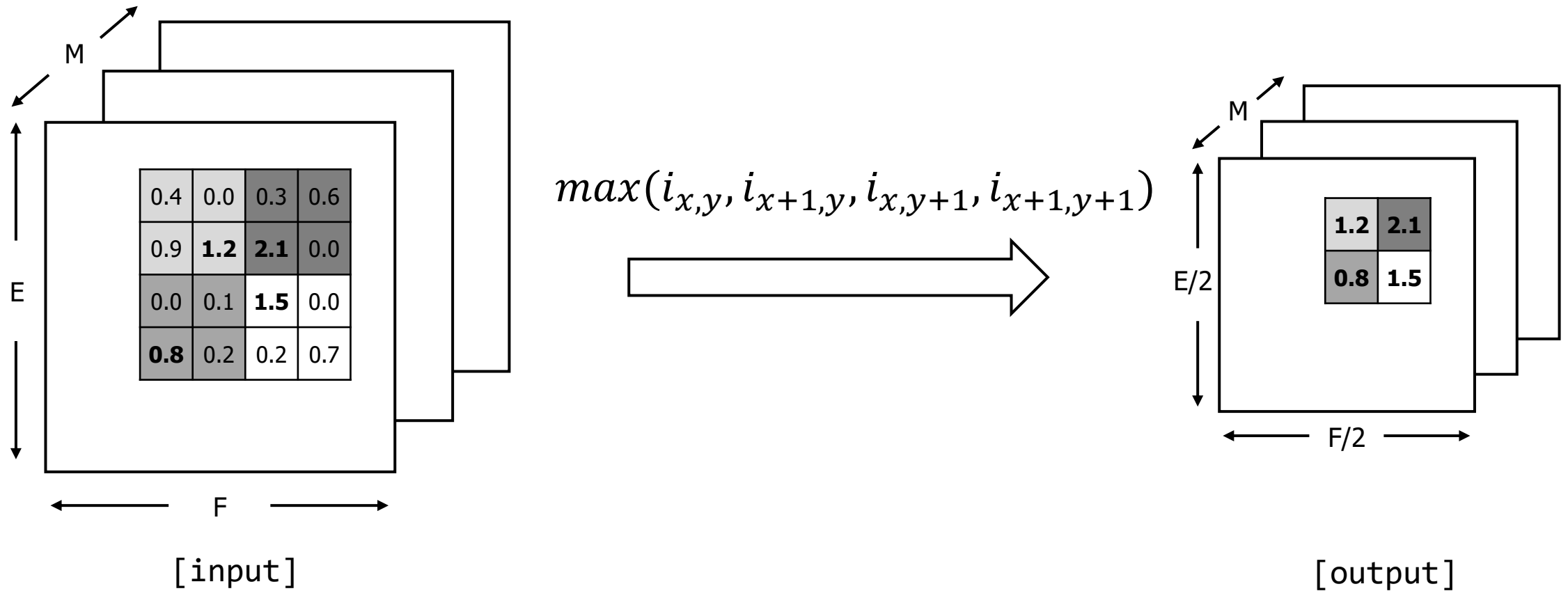
# 1<sup>st</sup> Convolution layer

- Input (32 x 32) -> Output (3 x 28 x 28)
- $O[u][x][y] = \sum_{i=0}^{S-1} \sum_{j=0}^{R-1} I[x+i][y+j] * W[u][i][j] + B[u]$
- $0 \leq u < M, 0 \leq x < F, 0 \leq y < E$
- $W = F + S - 1, H = E + R - 1$



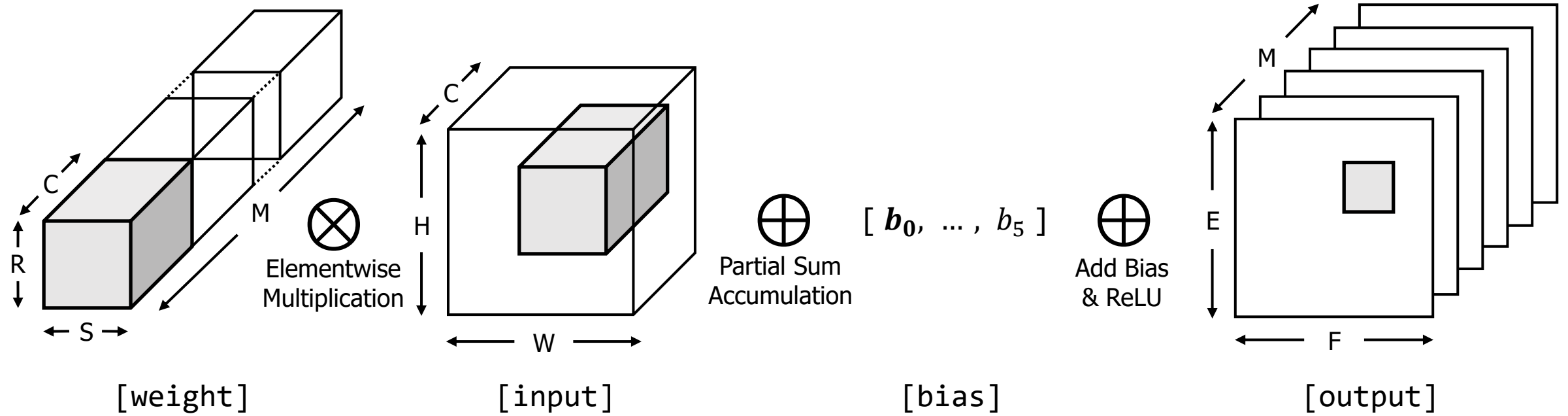
# 1<sup>st</sup> Pooling layer

- Input (3 x 28 x 28) -> Output (3 x 14 x 14)
- 2x2 Max pooling, stride 2



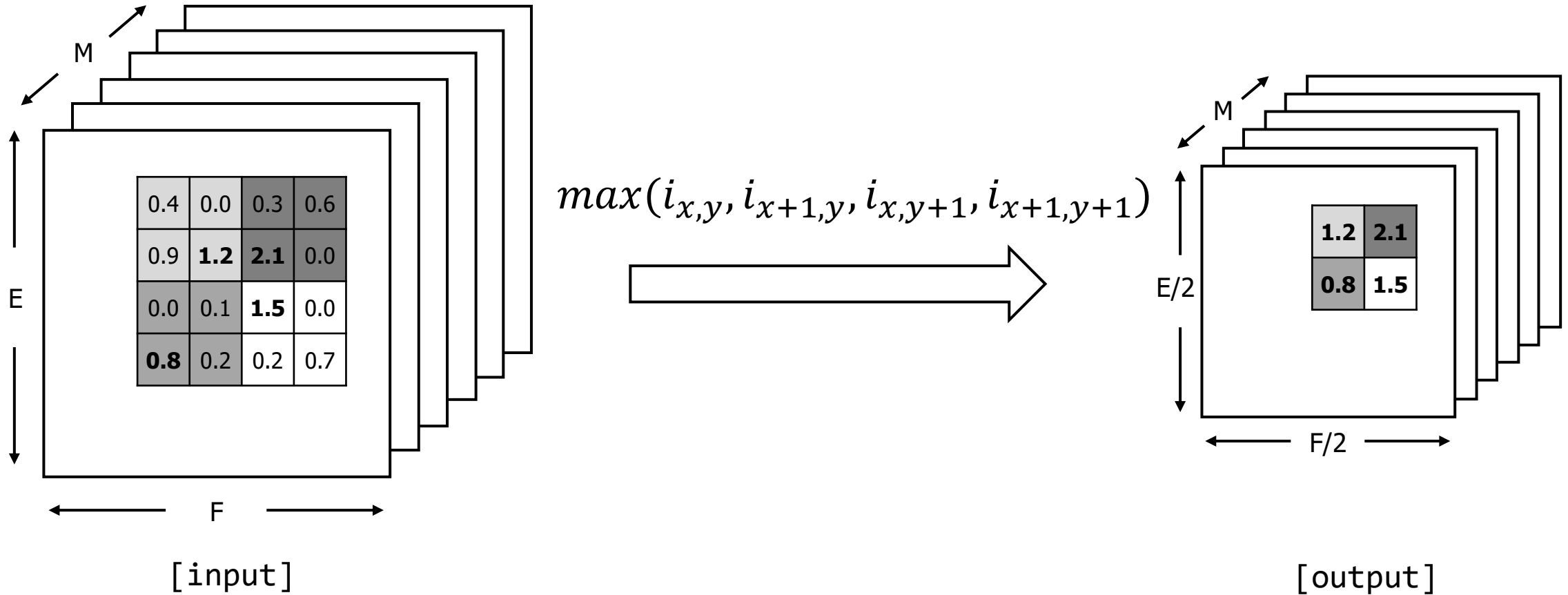
# 2<sup>nd</sup> Convolution layer

- Input (3 x 14 x 14) -> Output (6 x 10 x 10)
- $O[u][x][y] = \sum_{k=0}^{C-1} \sum_{i=0}^{S-1} \sum_{j=0}^{R-1} I[k][x+i][y+j] * W[u][k][i][j] + B[u]$
- $0 \leq u < M, 0 \leq x < F, 0 \leq y < E$
- $W = F + S - 1, H = E + R - 1$



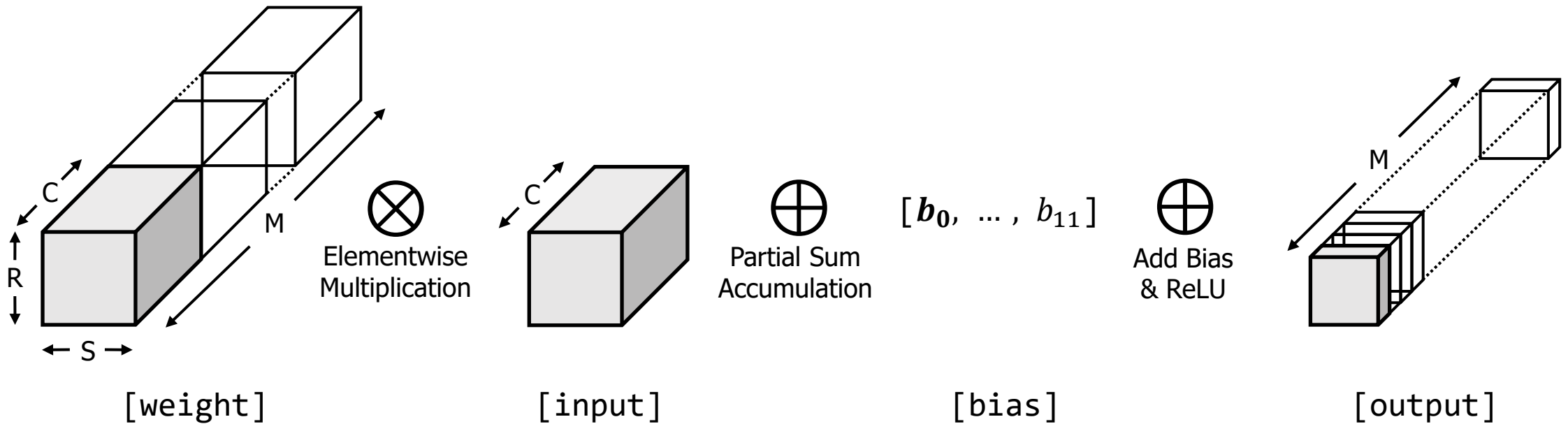
## 2<sup>nd</sup> Pooling layer

- Input (6 x 10 x 10) -> Output (6 x 5 x 5)
- 2x2 Max pooling, stride 2



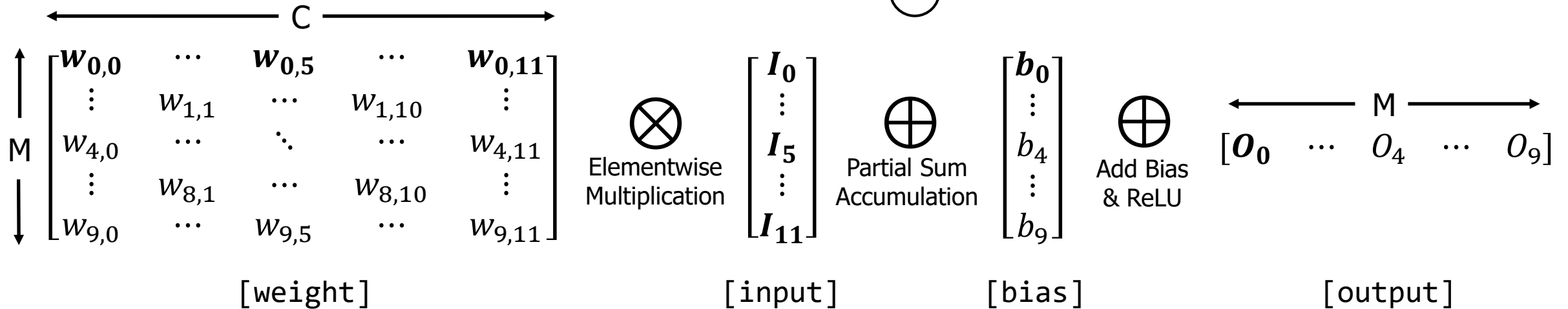
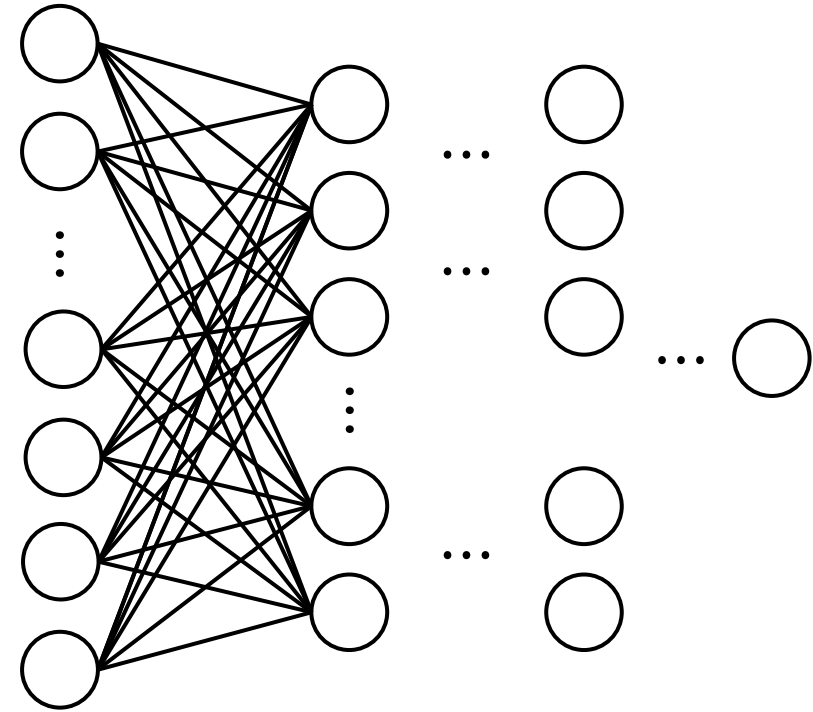
# 3<sup>rd</sup> Convolution layer

- Input (6 x 5 x 5) -> Output (12 x 1 x 1)
- $O[u] = \sum_{k=0}^{C-1} \sum_{i=0}^{S-1} \sum_{j=0}^{R-1} I[k][i][j] * W[u][k][i][j] + B[u]$
- $0 \leq u < M$
- $W = S, H = R, E = F = 1$



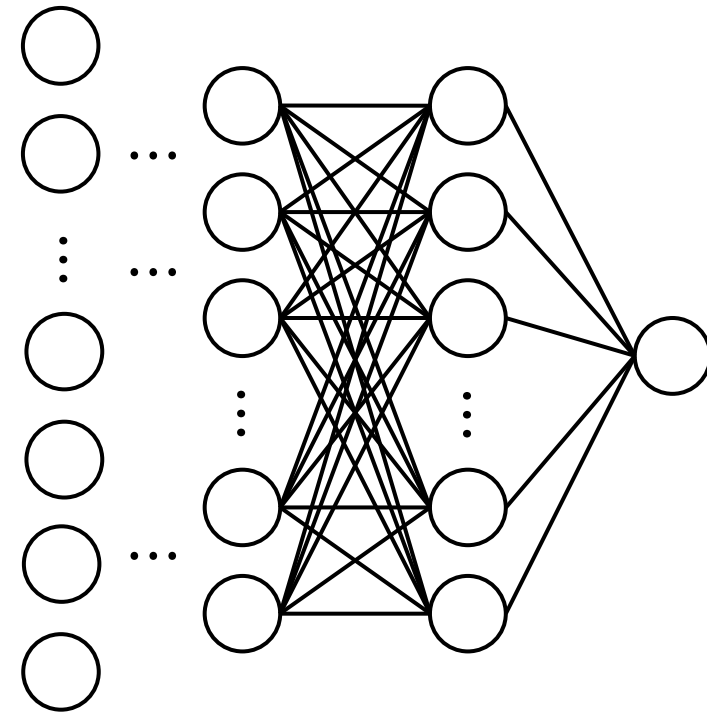
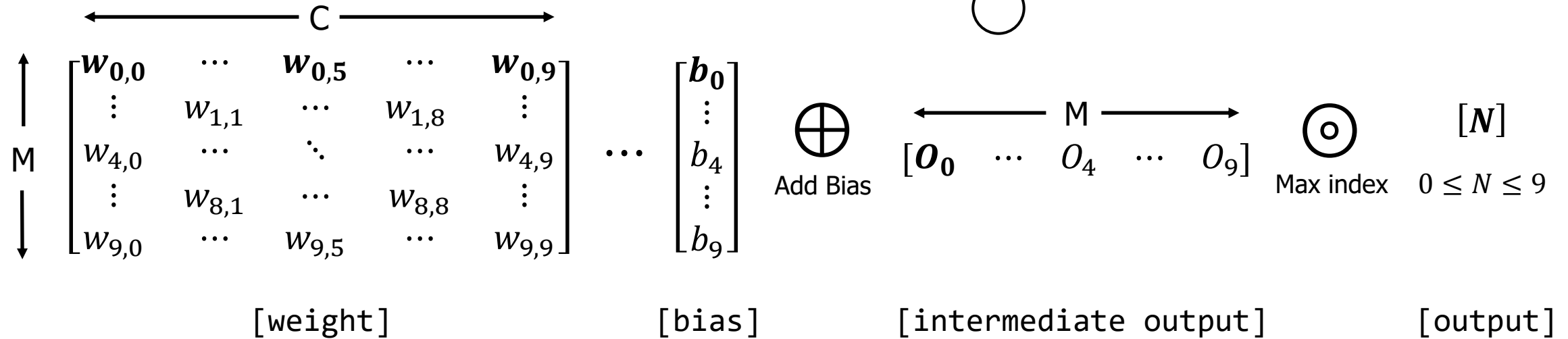
# 1<sup>st</sup> Fully Connected layer

- Input (12) -> Output (10)
- $O[u] = \sum_{k=0}^{C-1} I[k] * W[u][k] + B[u]$
- $0 \leq u < M$



# 2<sup>nd</sup> Fully Connected layer

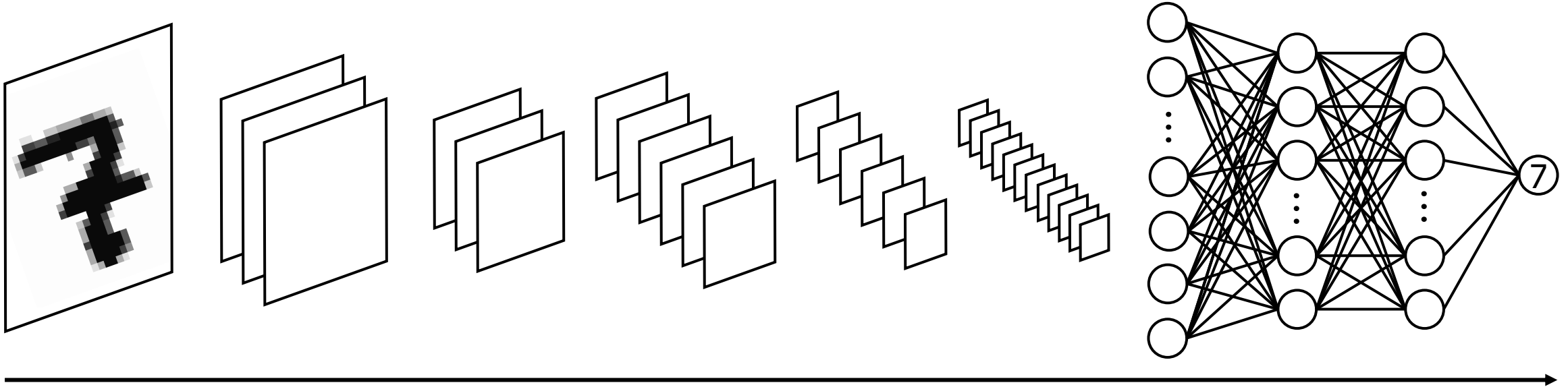
- Input (10) -> Output (10)
- $O[u] = \sum_{k=0}^{C-1} I[k] * W[u][k] + B[u]$
- $0 \leq u < M$
- Infer a number by finding index of max value.





# Overview of Optimized LeNet-5 (Light LeNet-5)

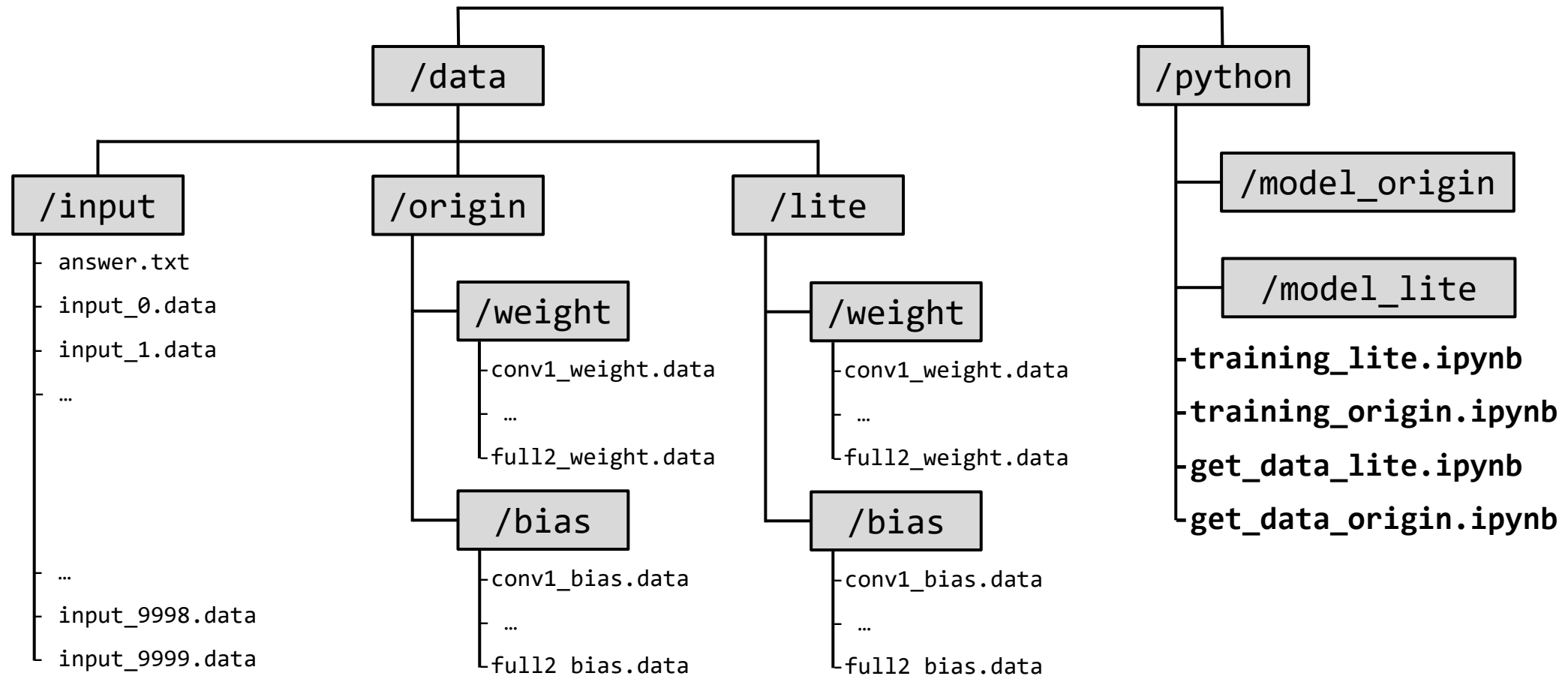
- Feature maps were reduced by **48%**, weights and biases were reduced by **96%**.
- *sigmoid*, *tanh* are replaced by *ReLU*.



	INPUT	CONV1	POOL1	CONV2	POOL2	CONV3	FULL1	FULL2	OUTPUT
Feature Map	32 x 32	3 @ 28 x 28	3 @ 14 x 14	6 @ 10 x 10	6 @ 5 x 5	12 @ 1 x 1	10	10	1
Weight + Bias	-	78	-	456	-	1812	130	110	-
Activation	-	<i>ReLU</i>	-	<i>ReLU</i>	-	<i>ReLU</i>	<i>ReLU</i>	<i>Max</i>	-

# Implementation of LeNet-5 using Python, Cont'd

- TensorFlow is used to build LeNet-5.
- Get data such as pre-trained weights and biases.
- Get data such as inputs and answer.



# Implementation of LeNet-5 using Python, Cont'd

- In `training_origin.ipynb`, build, train, and evaluate the original model.

## # Build model using Sequential API

```
import tensorflow as tf
```

```
...
```

```
model = models.Sequential()
```

```
model.add(layers.Conv2D(6, 5, 'tanh'))
```

```
model.add(layers.AveragePooling2D(2))
```

```
model.add(layers.Activation('sigmoid'))
```

```
model.add(layers.Conv2D(16, 5, 'tanh'))
```

```
model.add(layers.AveragePooling2D(2))
```

```
model.add(layers.Activation('sigmoid'))
```

```
model.add(layers.Conv2D(120, 16, 'tanh'))
```

```
model.add(layers.Flatten())
```

```
model.add(layers.Dense(10, 'tanh'))
```

```
model.add(layers.Dense(10, 'softmax'))
```

## # Training model

```
model.compile( ... )
```

```
model.fit( ... )
```

## # Get loss value and accuracy

```
model.evaluate( ... )
```

```
200/200 [=====] - loss: 0.1888, accuracy: 0.9406
```

## # Print model information

```
model.summary()
```

Layer (type)	Output Shape	Param #
=====		
conv1 (Conv2D)	(None, 28, 28, 6)	156
pool1 (AveragePooling2D)	(None, 14, 14, 6)	0
activation (Activation)	(None, 14, 14, 6)	0
conv2 (Conv2D)	(None, 10, 10, 16)	2416
pool2 (AveragePooling2D)	(None, 5, 5, 16)	0
activation_1 (Activation)	(None, 5, 5, 16)	0
conv3 (Conv2D)	(None, 1, 1, 120)	48120
flatten (Flatten)	(None, 120)	0
full1 (Dense)	(None, 84)	10164
full2 (Dense)	(None, 10)	850
=====		
Total params: 61,706		
Trainable params: 61,706		
Non-trainable params: 0		

# Implementation of LeNet-5 using Python, Cont'd

- In `training_lite.ipynb`, build, train, and evaluate light model.

## # Build model using Sequential API

```
import tensorflow as tf
```

```
...
```

```
model = models.Sequential()
```

```
model.add(layers.Conv2D(3, 5, 'relu'))
```

```
model.add(layers.MaxPooling2D(2))
```

```
model.add(layers.Conv2D(6, 5, 'relu'))
```

```
model.add(layers.MaxPooling2D(2))
```

```
model.add(layers.Conv2D(12, 5, 'relu'))
```

```
model.add(layers.Flatten())
```

```
model.add(layers.Dense(10, 'relu'))
```

```
model.add(layers.Dense(10, 'softmax'))
```

## # Training model

```
model.compile( ... )
```

```
model.fit( ... )
```

## # Get loss value and accuracy

```
model.evaluate( ... )
```

```
200/200 [=====] - loss: 0.0773, accuracy: 0.9747
```

## # Print model information

```
model.summary()
```

Layer (type)	Output Shape	Param #
=====		
conv1 (Conv2D)	(None, 28, 28, 3)	78
pool1 (MaxPooling2D)	(None, 14, 14, 3)	0
conv2 (Conv2D)	(None, 10, 10, 6)	456
pool2 (MaxPooling2D)	(None, 5, 5, 6)	0
conv3 (Conv2D)	(None, 1, 1, 12)	1812
flatten (Flatten)	(None, 12)	0
full1 (Dense)	(None, 10)	130
full2 (Dense)	(None, 10)	110

```
=====
```

```
Total params: 2,586
```

```
Trainable params: 2,586
```

```
Non-trainable params: 0
```

```
=====
```

# Implementation of LeNet-5 using Python

- In `get_data_lite.ipynb`, get the weights and biases processed for ease of use.

## # Get weights and biases

```
weight_bias_dict = {}
layer_list = []

for layer in model.layers:
    layer_list.append(layer.name)

    # No trainable params
    if 'pool' in layer.name or 'flatten' in layer.name:
        continue

    weight_bias_dict[layer.name] = layer.get_weights()
```

## # Process weights to use easily in C/C++

```
for key, value in weight_bias_dict.items():
    if 'conv' in key:
        tp_weight = tf.transpose(value[0])
        weight = tf.transpose(tp_weight, [0,1,3,2])
        value[0] = tf.squeeze(weight)

    if 'full' in key:
        tp_weight = tf.transpose(value[0])
        value[0] = tf.squeeze(tp_weight)
```

## # Shape of weights and biases before processing

```
=====
name : conv1  weight shape : (5, 5, 1, 3)   bias shape : (3,)
name : conv2  weight shape : (5, 5, 3, 6)   bias shape : (6,)
name : conv3  weight shape : (5, 5, 6, 12)  bias shape : (12,)
name : full1  weight shape : (12, 10)       bias shape : (10,)
name : full2  weight shape : (10, 10)       bias shape : (10,)
=====
```

## # Shape of weights and biases after processing

```
=====
name : conv1  weight shape : (3, 5, 5)      bias shape : (3,)
name : conv2  weight shape : (6, 3, 5, 5)   bias shape : (6,)
name : conv3  weight shape : (12, 6, 5, 5)  bias shape : (12,)
name : full1  weight shape : (10, 12)       bias shape : (10,)
name : full2  weight shape : (10, 10)       bias shape : (10,)
=====
```

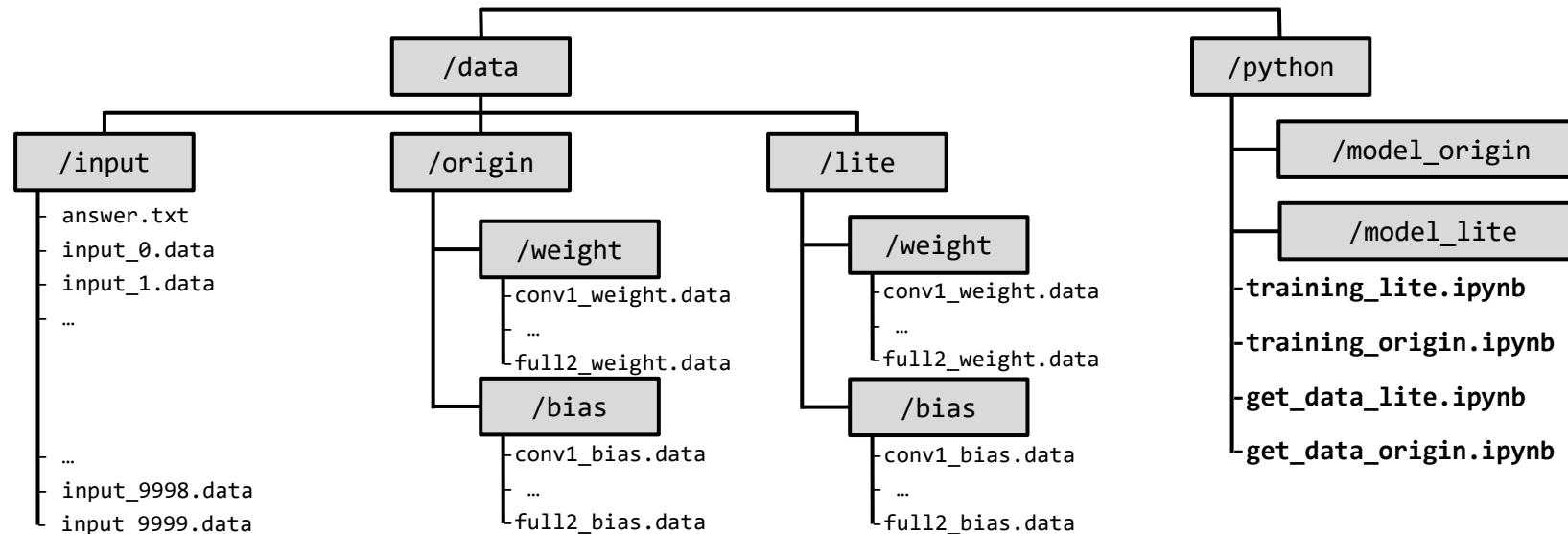
## # Flatten and save weights and biases

```
for key, value in weight_bias_dict.items():
    value[0] = tf.reshape(value[0], [-1]) # -1 means flatten

    # save as txt file
    np.savetxt('../data/weight/...' value[0], fmt = "%.10f")
    np.savetxt('../data/bias/...' value[1], fmt = "%.10f")
```

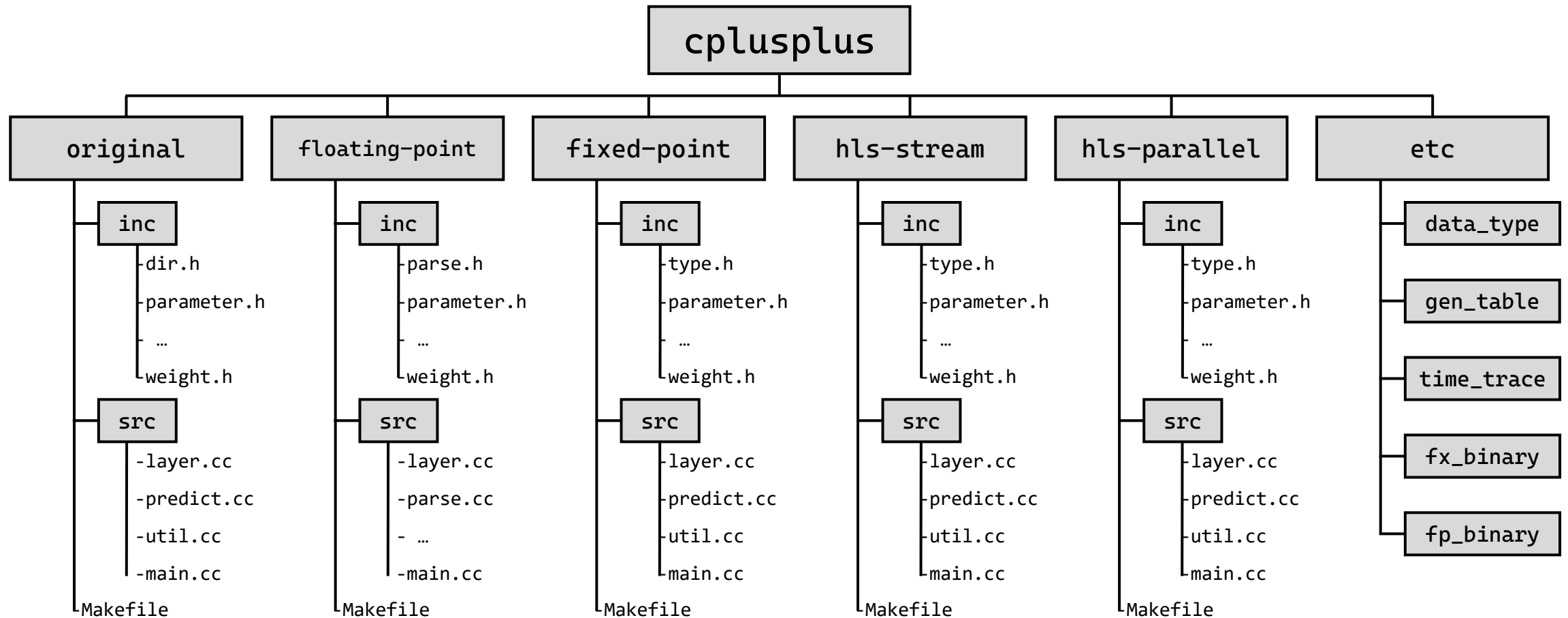
# Overview of implementation of LeNet-5 using python

- TensorFlow is used to build LeNet-5.
  - In `training_origin.ipynb`, build, train, and evaluate the original model.
  - In `training_lite.ipynb`, build, train, and evaluate light model.
- Get data such as pre-trained weights and biases.
  - In `get_data_lite.ipynb`, get the weights and biases processed for ease of use.
- Get data such as inputs and answer.
  - In `get_data_lite.ipynb`, get the inputs and answer.



# Implementation of LeNet-5 using C/C++

- cplusplus directory structure.
- Makefile can compile each folder.
- etc directory serves to analyze the result, save input txt file as a binary, or generate weight table.



# Fixed-Point vs. Floating-Point, Cont'd

- Comparison by data type using *Vivado HLS*.

```
/*
  Tests to determine latency, utilization,
  and max frequency by data type.
*/

#define NUM 32

typedef int data_t;
...
// typedef ap_fixed<32, 16> data_t;
...
// typedef half data_t;

void test(data_t in[NUM], ... , data_t out[NUM])
{
    for (int i = 0; i < NUM; i++) {

        data_t result = bias[i];

        for (int j = 0; j < NUM; j++) {
            result += in[j] * weight[i][j];
        }
        out[i] = result;
    }
}
```

[ Latency (clock cycles) ]

	short	fix16	half	int	fix32	float
Latency	3169	3169	11361	4193	4193	11361

[ Resource Usage Implementaion ]

	short	fixed16	half	int	fixed32	float
LUT	21	38	166	68	100	330
FF	40	56	280	90	92	460
DSP	1	1	4	3	4	5
SLICE	10	14	96	26	36	184

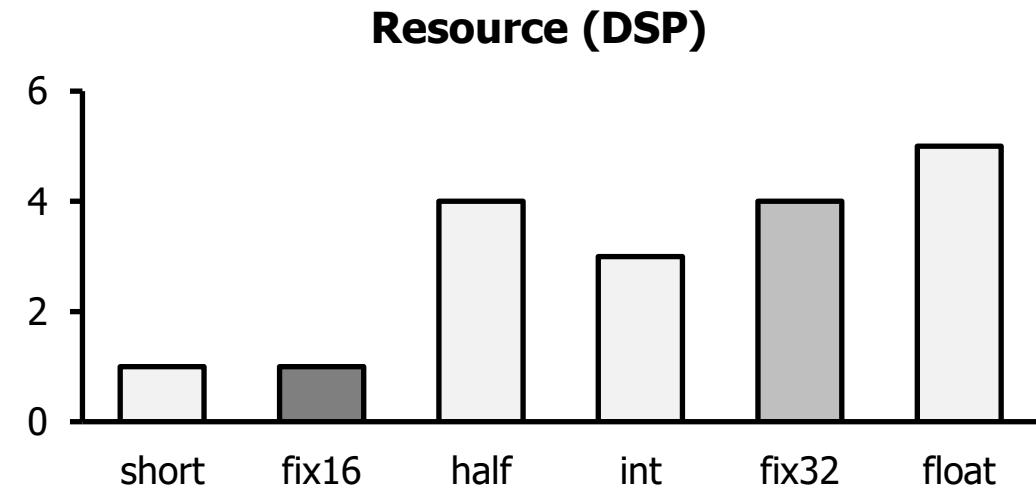
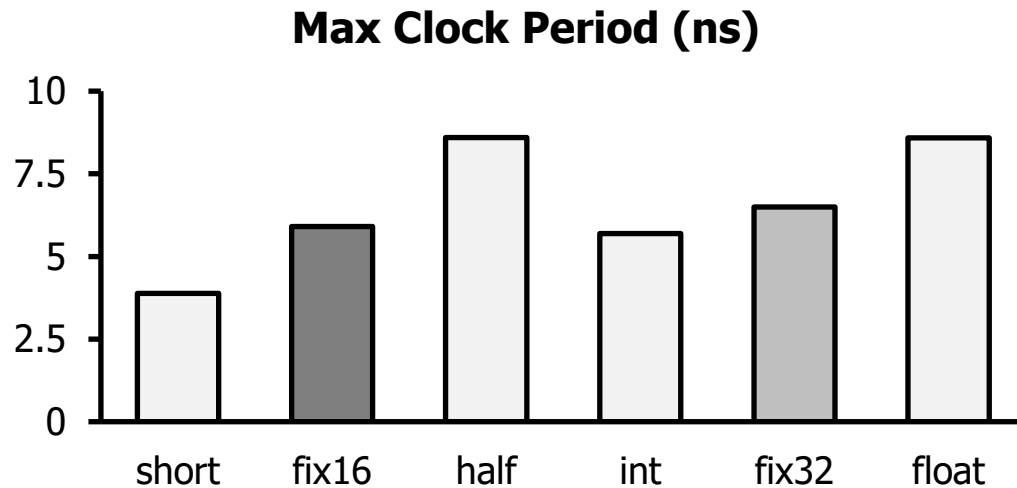
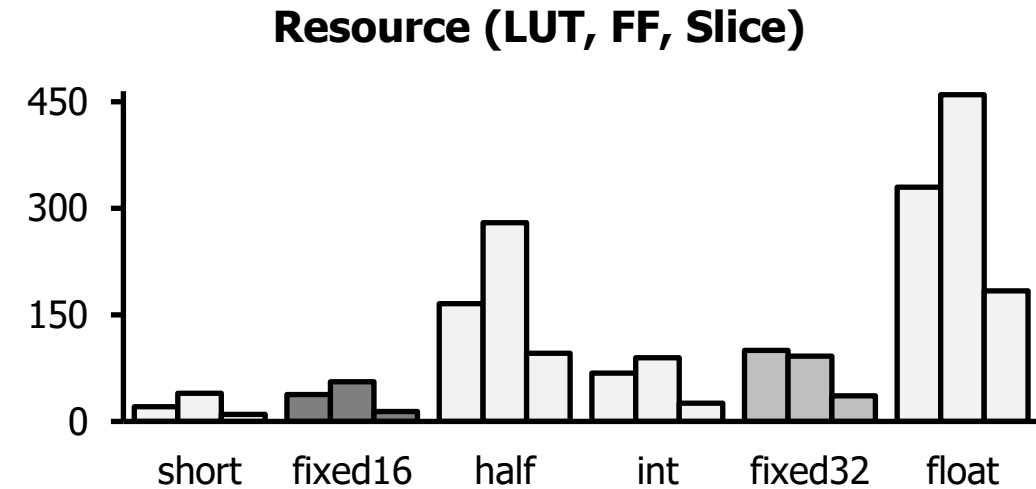
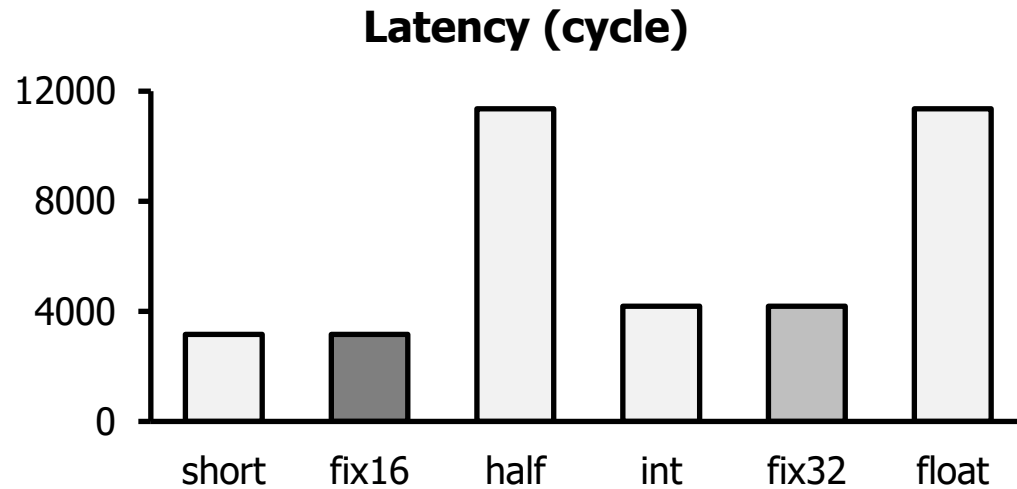
[ Final Timing Implementation ]

	short	fix16	half	int	fix32	float
CP Req.	10.000	10.000	10.000	10.000	10.000	10.000
Post-Impl	3.884	5.911	8.598	5.692	6.501	8.585



# Fixed-Point vs. Floating-Point

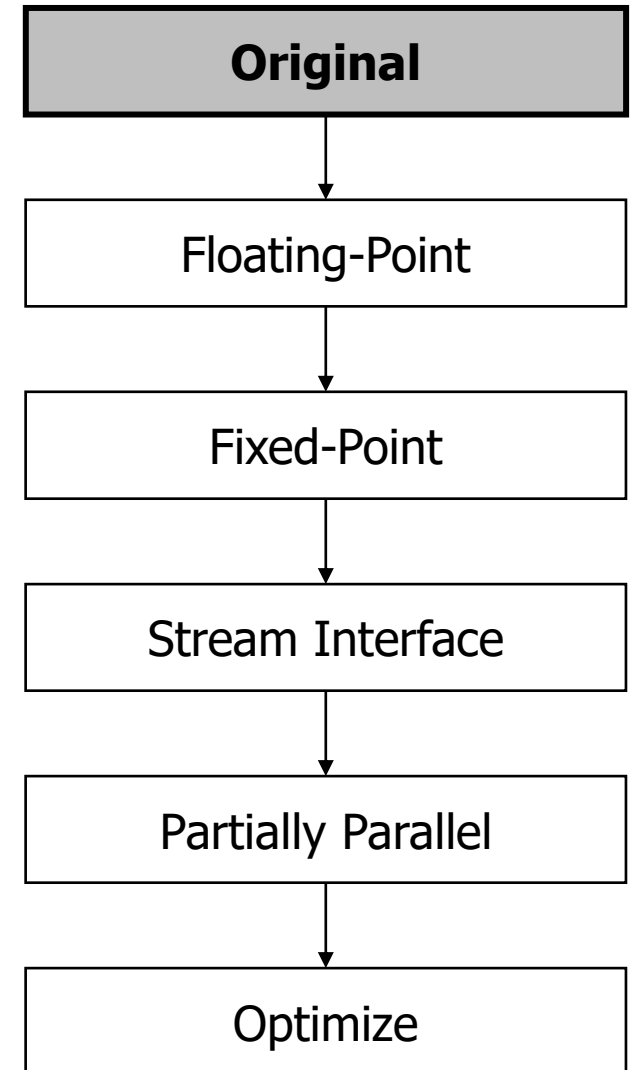
- Results of synthesis and implementation using *Vivado HLS*.



# Original LeNet-5 with C/C++, Cont'd

- Implementation of original LeNet-5.
- Check accuracy to ensure that it is implemented correctly.
- Use *tanh* or *sigmoid* instead of *ReLU* for activation function.
- Use Average Pooling instead of Max Pooling.

Layer	C	W x H	Kernel	Stride	Activation	Weight + Bias	Feature map
INPUT	1	32 x 32	-	-	-	-	1024
CONV1	6	28 x 28	5x5	1	<i>tanh</i>	$150 + 6 = 156$	4704
A.POOL1	6	14 x 14	2x2	2	<i>sigmoid</i>	-	1176
CONV2	16	10 x 10	5x5	1	<i>tanh</i>	$2400 + 16 = 2416$	1600
A.POOL2	16	5 x 5	2x2	2	<i>sigmoid</i>	-	400
CONV3	120	1 x 1	5x5	1	<i>tanh</i>	$48000 + 120 = 48120$	120
FC1	-	120	-	-	<i>tanh</i>	$10080 + 84 = 10164$	84
FC2	-	84	-	-	<i>softmax</i>	$840 + 10 = 850$	10
<b>Total</b>	-	-	-	-	-	<b>61706</b>	<b>9118</b>



# Original LeNet-5 with C/C++, Cont'd

```
/* layer.cc */
...
void full1_layer(float input[...][...][...],
                float output[...][...][...])
{
    for (int row = 0; row < ... ; ...) {

        float acc = full1_bias[row];

        for (int col = 0; col < ... ; ...) {
            acc += input[col] * full1_weight[row][col];
        }

        output[row] = (float)std::tanh(acc); // tanh activation
    }
}

void pool1_layer(float input[...][...][...],
                float output[...][...][...])
{
    for (int num = 0; num < ... ; ...) {
        for (int row = 0; row < ... ; ...) {
            for (int col = 0; col < ... ; ...) {
                float avg = 0.0;
                avg += input[num][row][col];
                ...
                avg = (float)(avg / 4.0);

                // sigmoid activation
                output[num][row >> 1][col >> 1] = sigmoid(avg);
            }
        }
    }
}
```

```
/* predict.cc */

uint8_t predict(float* input) {

    /* Intermediate outputs */
    float conv1_output[...][...][...] = { 0.0 };
    ...
    float full2_output[...] = { 0.0 };

    /* 1st convolution layer */
    conv1_layer(input, conv1_output);

    /* 1st pooling layer */
    pool1_layer(conv1_output, pool1_output);

    /* 2nd convolution layer */
    conv2_layer(pool1_output, conv2_output);

    /* 2nd pooling layer */
    pool2_layer(conv2_output, pool2_output);

    /* 3th convolution layer */
    conv3_layer(pool2_output, conv3_output);

    /* 1st fully connected layer */
    full1_layer(conv3_output, full1_output);

    /* 2nd fully connected layer */
    full2_layer(full1_output, full2_output);

    /* Return result */
    return softmax(full2_output);
}
```

# Original LeNet-5 with C/C++

- Similar to the accuracy obtained from python (0.9406)

```
/* main.cc */

int main (int argc, char* argv[]) {
    int err_cnt = 0;
    ...
    for (size_t i = 0; i < TEST_NUM; i++) {
        ...
        my_answer[i] = predict(input_array);
        ...
        if (my_answer != answer[i]) {
            err_cnt++;
            cout << "[ FALSE ] " ... << "\n";
        }
        else {
            cout << "[ TRUE ] " ... << "\n";
        }
    }

    accuracy = ((float)(TEST_NUM - err_cnt)/(float)TEST_NUM) * 100;
    cout << "\n[Original LeNet-5]\n";
    cout << "Accuracy : " << setw(5) << accuracy << " ";
    cout << "Error      : " << setw(5) << err_cnt << " ";

    return 0;
}
```

# Run all inputs

\$ ./main

	Number	Answer	Predict
[TRUE] [ 0]	7	7	
[TRUE] [ 1]	2	2	
[TRUE] [ 2]	1	1	
[TRUE] [ 3]	0	0	
...			
[TRUE] [9996]	3	3	
[TRUE] [9997]	4	4	
[TRUE] [9998]	5	5	
[TRUE] [9999]	6	6	

**[Original LeNet-5]**

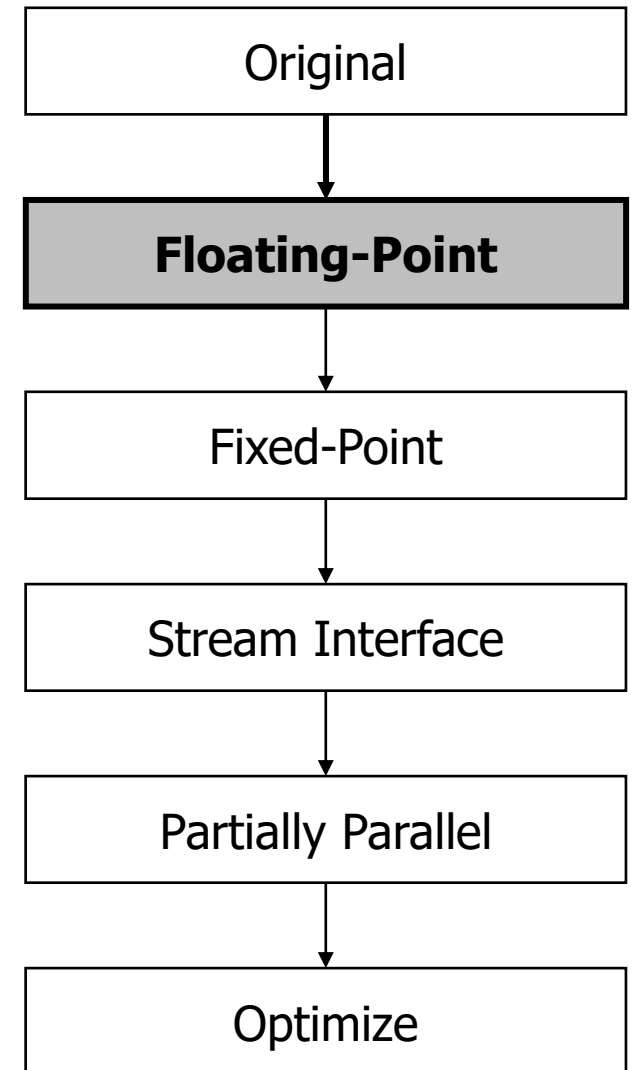
Accuracy : **94.55** [%]

Error : 545 [cases]

# Light LeNet-5 with C/C++ (Floating-Point), Cont'd

- Implementation of light LeNet-5 using floating point.
- Use *ReLU* instead of *sigmoid* or *tanh* for activation function.
- Use Max Pooling instead of Average Pooling.
- Use *max* instead of *softmax* for output layer.

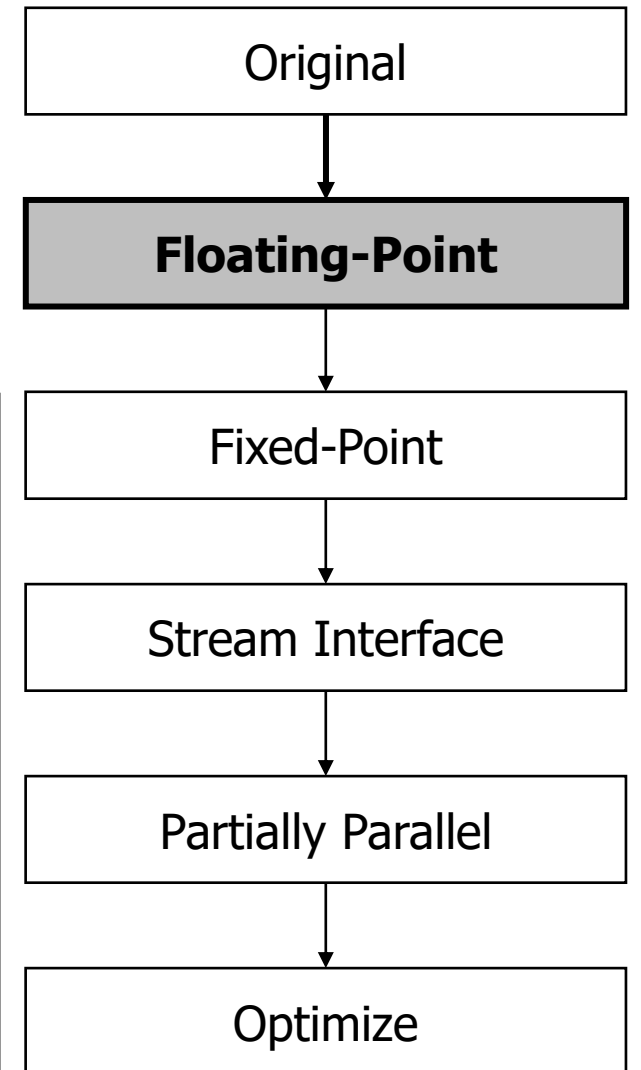
Layer	C	W x H	Kernel	Stride	Activation	Weight + Bias	Feature map
INPUT	1	32 x 32	-	-	-	-	1024
CONV1	3	28 x 28	5x5	1	<i>ReLU</i>	$75 + 3 = 78$	2352
M.POOL1	3	14 x 14	2x2	2	-	-	588
CONV2	6	10 x 10	5x5	1	<i>ReLU</i>	$450 + 6 = 456$	600
M.POOL2	6	5 x 5	2x2	2	-	-	150
CONV3	12	1 x 1	5x5	1	<i>ReLU</i>	$1800 + 12 = 1812$	12
FC1	-	120	-	-	<i>ReLU</i>	$120 + 10 = 130$	10
FC2	-	84	-	-	<i>Max</i>	$100 + 10 = 110$	10
<b>Total</b>	-	-	-	-	-	<b>2586</b>	<b>4746</b>



# Light LeNet-5 with C/C++ (Floating-Point), Cont'd

- Enable to run with 3 options.
  - Run one input.
  - Run one input + Print intermediate results (Debug).
  - Run all (Check accuracy and find Max/Min value).
- Integer part of fixed point can be determined through Run all option.

<pre># How to use  \$ ./main --help  [Run one input] Usage : ./main --input=[input.data]         ./main -i=[input.data]  [Run one input with print option] Usage : ./main --input=[input.data] --print=[bool]         ./main -i=[input.data] -p=[bool]  [Run all inputs] Usage : ./main --all=[bool]         ./main -a=[bool]  ...</pre>	<pre># Run all inputs  \$ ./main --all=true  ...  # To determine the integer part of fixed point [CONV1] output  MAX :  4.7115  MIN : - 2.9715 [CONV2] output  MAX : 12.2852  MIN : -20.4851 [CONV3] output  MAX : 36.0140  MIN : -33.7252 [FULL1] output  MAX : 33.5324  MIN : -14.5665 [FULL2] output  MAX : 27.2409  MIN : -30.0438</pre>
--	--



# Light LeNet-5 with C/C++ (Floating-Point), Cont'd

```
/* layer.cc */
...
void full1_layer(float input[...], float output[...])
{
    for (int row = 0; row < ... ; ...) {

        float acc = full1_bias[row];

        for (int col = 0; col < ... ; ...) {
            acc += input[col] * full1_weight[row][col];
        }
    }
}
... // ReLU function exists to find the MAX/MIN value
void relu4_layer(float input[...], float output[...])
{
    for (int i = 0; i < ...; ...)
        output[i] = input[i] > 0.0 ? input[i] : 0.0;
}
...
void pool1_layer(float input[...][...][...], float
output[...][...][...])
{
    for (int num = 0; num < ... ; ...) {
        for (int row = 0; row < ... ; ...) {
            for (int col = 0; col < ... ; ...) {

                max = input[num][row][col];
                ... // find max
                output[num][row >> 1][col >> 1] = max;

            }
        }
    }
}
```

```
/* predict.cc */

uint8_t predict(float* input, float* conv1_max, float* conv1_min,
                ...
                float* full2_max, float* full2_min) {

    /* Intermediate outputs */
    float conv1_output[...][...][...] = { 0.0 };
    float relu1_output[...][...][...] = { 0.0 };
    float pool1_output[...][...][...] = { 0.0 };
    ...
    float full2_output[...] = { 0.0 };

    /* 1st convolution layer */
    conv1_layer(input, conv1_output);

    /* 1st ReLU layer */
    relu1_layer(conv1_output, relu1_output);
    ...
    /* 2nd fully connected layer */
    full2_layer(relu4_output, full2_output);

    /* Find the MAX/MIN value of each intermediate result.
    *conv1_max = *std::max_element(conv1_output, ...);
    ...
    *full2_max = *std::max_element(full2_output, ...);
    *conv1_min = *std::min_element(conv1_output, ...);
    ...
    *full2_min = *std::min_element(full2_output, ...);

    /* Return result */
    return result(full2_output);
}
```

# Light LeNet-5 with C/C++ (Floating-Point), Cont'd

- Enble to run with 3 options. (Run all option)

## # How to use

```
$ ./main --help
```

### [Run one input]

```
Usage : ./main --input=[input.data]
        ./main -i=[input.data]
```

### [Run one input with print option]

```
Usage : ./main --input=[input.data] --print=[bool]
        ./main -i=[input.data] -p=[bool]
```

### [Run all inputs]

```
Usage : ./main --all=[bool]
        ./main -a=[bool]
```

```
--input=[input] | -i=[input]  File name to run.
                               [input] must be txt file.
                               [input] extension must be *.data.
                               This option is required.
```

```
--print=[bool] | -p=[bool]    Determine whether to print intermediate result.
                               [bool] must be true or false with small letters.
                               This option is optional.
```

```
--all=[bool] | -a=[bool]      Determine whether run all test cases.
                               [bool] must be true or false with small letters.
                               This option is optional.
```

```
--help | -h                  Display this information.
```

## # Run all inputs

```
$ ./main --all=true
```

	Number	Answer	Predict
[TRUE] [ 0]	0	7	7
[TRUE] [ 1]	1	2	2
[TRUE] [ 2]	2	1	1
[TRUE] [ 3]	3	0	0
...			
[TRUE] [9996]	9996	3	3
[TRUE] [9997]	9997	4	4
[TRUE] [9998]	9998	5	5
[TRUE] [9999]	9999	6	6

### [Floating-Point]

```
Accuracy : 97.47 [%]
```

```
Error : 253 [cases]
```

### # To determine the integer part of fixed point

[CONV1] output	MAX : 4.7115	MIN : - 2.9715
[CONV2] output	MAX : 12.2852	MIN : -20.4851
[CONV3] output	MAX : 36.0140	MIN : -33.7252
[FULL1] output	MAX : 33.5324	MIN : -14.5665
[FULL2] output	MAX : 27.2409	MIN : -30.0438



# Light LeNet-5 with C/C++ (Floating-Point)

- Enble to run with 3 options. (Run one input, Run one input + Print intermediate results options)

```
# Run one input
```

```
$ ./main --input=../../data/input/input_0.data
```

```
=====
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.33 0.73 0.62 0.59 0.24 0.14 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.87 1.00 1.00 1.00 1.00 0.95 0.78 0.78 0.78 0.78 0.78 0.78 0.78 0.67 0.20 0.00 0.00 0.00
0.00 0.00 0.26 0.45 0.28 0.45 0.64 0.89 1.00 0.88 1.00 1.00 1.00 0.98 0.90 1.00 1.00 0.55 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.07 0.26 0.05 0.26 0.26 0.26 0.23 0.08 0.93 1.00 0.42 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.33 0.99 0.82 0.07 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.09 0.91 1.00 0.33 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.51 1.00 0.93 0.17 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.23 0.98 1.00 0.24 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.52 1.00 0.73 0.02 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.04 0.80 0.97 0.23 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.49 1.00 0.71 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.29 0.98 0.94 0.22 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.07 0.87 1.00 0.65 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.80 1.00 0.86 0.14 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.15 1.00 1.00 0.30 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.12 0.88 1.00 0.45 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.52 1.00 1.00 0.20 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.24 0.95 1.00 0.20 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.47 1.00 1.00 0.86 0.16 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.47 1.00 0.81 0.07 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
=====
```

```
Number : 7
```

```
# Run one input + print intermediate results
```

```
$ ./main -i=../../data/input/input_0.data --print=true
```

```
[CONV1 output] [3][28][28]
```

```
...
```

```
[POOL1 output] [3][14][14]
```

```
...
```

```
[CONV2 output] [6][10][10]
```

```
...
```

```
[POOL2 output] [6][5][5]
```

```
...
```

```
1.72 0.52 0.00 0.00 0.00 0.22 0.41 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
1.64 1.93 2.15 3.01 3.50 1.82 0.00 0.00 0.00 0.00
3.83 4.38 5.63 7.22 7.63 3.85 0.00 0.00 0.00 1.00
0.96 1.25 1.73 2.39 0.96 0.00 0.00 0.00 0.00 2.06
0.09 0.24 0.74 0.92 0.00 0.00 0.00 0.00 0.10 3.00
0.18 0.19 0.50 0.00 0.00 0.00 0.00 0.00 1.91 2.51
0.29 1.23 0.60 0.00 0.00 0.00 0.00 0.53 2.99 1.52
0.00 0.00 0.00 0.00 0.00 0.00 0.00 2.58 2.39 0.51
```

```
...
```

```
[CONV3 output] [12]
```

```
0.00 4.09 4.29 1.92 10.52 10.28 8.23 0.00 0.00 2.30 0.00 0.00
```

```
[FULL1 output] [10]
```

```
8.94 6.80 8.77 0.00 3.41 9.39 0.00 0.00 0.00 0.00
```

```
[FULL2 output] [10]
```

```
-3.29 -0.61 3.09 2.74 -5.36 -2.09 -13.36 11.65 2.13 -0.26
```

```
Number : 7
```

# Light LeNet-5 with C/C++ (Fixed-Point), Cont'd

- Compare floating-point and fixed-point in terms of :
  - Latency
  - Resource
  - Max frequency

[ Latency (clock cycles) ]

	short	fix16	half	int	fix32	float
Latency	3169	3169	11361	4193	4193	11361

[ Resouce Usage Implementaion ]

	short	fixed16	half	int	fixed32	float
LUT	21	38	166	68	100	330
FF	40	56	280	90	92	460
DSP	1	1	4	3	4	5
SLICE	10	14	96	26	36	184

[ Final Timing Implementation ]

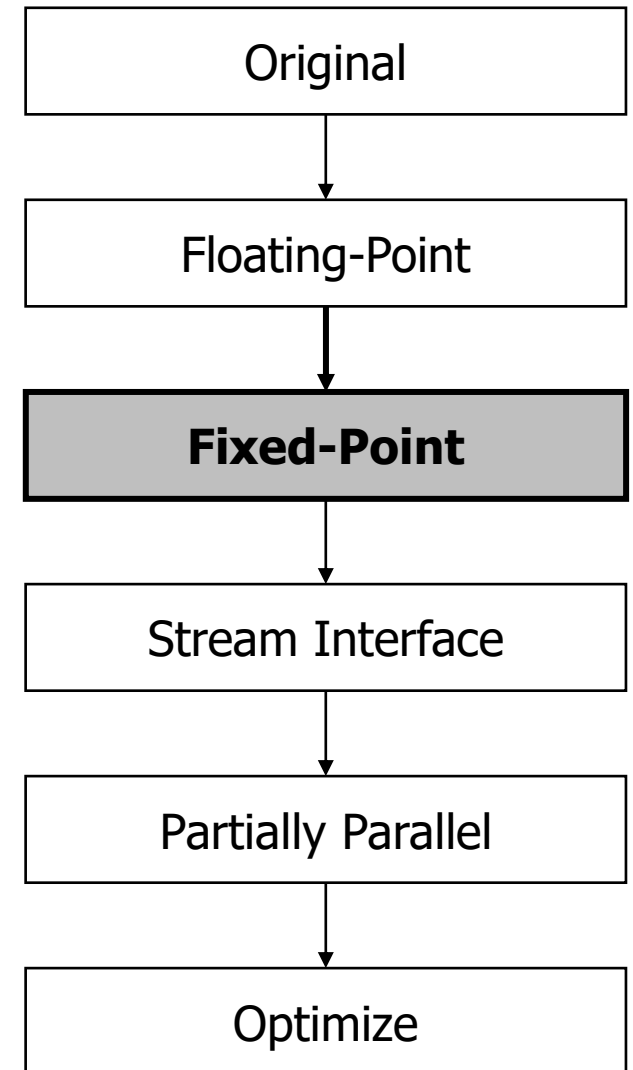
	short	fix16	half	int	fix32	float
CP Req.	10.000	10.000	10.000	10.000	10.000	10.000
Post-Impl	3.884	5.911	8.598	5.692	6.501	8.585

[ ZYBO Z7-10 Resource ]

	ZYBO Z7-10
LUT	17,600
Flip-Flop	35,200
Block RAM	270 KB
DSP	80
Logic slice	4,400

[ ZYBO Z7-20 Resource ]

	ZYBO Z7-20
LUT	53,200
Flip-Flop	106,400
Block RAM	630 KB
DSP	220
Logic slice	13,300



# Light LeNet-5 with C/C++ (Fixed-Point), Cont'd

- Determine integer part of fixed point.
  - Using `etc/data_type/`
- Include Xilinx HLS library to use `<ap_fixed.h>`
- For the *Round, Convert* formulas,
- Refer to the paper "*Deep Learning with Limited Numerical Precision*"

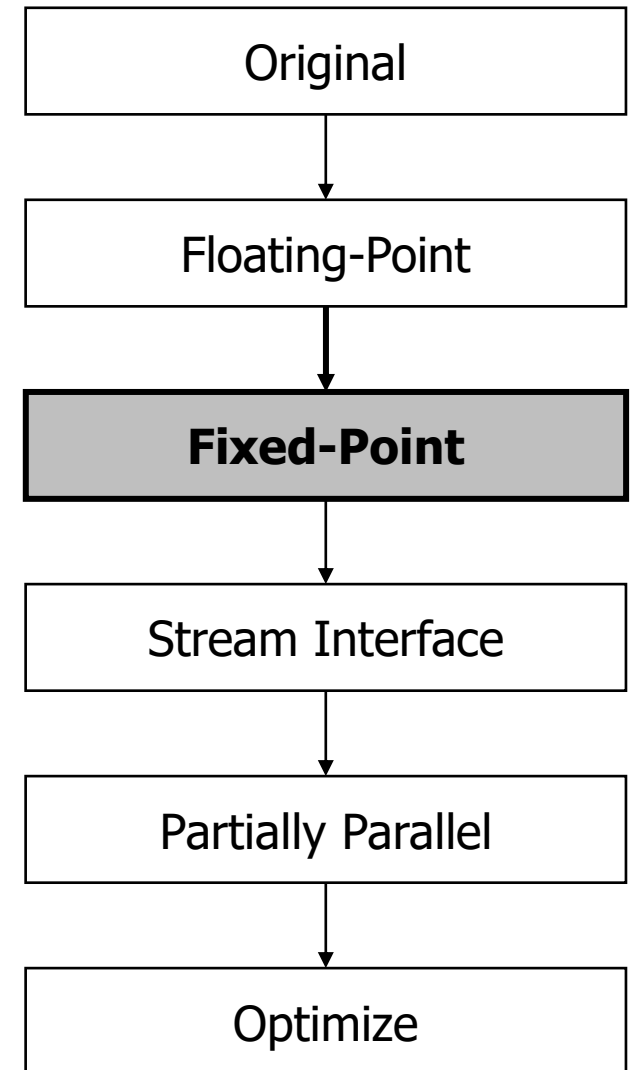
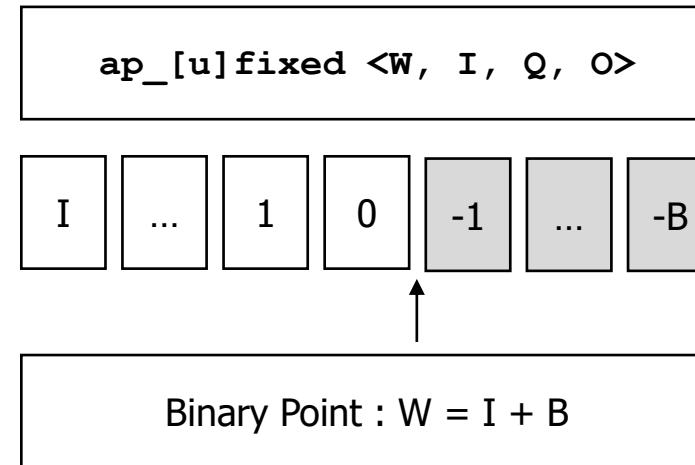
[ Round & Saturation formula]

$$\text{Round}(x, \langle I, F \rangle) = \begin{cases} [x] & \text{if } [x] \leq x \leq [x] + \frac{\epsilon}{2} \\ [x] + \epsilon & \text{if } [x] + \frac{\epsilon}{2} < x \leq [x] + \epsilon \end{cases}$$

where  $\epsilon = 2^{-F}$

$$\text{Convert}(x, \langle I, F \rangle) = \begin{cases} -2^{I-1} & \text{if } x \leq -2^{I-1} \\ 2^{I-1} - 2^{-F} & \text{if } x \geq 2^{I-1} - 2^{-F} \\ \text{Round}(x, \langle I, F \rangle) & \text{otherwise} \end{cases}$$

[ Fixed point data type ]



# Light LeNet-5 with C/C++ (Fixed-Point), Cont'd

- Determine integer part of fixed point using etc/data\_type directory.

```
/* data_type.cc */
```

```
// Data from ./main --all=true command output
```

```
#define CONV1_MAX    4.71154
#define CONV1_MIN    -2.97152
#define CONV2_MAX    12.2852
#define CONV2_MIN    -20.4851
#define CONV3_MAX    36.0140
#define CONV3_MIN    -33.7252
#define FULL1_MAX    33.5324
#define FULL1_MIN    -14.5665
#define FULL2_MAX    27.2409
#define FULL2_MIN    -30.0438
```

$$QI = \text{ceiling}(\log_2(\max(\text{abs}[\alpha_{\text{max}}, \alpha_{\text{min}}]) + 1)) + 1$$

```
// Implement the above formula as a function
```

```
int fx_integer(float m, float n) {
    int QI;

    QI = ceil(log2(max(abs(m), abs(n)) + 1)) + 1;

    return QI;
}
```

```
/* data_type.cc */
```

```
conv1_o = fx_integer(CONV1_MAX, CONV1_MIN);
...
conv3_w = fx_integer(max_conv3_weight, min_conv3_weight);
...
full2_b = fx_integer(max_full2_bias, min_full2_bias);
```

```
# Run data_type
```

```
$ ./data_type
```

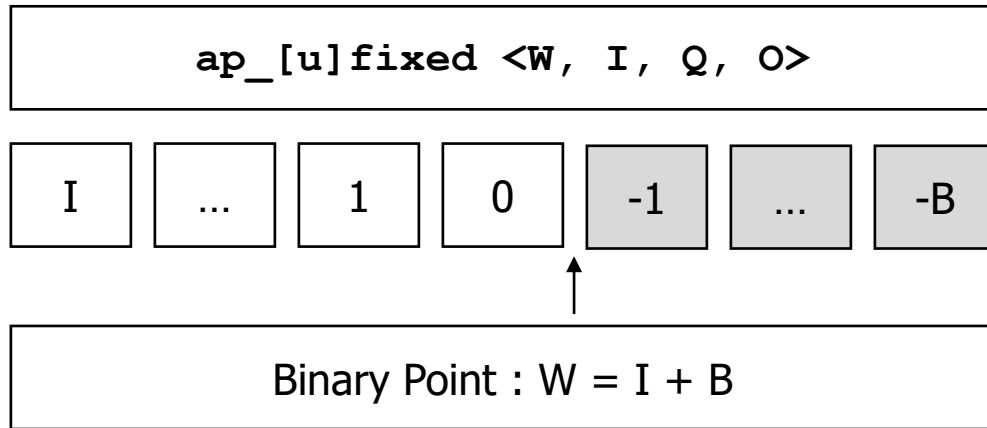
```
[CONV1_OUTPUT] integer part : 4
[CONV2_OUTPUT] integer part : 6
[CONV3_OUTPUT] integer part : 7
[FULL1_OUTPUT] integer part : 7
[FULL2_OUTPUT] integer part : 6
```

```
[CONV1_WEIGHT] integer part : 2
[CONV2_WEIGHT] integer part : 2
[CONV3_WEIGHT] integer part : 2
[FULL1_WEIGHT] integer part : 2
[FULL2_WEIGHT] integer part : 3
```

```
[CONV1_BIAS]    integer part : 2
...
[FULL2_BIAS]    integer part : 2
```

# Light LeNet-5 with C/C++ (Fixed-Point), Cont'd

- Refer to *Xilinx User Guide 902*
- Refer to *Deep Learning with Limited Numerical Precision*



$$\text{Round}(x, \langle I, F \rangle) = \begin{cases} [x] & \text{if } [x] \leq x \leq [x] + \frac{\epsilon}{2} \\ [x] + \epsilon & \text{if } [x] + \frac{\epsilon}{2} < x \leq [x] + \epsilon \end{cases}$$

where  $\epsilon = 2^{-F}$

$$\text{Convert}(x, \langle I, F \rangle) = \begin{cases} -2^{I-1} & \text{if } x \leq -2^{I-1} \\ 2^{I-1} - 2^{-F} & \text{if } x \geq 2^{I-1} - 2^{-F} \\ \text{Round}(x, \langle I, F \rangle) & \text{otherwise} \end{cases}$$

Identifier	Description	
W	Word length in bits	
I	The number of bits used to represent the integer value.	
Q	Quantization mode	
	Mode	Description
	AP_TRN	Truncation to minus infinity (default)
	...	...
	<b>AP_RND</b>	<b>Rounding to plus infinity</b>
O	Overflow mode	
	Mode	Description
	<b>AP_SAT</b>	<b>Saturation</b>
	...	...
	AP_WRAP	Wrap around (default)

# Light LeNet-5 with C/C++ (Fixed-Point), Cont'd

```
/* layer.cc */
...
void full1_layer(conv3_t input[...], full1_t output[...]) {

    for (int row = 0; row < ... ; ...) {
        // Temporary enough width accumulate register (32-bit)
        full1_temp acc = full1_bias[row];

        for (int col = 0; col < ... ; ...) {
            acc += input[col] * full1_weight[row][col];
        }
        if (acc > 0) output[row] = (full1_t)acc;
        else        output[row] = (full1_t)0.0;
    }
}

...
void pool1_layer(conv1_t input[...][...][...], conv1_t output[...][...][...]) {
    for (int num = 0; num < ... ; ...) {
        for (int row = 0; row < ... ; ...) {
            for (int col = 0; col < ... ; ...) {

                conv1_t n1 = input[num][row][col];
                ...
                conv1_t n4 = input[num][row+1][col+1];

                conv1_t max1 = std::max(n1, n2);
                conv1_t max2 = std::max(n3, n4);
                conv1_t max  = std::max(max1, max2);
                output[num][row >> 1][col >> 1] = max;
            }
        }
    }
}
```

```
/* predict.cc */

uint8_t predict(float* input) {

    /* Intermediate outputs */
    conv1_t conv1_output[...][...][...] = { 0.0 };
    conv1_t pool1_output[...][...][...] = { 0.0 };
    ...
    full2_t full2_output[...] = { 0.0 };

    /* 1st convolution layer */
    conv1_layer(input, conv1_output);

    /* 1st pooling layer */
    pool1_layer(conv1_output, pool1_output);

    /* 2nd convolution layer */
    conv2_layer(pool1_output, conv2_output);

    /* 2nd pooling layer */
    pool2_layer(conv2_output, pool2_output);

    /* 3rd convolution layer */
    conv3_layer(pool2_output, conv3_output);

    /* 1st fully connected layer */
    full1_layer(conv3_output, full1_output);

    /* 2nd fully connected layer */
    full2_layer(full1_output, full2_output);

    /* Return result */
    return result(full2_output);
}
```

# Light LeNet-5 with C/C++ (Fixed-Point)

- Include Xilinx HLS library to use <ap\_fixed.h>

```
/* type.h */

#include <ap_fixed.h> // For fixed-point data type

typedef ap_ufixed<16, 1, AP_RND, AP_SAT> input_t;

typedef ap_fixed<16, 2, AP_RND, AP_SAT> weight_t;

typedef ap_fixed<16, 4, AP_RND, AP_SAT> conv1_t;
typedef ap_fixed<16, 5, AP_RND, AP_SAT> conv2_t;
typedef ap_fixed<16, 7, AP_RND, AP_SAT> conv3_t;
typedef ap_fixed<16, 7, AP_RND, AP_SAT> full1_t;
typedef ap_fixed<16, 7, AP_RND, AP_SAT> full2_t;

/*
    Temporary fixed-point data type with enough width
    to prevent saturation/overflow and avoid any loss of
    precision while accumulating the sum over all products.
    [2*I, 2*F]
*/

typedef ap_fixed<32, 8, AP_RND, AP_SAT> conv1_temp;
typedef ap_fixed<32, 10, AP_RND, AP_SAT> conv2_temp;
typedef ap_fixed<32, 14, AP_RND, AP_SAT> conv3_temp;
typedef ap_fixed<32, 14, AP_RND, AP_SAT> full1_temp;
typedef ap_fixed<32, 14, AP_RND, AP_SAT> full2_temp;
```

```
/* Makefile */

CC = g++
CXXLFAGS = -Wall -Wno-comment -g -O2
LIBS += -I $(XILINX_HLS)/include

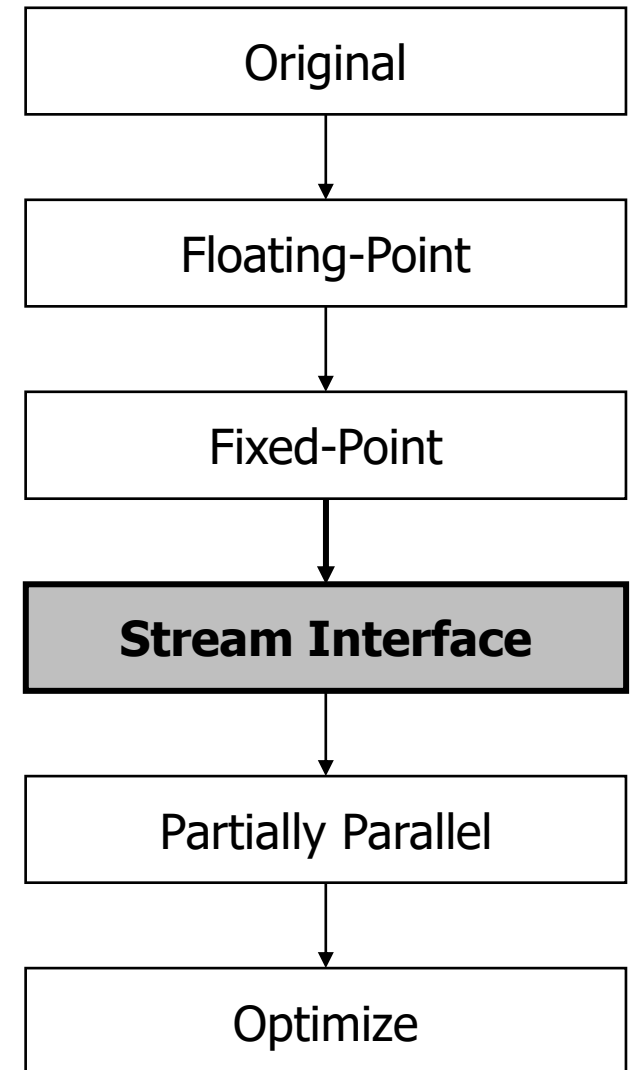
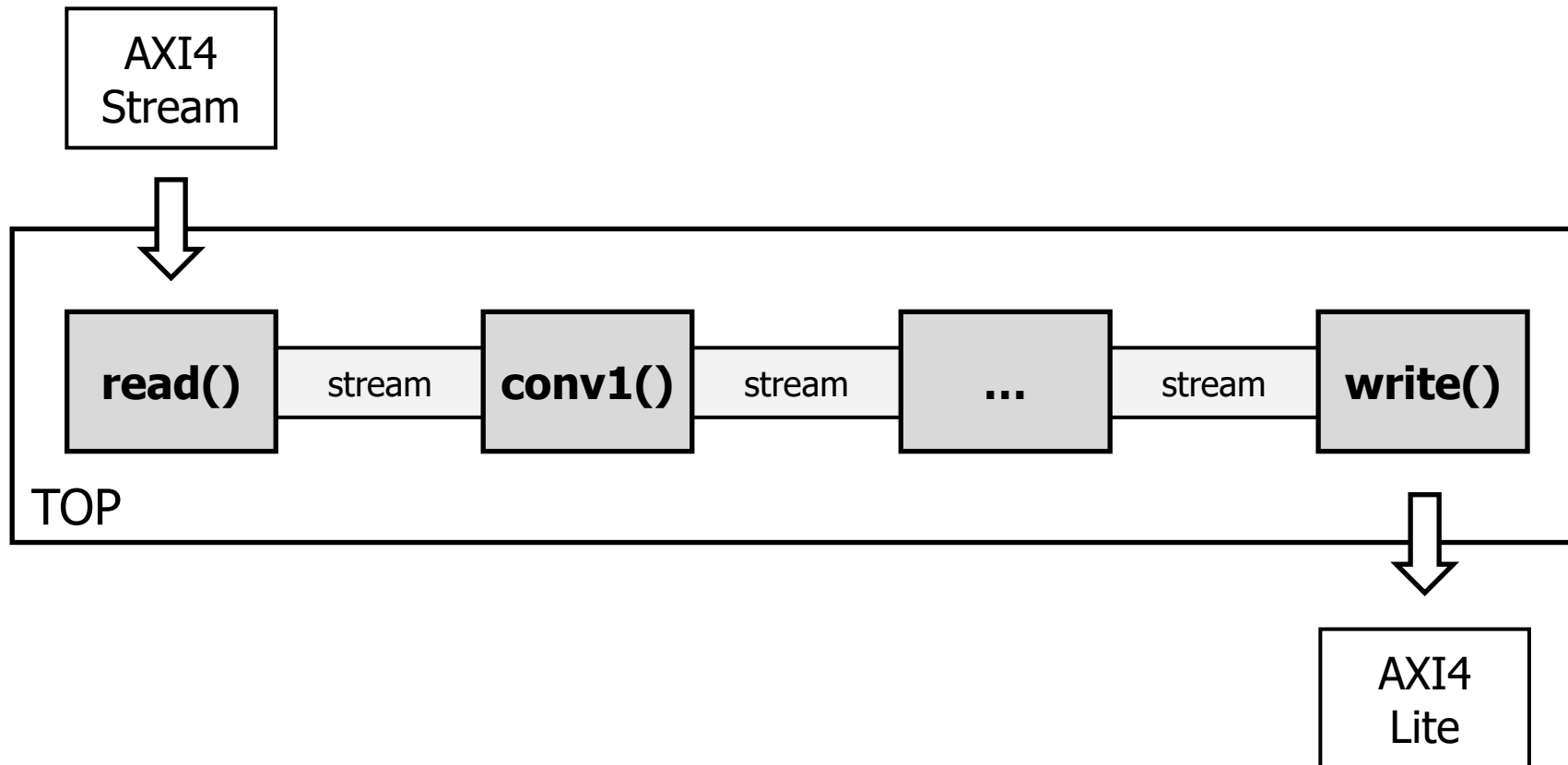
$(TARGET): $(OBJECTS)
    $(CC) $(CXXLFAGS) $(OBJECTS) -o $(TARGET) $(LIBS)
=====
# Run all inputs (just one option)
$ ./main

          Number  Answer  Predict
[TRUE] [   0]    7        7
[TRUE] [   1]    2        2
[TRUE] [   2]    1        1
[TRUE] [   3]    0        0
...
[TRUE] [9996]    3        3
[TRUE] [9997]    4        4
[TRUE] [9998]    5        5
[TRUE] [9999]    6        6

[Fixed-Point]
Accuracy : 97.47 [%]
Error    : 253 [cases]
```

# Light LeNet-5 with HLS (Stream Interface), Cont'd

- Implemented as stream interface instead of array interface as before.
- Closer to *Vivado HLS* canonical form using *DATAFLOW* optimization.
- For improving design throughput and latency.



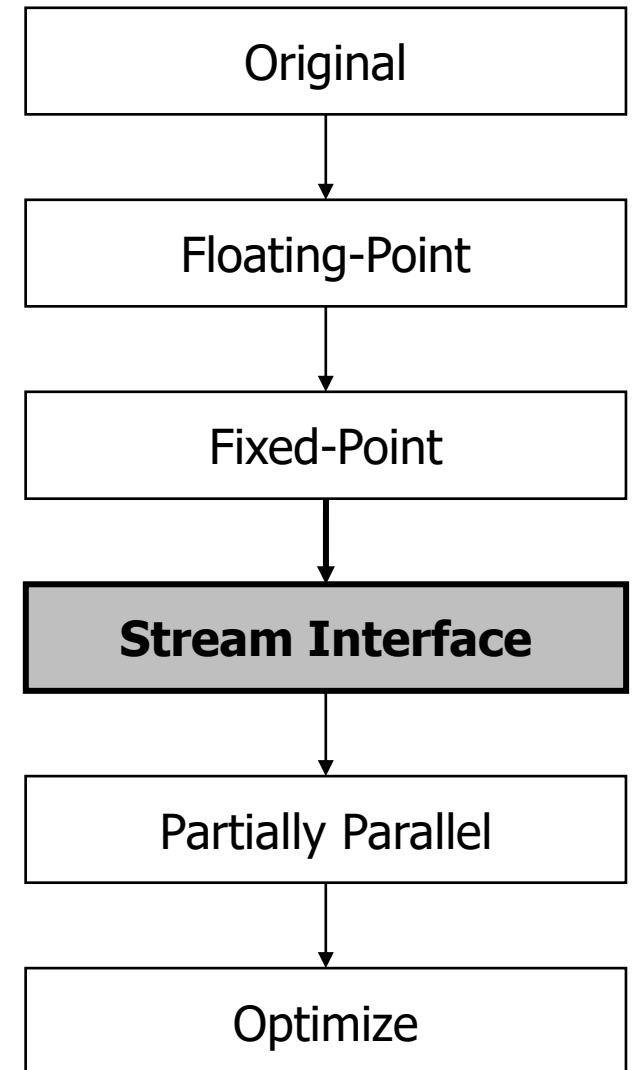
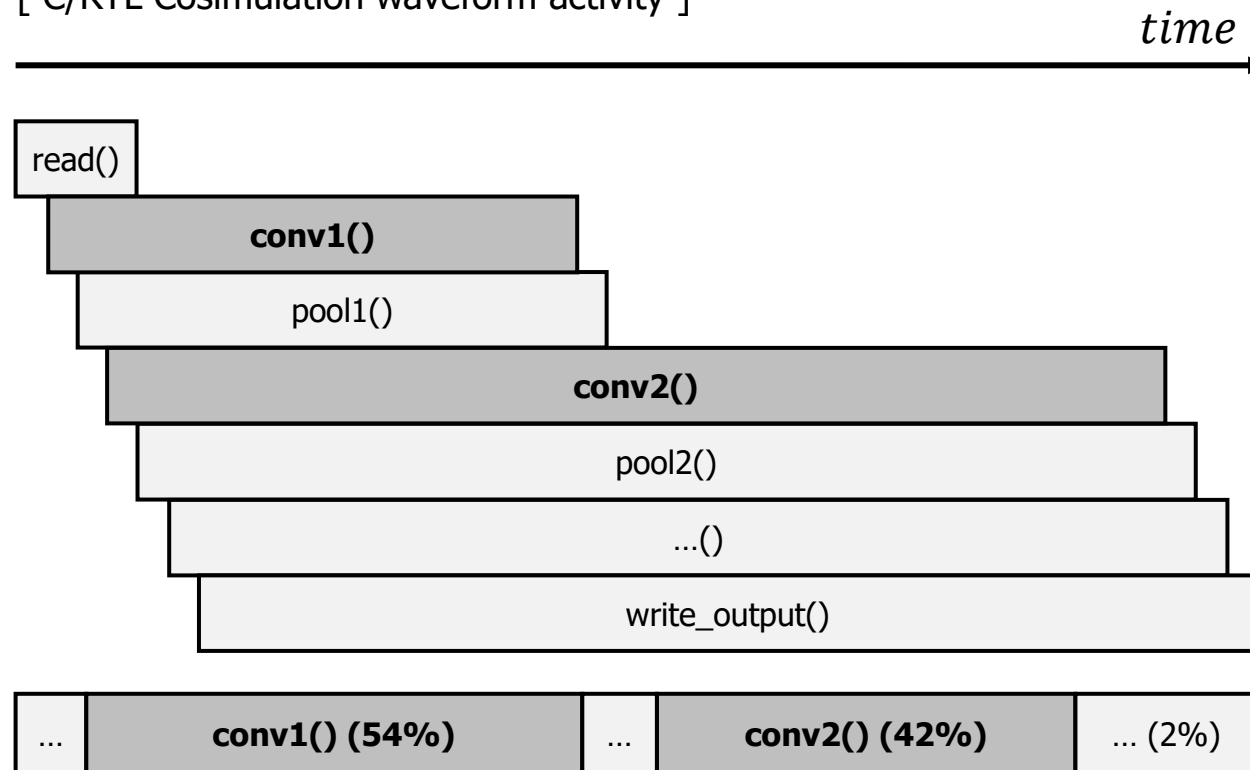


# Light LeNet-5 with HLS (Stream Interface), Cont'd

- *Vivado HLS* C/RTL Cosimulation waveform
- *Vivado HLS* Performance Estimates.
- It can be seen that **conv1()** and **conv2()** are the cause of bottleneck.

[ Performance Estimates ] [ C/RTL Cosimulation waveform activity ]

	Latency
read()	1025
<b>conv1()</b>	<b>264691</b>
pool1()	6734
<b>conv2()</b>	<b>203618</b>
pool2()	1856
conv3()	8324
full1()	524
full2()	442
write()	31
<b>Total</b>	<b>265683</b>



# Light LeNet-5 with HLS (Stream Interface), Cont'd

- Implemented as stream interface instead of array interface as before.

```
/* type.h */

#include <ap_fixed.h>          // For fixed-point data type
#include <hls_stream.h>        // For stream data type
#include <ap_axi_sdata.h>      // For AXI stream data type

// Data type for AXI streaming with side-channel
typedef ap_axiu<16, 0, 0, 0> axis_t;

typedef ap_fixed<16, 1, AP_RND, AP_SAT> input_t;
...
typedef ap_fixed<16, 7, AP_RND, AP_SAT> full2_t;
...
typedef ap_fixed<32, 8, AP_RND, AP_SAT> conv1_temp;
typedef ap_fixed<32,14, AP_RND, AP_SAT> full2_temp;
...

// Stream data type for AXI streaming with side-channel
typedef hls::stream<axis_t> stream_axis;

typedef hls::stream<input_t> stream_input;
typedef hls::stream<conv1_t> stream_conv1;
...
typedef hls::stream<full2_t> stream_full2;
```

```
/* layer.cc */

void read_input(stream_axis &input, stream_input &output) {
    axis_t temp_axis;
    input_t temp_input;

    for (int i = 0 ; i < IMAGE_SIZE; i++) {
        temp_axis = input.read();

        // range() selects all bits in normal order
        temp_input.range() = temp_axis.data.range();
        output.write(temp_input);
    }
}

...

void write_output(stream_full2 &input, uint8_t* output) {
    full2_t input_1d[...] = { 0, };

    /* Read input */
    for (int i = 0; i < FULL2_OUTPUT_SIZE; i++)
        input_1d[i] = input.read();

    ... // Find Max and index
    *output = (uint8_t)max_index;
}
```

# Light LeNet-5 with HLS (Stream Interface), Cont'd

- Implemented as stream interface instead of array interface as before.

```
/* layer.cc */
...
void full1_layer(stream_conv3 &input, stream_full1 &output) {

    conv3_t input_1d[CONV3_OUTPUT_NUM] = { 0, };

    /* Read input */
    for (int i = 0; i < CONV3_OUTPUT_NUM; i++) {
        input_1d[i] = input.read();
    }

    /* Compute & Write */
    for (int row = 0; row < ... ; ...) {

        full1_temp acc = full1_bias[row];

        // Compute
        for (int col = 0; col < ... ; ...) {
            acc += (full1_temp)(input_1d[col] * full1_weight[...][...]);
        }

        // ReLU Activation & Write
        if (acc > 0) output.write((full1_t)acc);
        else        output.write((full1_t)0.0);
    }
}
```

```
/* predict.cc */

void predict(stream_axis &input, uint8_t* output) {
    #pragma HLS INTERFACE axis port = input

    #pragma HLS INTERFACE s_axilite port = output ...
    #pragma HLS INTERFACE s_axilite port = return ...
    ...
    #pragma HLS DATAFLOW

    stream_input input_stream;

    stream_conv1 conv1_stream;
    ...
    stream_full2 full2_stream;

    /* Read input */
    read_input(input, input_stream);

    /* 1st CONV layer */
    conv1_layer(input_stream, conv1_stream);

    ...

    /* Write result */
    write_output(full2_stream, output);
}
```

# Light LeNet-5 with HLS (Stream Interface)

- Implemented as stream interface instead of array interface as before.

```
/* main.cc or testbench.cc */
```

```
int main (int argc, char* argv[]) {
```

```
    int err_cnt = 0;
```

```
    ...
```

```
    for (size_t i = 0; i < TEST_NUM; i++) {
```

```
        ...
```

```
        predict(input_axi_stream, &my_answer[i]);
```

```
        ...
```

```
        if (my_answer != answer[i]) {
```

```
            err_cnt++;
```

```
            cout << "[ FALSE ] " ... << "\n";
```

```
        }
```

```
        else {
```

```
            cout << "[ TRUE ] " ... << "\n";
```

```
        }
```

```
    }
```

```
    ...
```

```
    cout << "\n[Fixed-Point + HLS stream]\n";
```

```
    cout << "Accuracy : " << ... << accuracy << ... ;
```

```
    cout << "Error      : " << ... << err_cnt << ... ;
```

```
    return 0;
```

```
}
```

```
# Run all inputs (just one option)
```

```
$ ./main
```

```
          Number  Answer  Predict
```

```
[TRUE] [  0]  7      7
```

```
[TRUE] [  1]  2      2
```

```
[TRUE] [  2]  1      1
```

```
[TRUE] [  3]  0      0
```

```
    ...
```

```
[TRUE] [9996]  3      3
```

```
[TRUE] [9997]  4      4
```

```
[TRUE] [9998]  5      5
```

```
[TRUE] [9999]  6      6
```

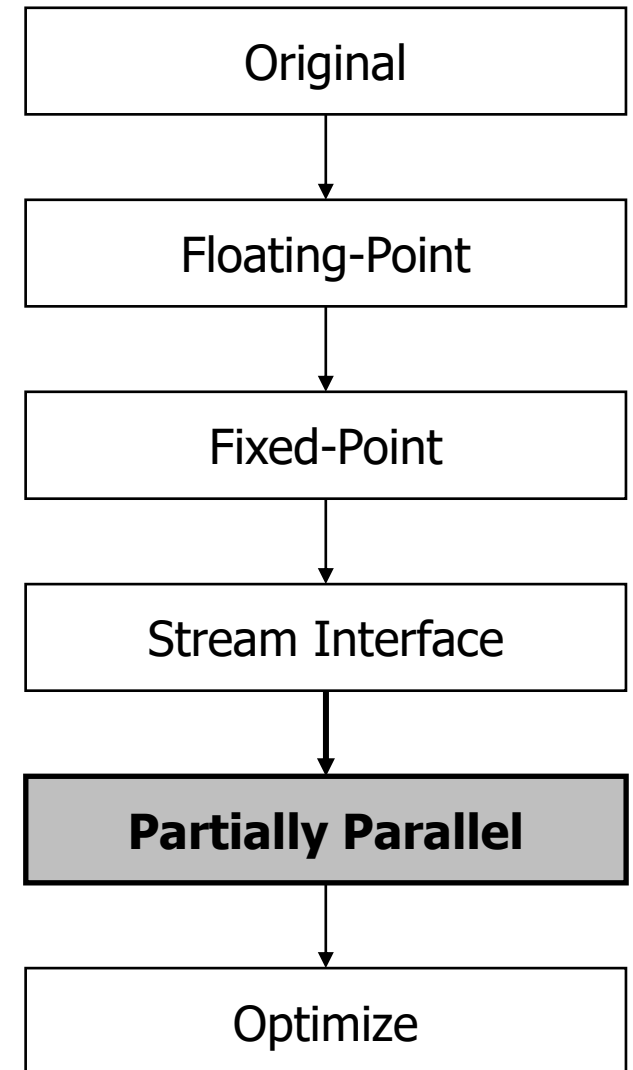
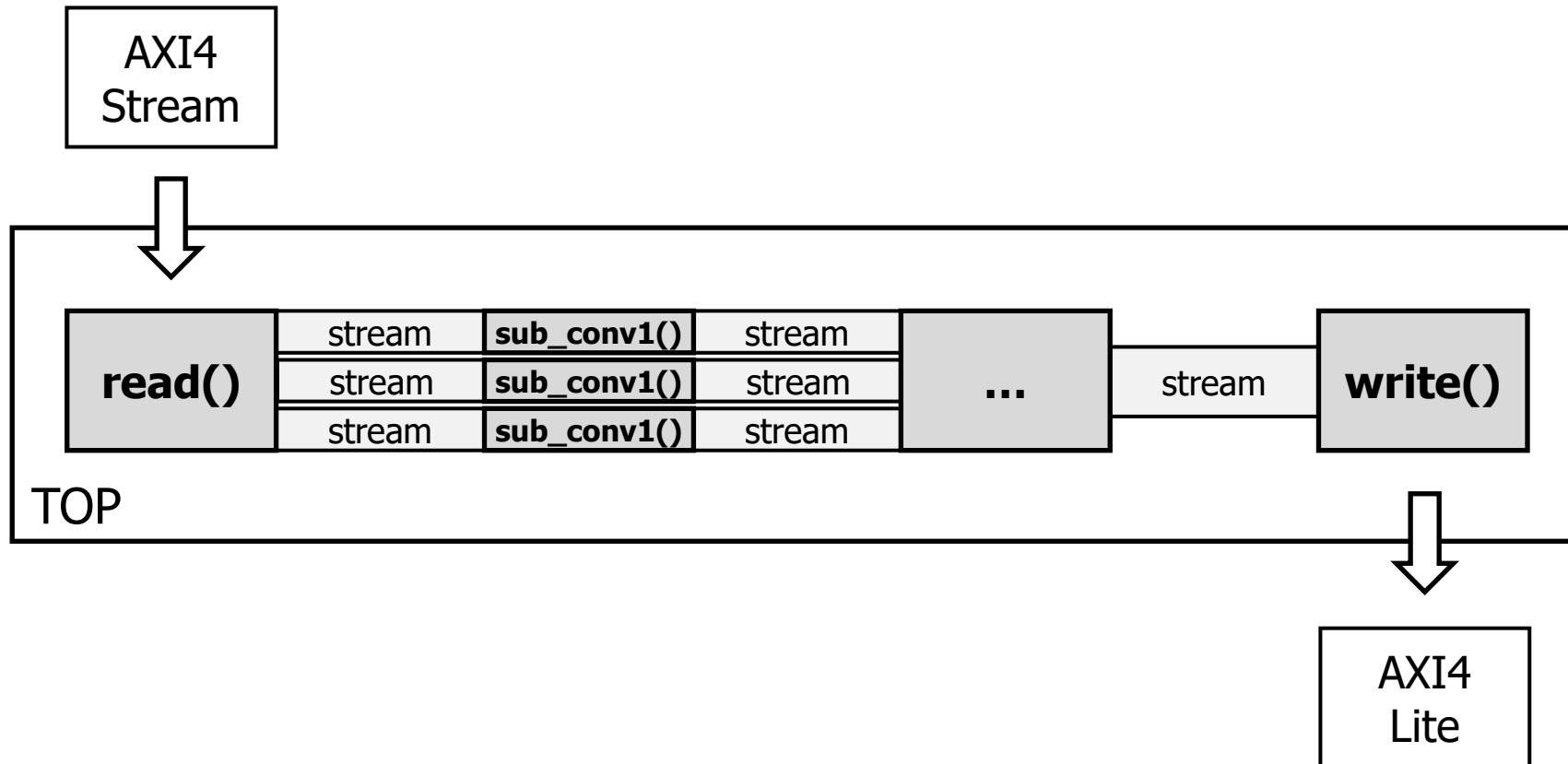
```
[Fixed-Point + HLS stream]
```

```
Accuracy : 97.47 [%]
```

```
Error      :      253 [cases]
```

# Light LeNet-5 with HLS (Partially Parallel), Cont'd

- Longest latency **conv1()** is divided to be calculated in parallel.
- Since the total latency is bounded by highest latency **conv1()**,
- Reducing the latency by parallelizing **conv1()** will reduce total latency.



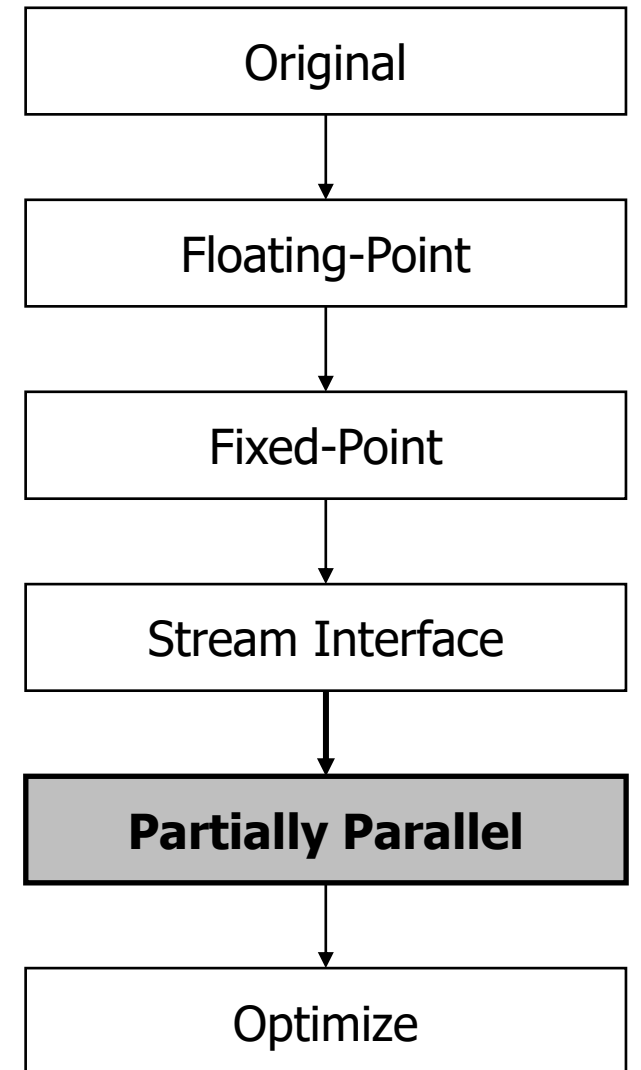
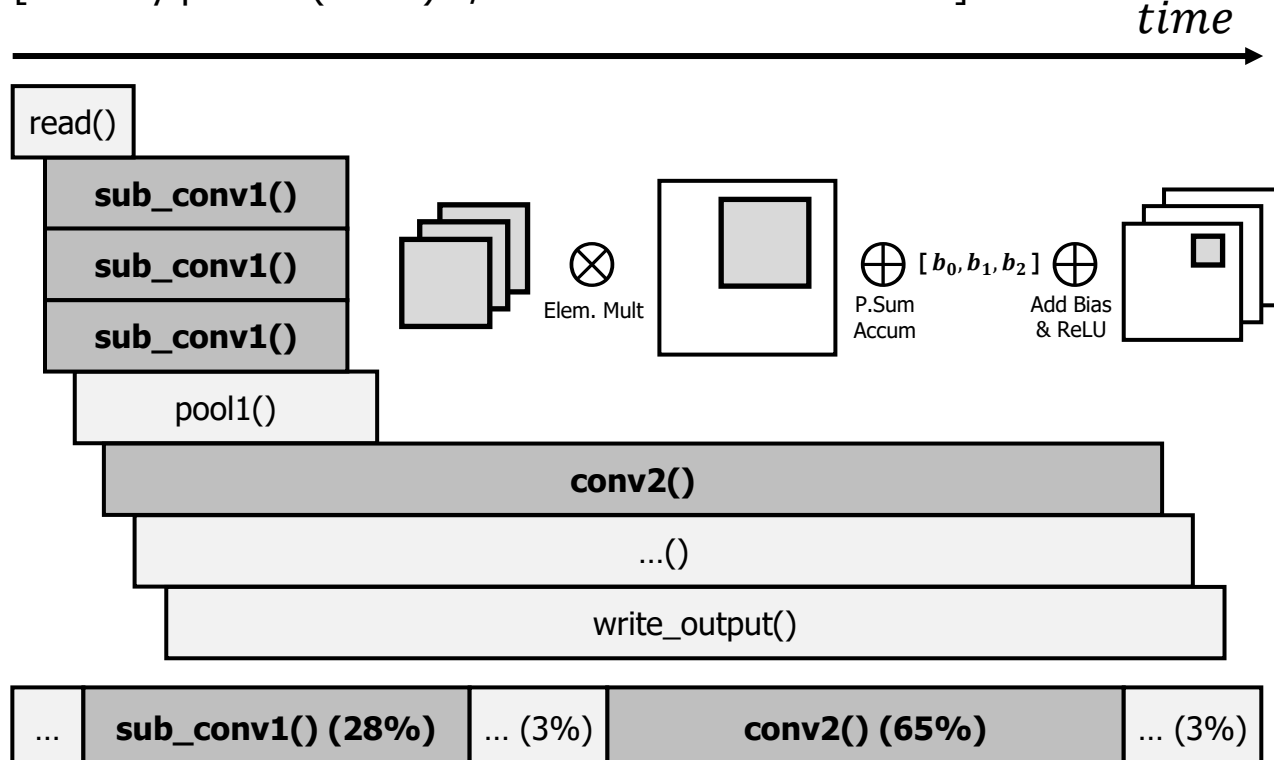
# Light LeNet-5 with HLS (Partially Parallel), Cont'd

- When **conv1()** was partially parallelized,
- The latency of **conv1()** decreased by **66%**.
- But, the total latency is bounded to new highest latency **conv2()**.

[ Performance Estimates ]

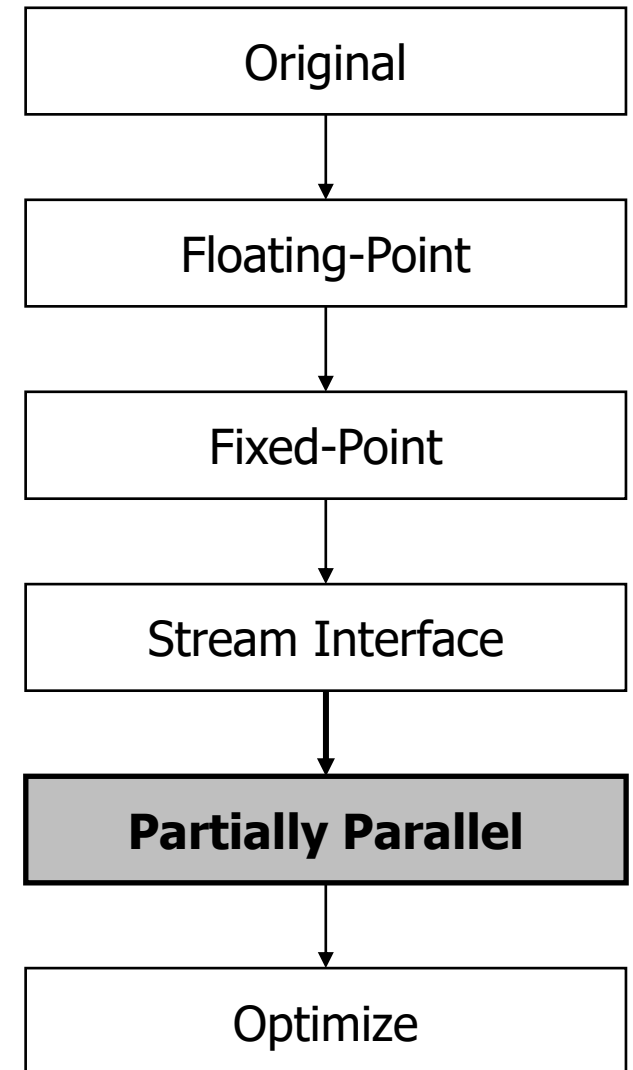
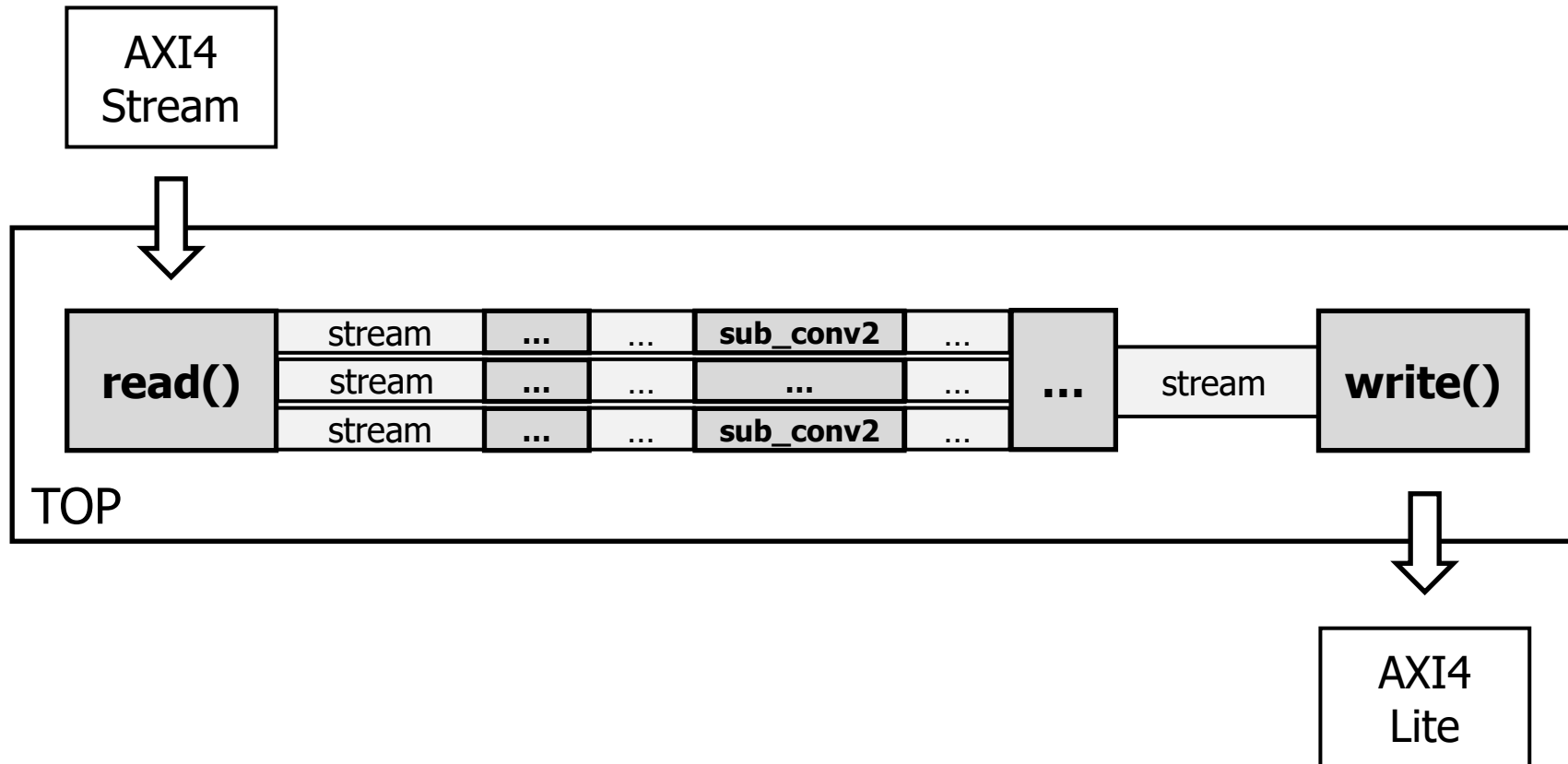
	Latency
read()	1025
<b>sub_conv1()</b>	<b>88954</b>
pool1()	10536
<b>conv2()</b>	<b>203618</b>
pool2()	1856
conv3()	8324
full1()	524
full2()	442
write()	31
<b>Total</b>	<b>207716</b>

[ Partially parallel (conv1) C/RTL Cosimulation waveform ]



# Light LeNet-5 with HLS (Partially Parallel), Cont'd

- New highest latency **conv2()** also divided to be calculated in parallel.
- Since the total latency is bounded by highest latency **conv2()**,
- Reducing the latency by parallelizing **conv2()** will reduce total latency.



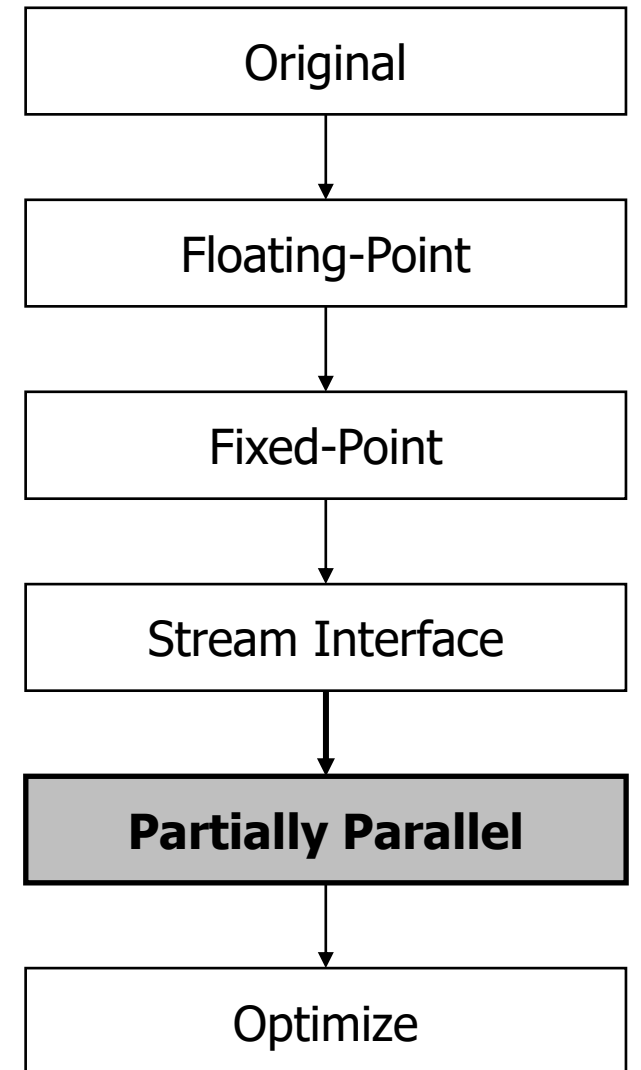
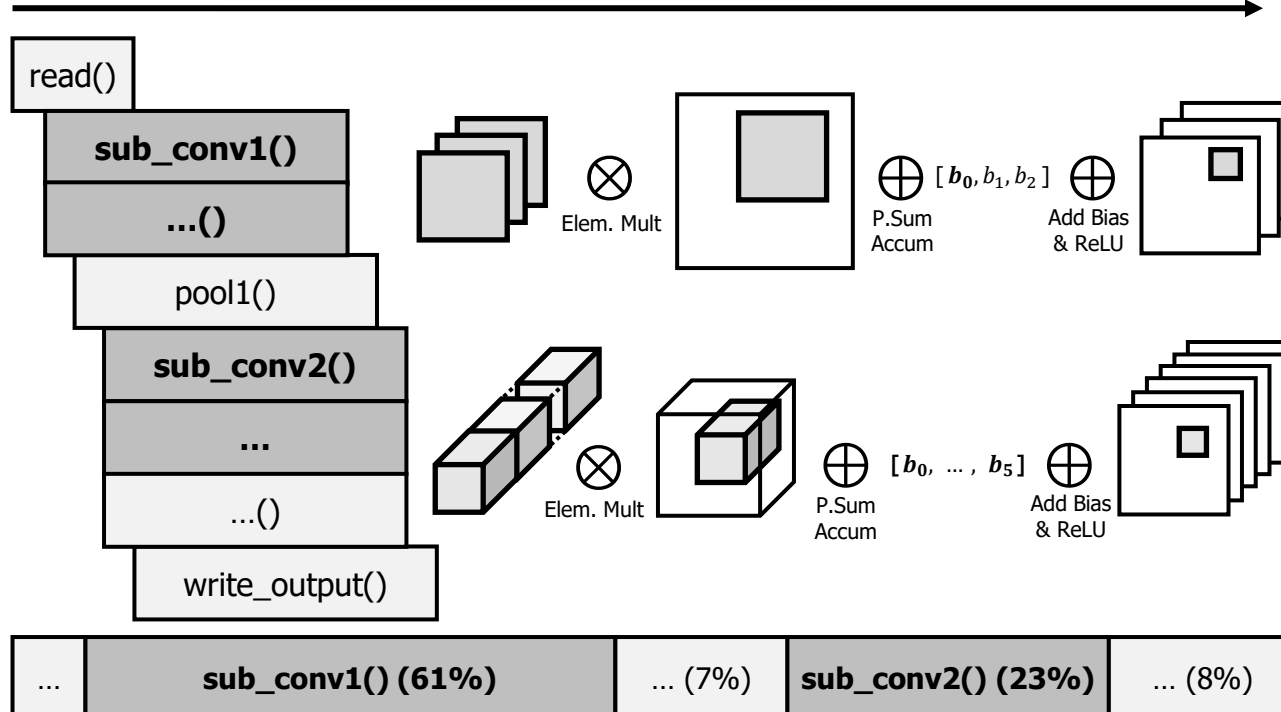
# Light LeNet-5 with HLS (Partially Parallel), Cont'd

- When **conv2()** was partially parallelized,
- The latency of **conv2()** decreased by **83%**.
- Total latency decreased by **66%** compared to the [Stream Interface].

[ Performance Estimates ]

[ Partially parallel (conv1 + conv2) C/RTL Cosimulation waveform ]

	Latency
read()	1025
<b>sub_conv1()</b>	<b>88954</b>
pool1()	10536
<b>sub_conv2()</b>	<b>34500</b>
pool2()	2544
conv3()	8324
full1()	524
full2()	442
write()	31
<b>Total</b>	<b>89946</b>





# Light LeNet-5 with HLS (Partially Parallel), Cont'd

- Overview of latency reduction.

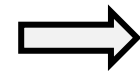
[ No parallel ]

	Latency
read()	1025
<b>conv1()</b>	<b>264691</b>
pool1()	6734
<b>conv2()</b>	<b>203618</b>
pool2()	1856
conv3()	8324
full1()	524
full2()	442
write()	31
<b>Total</b>	<b>265683</b>



[ Partially parallel (conv1) ]

	Latency
read()	1025
<b>sub_conv1()</b>	<b>88954</b>
pool1()	10536
<b>conv2()</b>	<b>203618</b>
pool2()	1856
conv3()	8324
full1()	524
full2()	442
write()	31
<b>Total</b>	<b>207716</b>



[ P.P (conv1 + conv2) ]

	Latency
read()	1025
<b>sub_conv1()</b>	<b>88954</b>
pool1()	10536
<b>sub_conv2()</b>	<b>34500</b>
pool2()	2544
conv3()	8324
full1()	524
full2()	442
write()	31
<b>Total</b>	<b>89946</b>

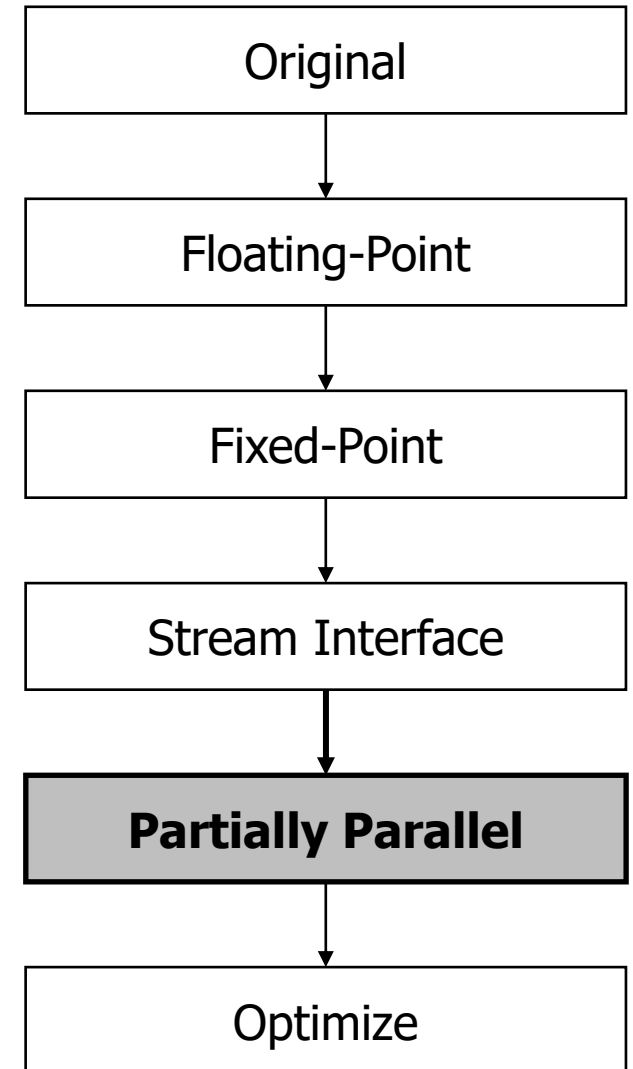
**66% ↓**

**83% ↓**

**22% ↓**

**57% ↓**

**Total latency 66% ↓**

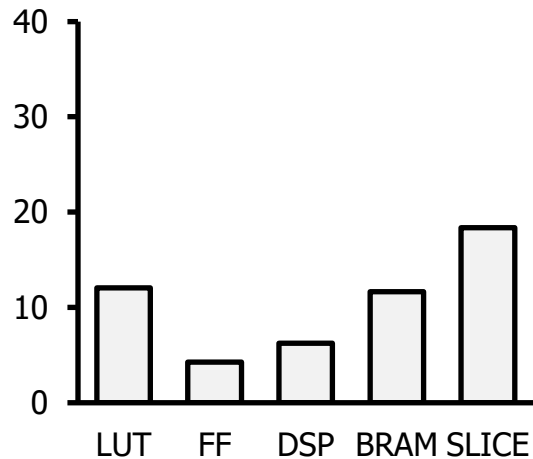


# Light LeNet-5 with HLS (Partially Parallel), Cont'd

- Overview of resource utilization.

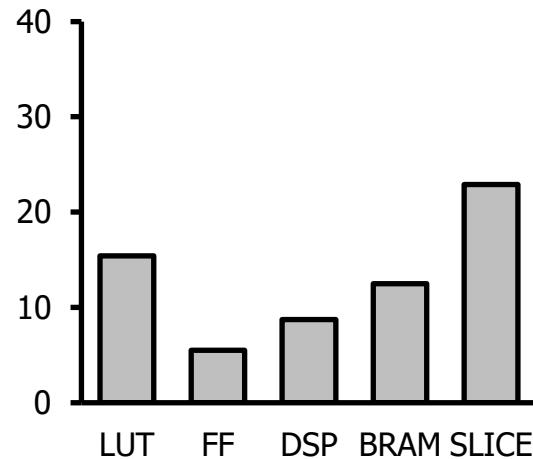
[No parallel]

	Usage	Usage (%)
LUT	2125	12.07
FF	1504	4.27
DSP	5	6.25
BRAM	14	11.67
SLICE	808	18.36



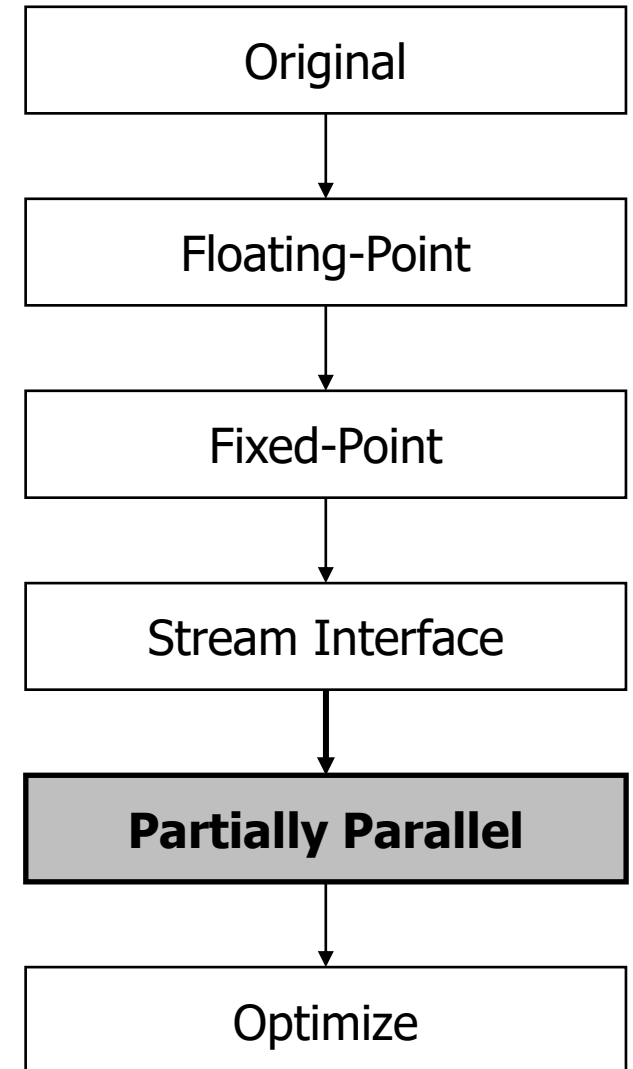
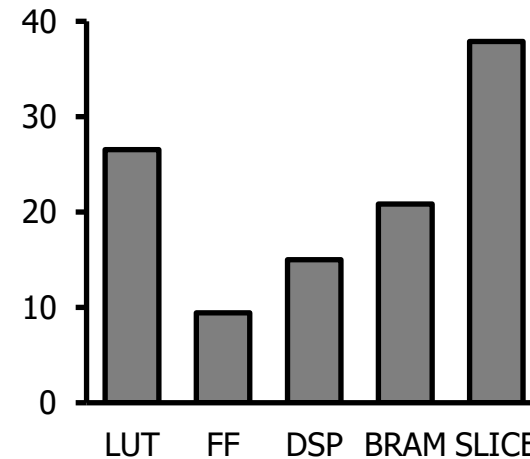
[Patially parallel (conv1)]

	Usage	Usage (%)
LUT	2713	15.41
FF	1946	5.53
DSP	7	8.75
BRAM	15	12.50
SLICE	1008	22.91



[Patially parallel (conv1 + conv2)]

	Usage	Usage (%)
LUT	4671	26.54
FF	3324	9.44
DSP	12	15.00
BRAM	25	20.83
SLICE	1667	37.89



# Light LeNet-5 with HLS (Partially Parallel), Cont'd

- Divide existing functions to be executed in parallel.

```
/* predict.cc */
```

```
void predict(stream_axis &input, uint8_t* output_r) {  
  
    stream_input sub0_input; stream_conv1 sub0_conv1;  
    stream_input sub1_input; stream_conv1 sub1_conv1;  
    stream_input sub2_input; stream_conv1 sub2_conv1;  
  
    stream_conv1 sub0_pool1;  
    ...  
    stream_conv1 sub5_pool1;  
  
    stream_conv2 sub0_conv2;  
    ...  
    stream_conv2 sub5_conv2;  
    ...  
    read_input(input, sub0_input,  
                sub1_input,  
                sub2_input);  
  
    /* 1st CONV layer */  
    sub0_conv1_layer(sub0_input, sub0_conv1);  
    sub1_conv1_layer(sub1_input, sub1_conv1);  
    sub2_conv1_layer(sub2_input, sub2_conv1);  
  
    /* 1st POOL layer */  
    pool1_layer(sub0_conv1, sub1_conv1, sub2_conv1,  
                sub0_pool1, sub1_pool1, sub2_pool1,  
                sub3_pool1, sub4_pool1, sub5_pool1);
```

```
/* predict.cc */
```

```
    ...  
    /* 2nd CONV layer */  
    sub0_conv2_layer(sub0_pool1, sub0_conv2);  
    ...  
    sub5_conv2_layer(sub5_pool1, sub5_conv2);  
  
    /* 2nd POOL layer */  
    pool2_layer(sub0_conv2,  
                sub1_conv2,  
                sub2_conv2,  
                sub3_conv2,  
                sub4_conv2,  
                sub5_conv2, pool2_stream);  
  
    /* 3rd CONV layer */  
    conv3_layer(pool2_stream, conv3_stream);  
  
    /* 1st FULL layer */  
    full1_layer(conv3_stream, full1_stream);  
  
    /* 2nd FULL layer */  
    full2_layer(full1_stream, full2_stream);  
  
    /* Return result */  
    write_output(full2_stream, output_r);  
}
```

# Light LeNet-5 with HLS (Partially Parallel), Cont'd

- Divide existing functions to be executed in parallel.

```
/* layer.cc */

void read_input(stream_axis &input,
                stream_input &sub0,
                stream_input &sub1,
                stream_input &sub2)
{
    axis_t temp_axis;
    input_t temp_input;

    for (int i = 0; i < IMAGE_SIZE; i++) {
        temp_axis = input.read();
        temp_input.range() = temp_axis.data.range();

        sub0.write(temp_input);
        sub1.write(temp_input);
        sub2.write(temp_input);
    }
}
```

```
/* layer.cc */

void sub0_conv1_layer(stream_input &input, stream_conv1 &output)
{
    input_t input_2d[IMAGE_ROW][IMAGE_COL] = { 0, };

    /* Read input */
    ...
    /* Compute & Write */
    for (int orow = 0; orow < ... ; ...) {
        for (int ocol = 0; ocol < ... ; ...) {
            conv1_temp acc = conv1_bias[0];

            for (int wr = 0; wr < ... ; ...) {
                for (int wc = 0; wc < ... ; ...) {
                    // Compute
                    acc += (conv1_temp)(input_2d[...][...] * weight_0[...][...]);
                }
            }
            // ReLU Activation & Write
            if (acc > 0) output.write((conv1_t)acc);
            else        output.write((conv1_t)0.0);
        }
    }
}
```

# Light LeNet-5 with HLS (Partially Parallel)

- Divide existing functions to be executed in parallel.

```
/* layer.cc */
```

```
void pool1_layer(stream_conv1 &input0, stream_conv1 &input1, stream_conv1 &input2,  
                 stream_conv1 &output0, stream_conv1 &output1, stream_conv1 &output2,  
                 stream_conv1 &output3, stream_conv1 &output4, stream_conv1 &output5)
```

```
{  
    conv1_t input_3d[CONV1_OUTPUT_NUM][CONV1_OUTPUT_ROW][CONV1_OUTPUT_COL] = { 0, };
```

```
    /* Read input */
```

```
    for (int i = 0; i < ... ; ...) {  
        for (int j = 0; j < ... ; ...) {  
            for (int k = 0; k < ... ; ...) {  
                if      (k == 0) input_3d[k][i][j] = input0.read();  
                else if (k == 1) input_3d[k][i][j] = input1.read();  
                else      input_3d[k][i][j] = input2.read();  
            }  
        }  
    }
```

```
    /* Compute & Write */
```

```
    for (int num = 0; num < ... ; ...) {  
        for (int row = 0; row < ... ; ... ) {  
            for (int col = 0; col < ... ; ...) {  
                ... // Find Max  
                output0.write(max);  
                ...  
                output5.write(max);  
            }  
        }  
    }  
}
```

```
# Run all inputs (just one option)
```

```
$ ./main
```

	Number	Answer	Predict
[TRUE]	[ 0]	7	7
[TRUE]	[ 1]	2	2
[TRUE]	[ 2]	1	1
[TRUE]	[ 3]	0	0
...			
[TRUE]	[9996]	3	3
[TRUE]	[9997]	4	4
[TRUE]	[9998]	5	5
[TRUE]	[9999]	6	6

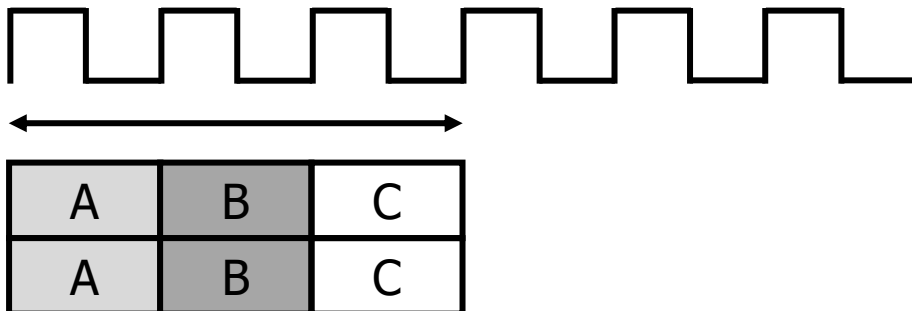
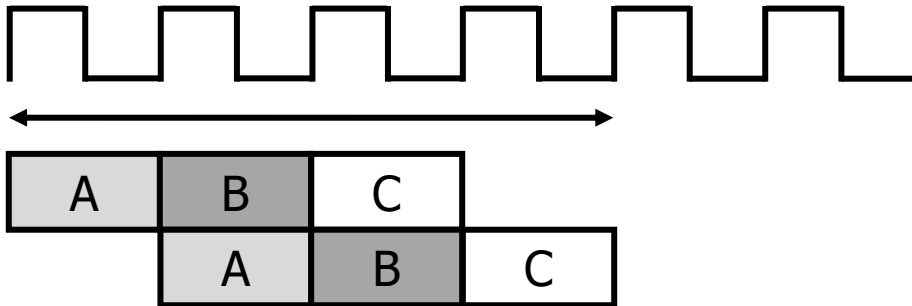
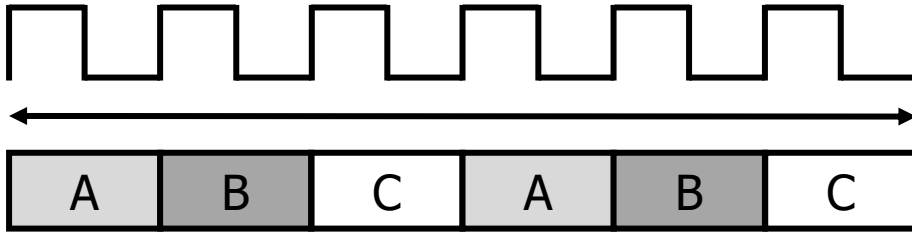
```
[Fixed-Point + HLS stream + Parallel]
```

```
Accuracy : 97.47 [%]
```

```
Error      :      253 [cases]
```

# Light LeNet-5 with HLS (Optimize), Cont'd

- Compare no pragma, HLS PIPELINE, HLS UNROLL



## No pragma

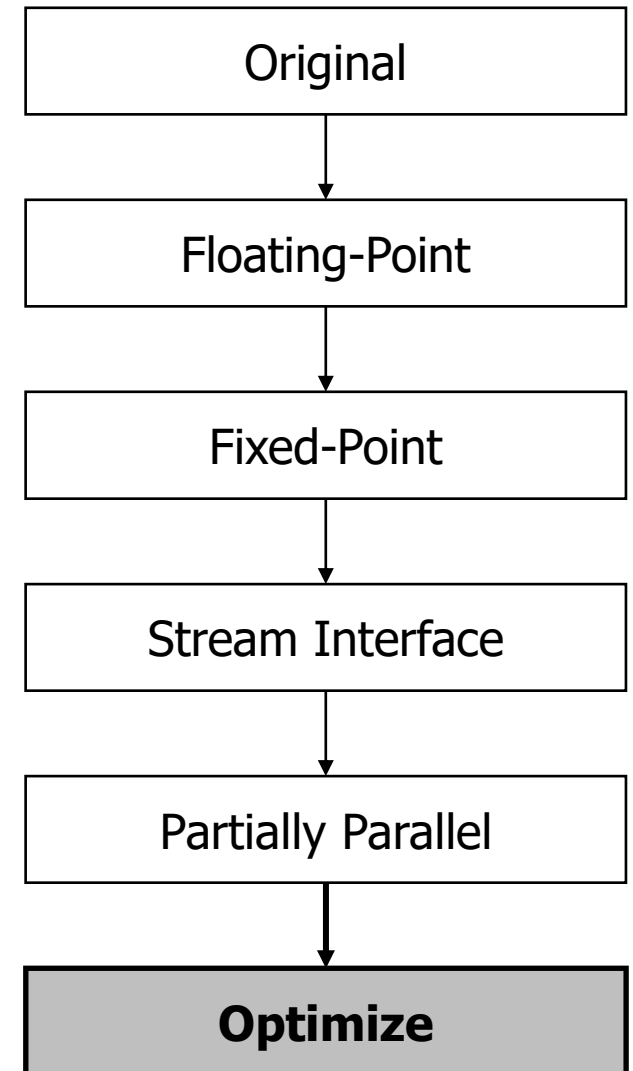
- High latency
- Low resource
- Low throughput

## HLS PIPELINE

- Medium latency
- Medium resource
- Medium throughput

## HLS UNROLL

- Low latency
- High resource
- High throughput



# Light LeNet-5 with HLS (Optimize), Cont'd

- When UNROLL pragma is applied to **conv1()** and **conv2()**,
- The latency of **conv1()** is decreased by **62%**.
- The latency of **conv2()** is decreased by **57%**.
- The **total latency** is decreased by **62%**.

[ Before apply pragma ]

	Latency
read()	1025
<b>sub_conv1()</b>	<b>88954</b>
pool1()	10536
<b>sub_conv2()</b>	<b>34500</b>
pool2()	2544
conv3()	8324
full1()	524
full2()	442
write()	31
<b>Total</b>	<b>89946</b>



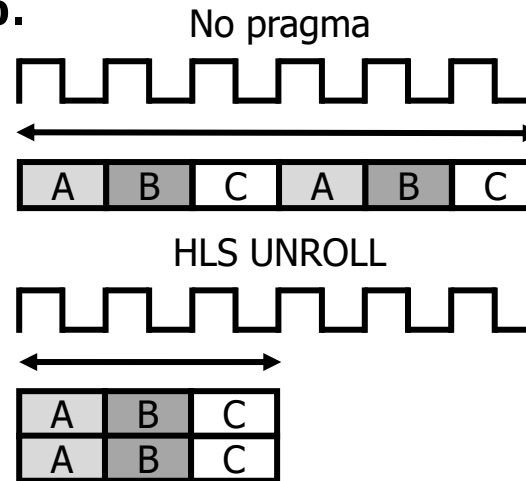
[ After apply pragma ]

	Latency
read()	1025
<b>sub_conv1()</b>	<b>34011</b>
pool1()	10536
<b>sub_conv2()</b>	<b>14912</b>
pool2()	2544
conv3()	8324
full1()	524
full2()	442
write()	31
<b>Total</b>	<b>34011</b>

**62% ↓**

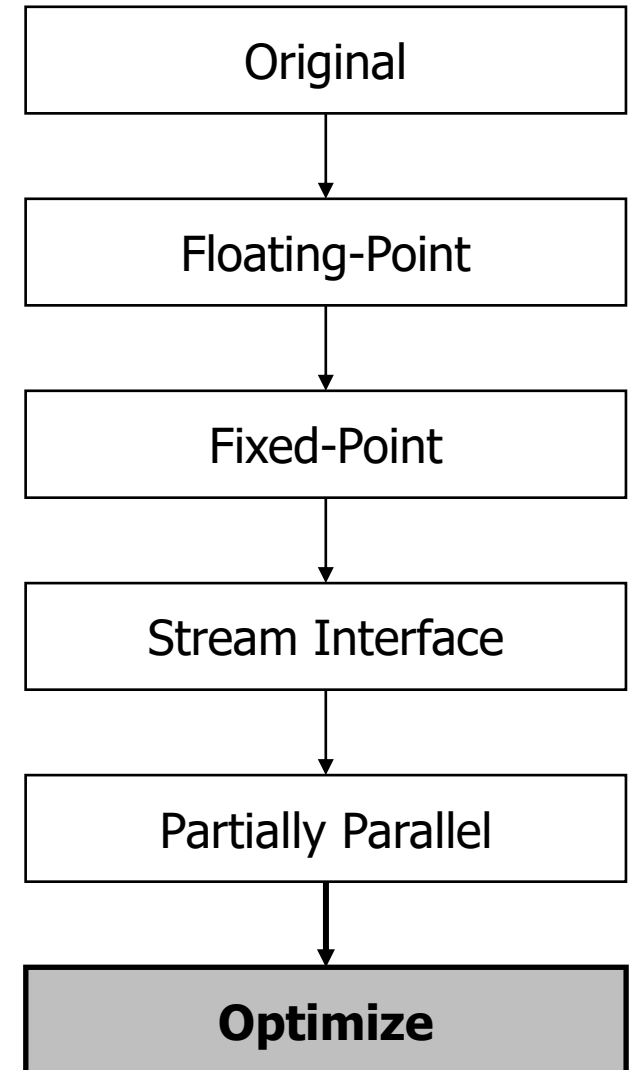
**57% ↓**

**62% ↓**



[ Resource Usage ]

	Usage	Usage (%)
LUT	8310	47.22
FF	4727	13.43
DSP	48	60.00
BRAM	19	15.83
SLICE	2935	66.70



# Light LeNet-5 with HLS (Optimize)

- Apply optimization each block.

```
/* layer.cc */

void sub0_conv1_layer(stream_input &input, stream_conv1 &output) {
    input_t input_2d[IMAGE_ROW][IMAGE_COL] = { 0, };

    /* Read input */
    for (int i = 0; i < IMAGE_ROW; i++) {
        for (int j = 0; j < IMAGE_COL; j++) {
            #pragma HLS PIPELINE II=1           // Read input in pipeline manner to reduce this function latency.
            input_2d[i][j] = input.read();
        }
    }

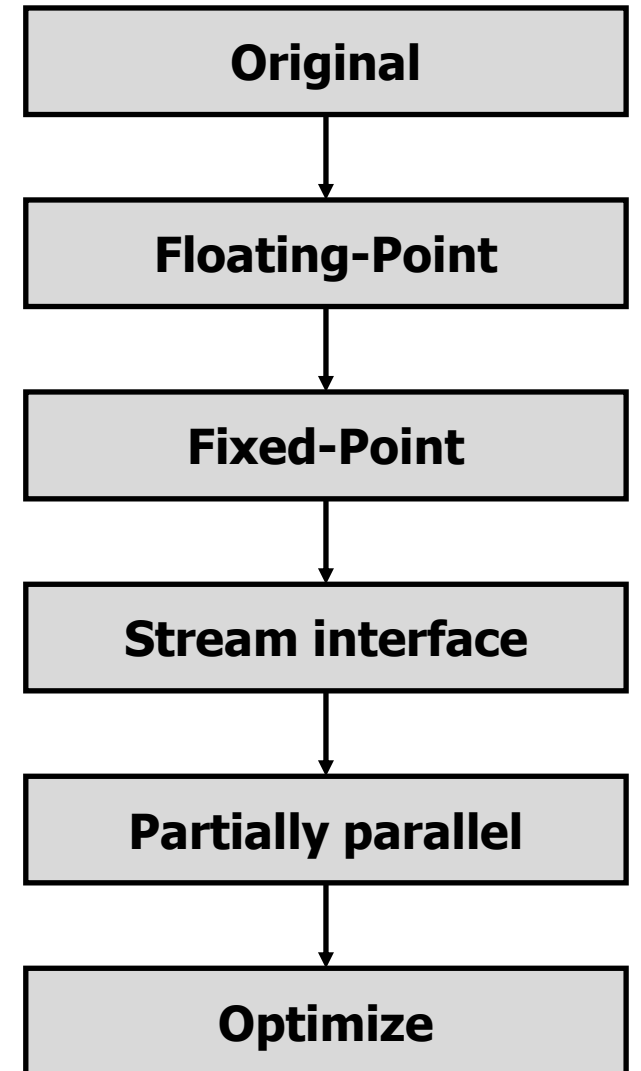
    /* Compute & Write */
    for (int orow = 0; orow < CONV1_OUTPUT_ROW; orow++) {
        for (int ocol = 0; ocol < CONV1_OUTPUT_COL; ocol++) {
            // Compute
            conv1_temp acc = conv1_bias[0];

            for (int wr = 0; wr < WEIGHT_ROW; wr++) {
                for (int wc = 0; wc < WEIGHT_COL; wc++) {
                    #pragma HLS UNROLL           // The loop is unrolled by WEIGHT_COL(5) and fully parallelized.
                    acc += (conv1_temp)(input_2d[orow + wr][ocol + wc] * conv1_weight_sub0[wr][wc]);
                }
            }
            // ReLU Activation & Write
            if (acc > 0) output.write((conv1_t)acc);
            else         output.write((conv1_t)0.0);
        }
    }
}
```



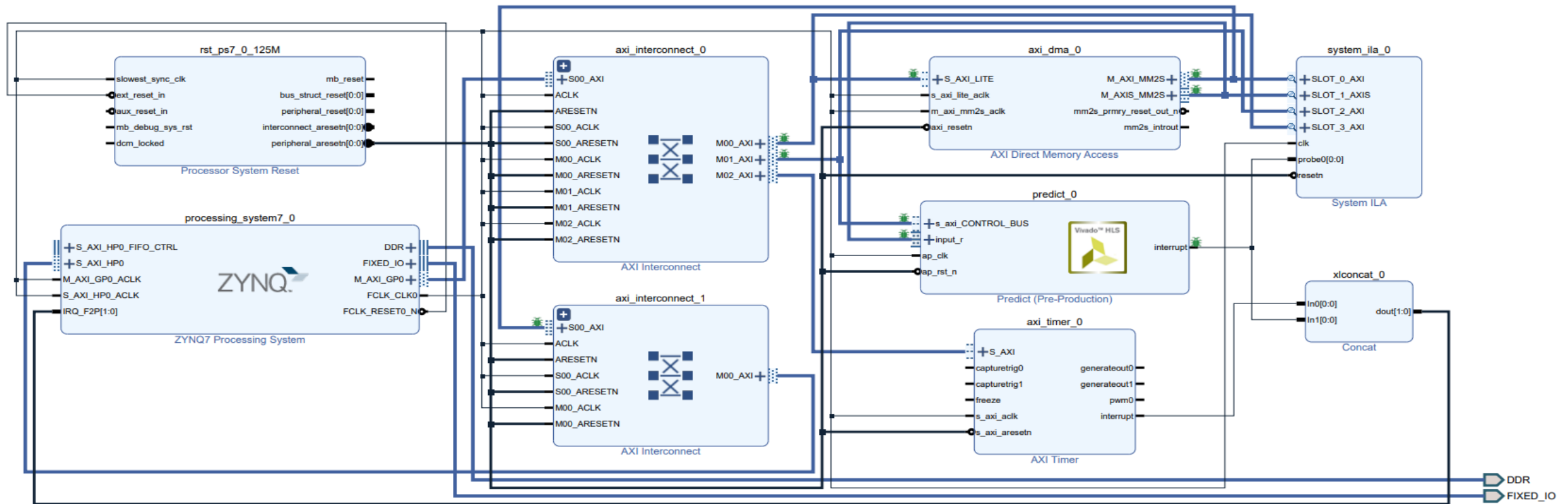
# Overview of Impl. of LeNet-5 using C/C++ and HLS

- **Original → Floating-Point**
  - Change activation function (*sigmoid, tanh* to *ReLU*).
  - Adjust the parameters of each layer to make the model smaller.
- **Floating-Point → Fixed-Point**
  - Confirmed that fixed-point has more benefits than floating-point.
  - Determine fixed-point identifier, for using fixed-point data type.
- **Fixed-Point → Stream interface**
  - Implemented as stream interface instead of array interface as before.
  - Closer to *Vivado HLS* canonical form using *DATAFLOW* optimization.
- **Stream interface → Partially parallel**
  - Reducing the latency by parallelizing *conv1()* and *conv2()*.
  - Increase the throughput by parallelizing *conv1()* and *conv2()*.
- **Partially parallel → Optimize**
  - Confirmed that *UNROLL* is more benefits than *PIPELINE*.
  - Apply optimization and confirmed reduced total latency.



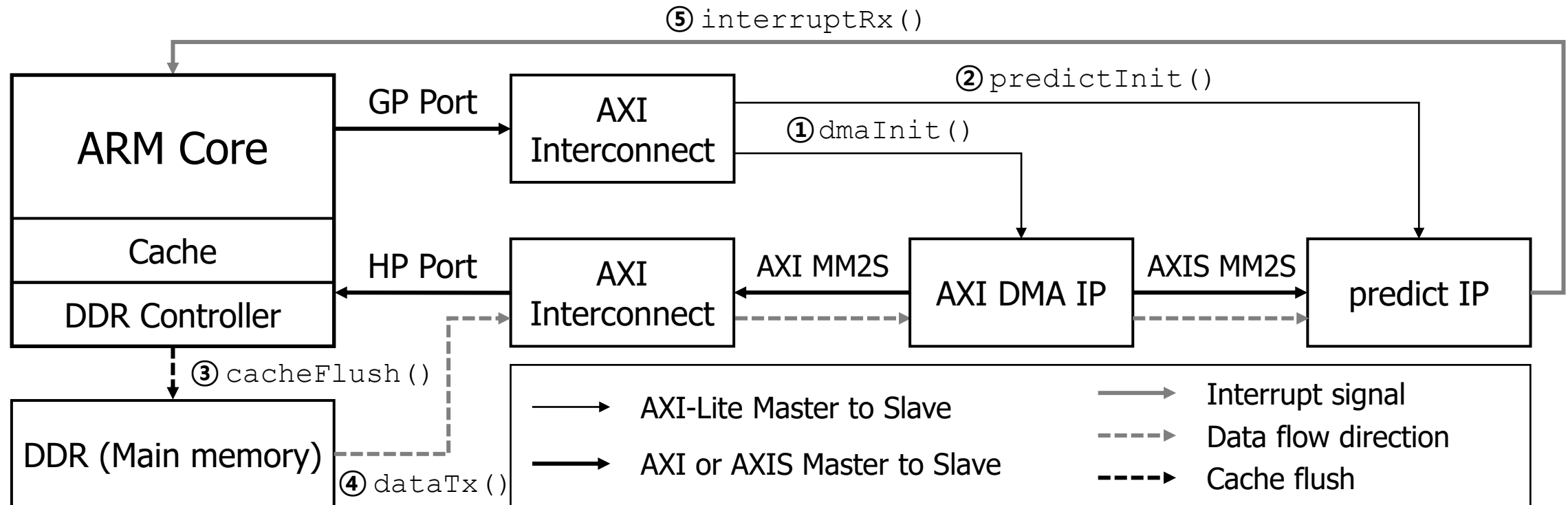
# HW Design of LeNet-5 (Programmable Logic), Cont'd

- The AXI DMA provides high-speed data movement between system memory and an AXI4-Stream-based target IP.
- The Integrated Logic Analyzer (ILA) IP core is logic analyzer core that can be used to monitor the internal signals of a design.
- The AXI Timer provides an AXI4-Lite interface to communicate with the Processing System.
- Interrupt is driven when the `ap_done` or `ap_ready` signals are HIGH.



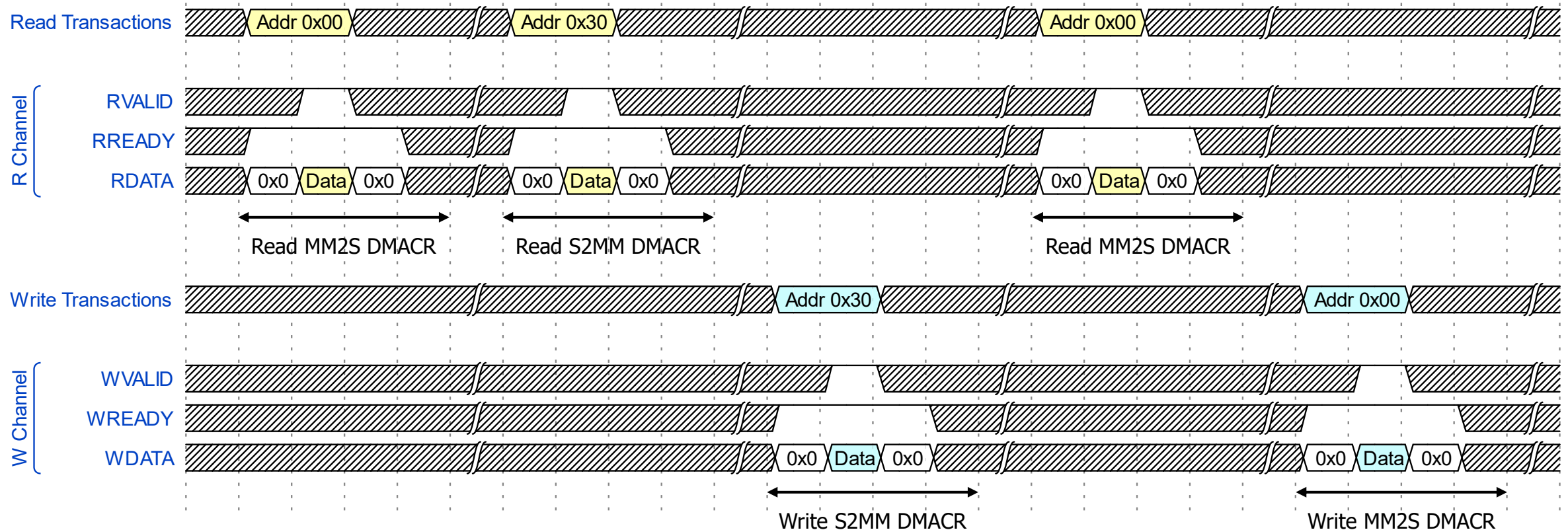
# HW Design of LeNet-5 (Programmable Logic), Cont'd

- Initialize AXI DMA IP and predict IP through `dmaInit()` and `predictInit()`.
- Flush the cache before transferring data via DMA through `cacheFlush()`.
- AXI DMA IP reads data through DDR and transfers it to predict IP through `dataTx()`.
- Wait for the predict IP to process, and read the result when interrupt signal is raised.



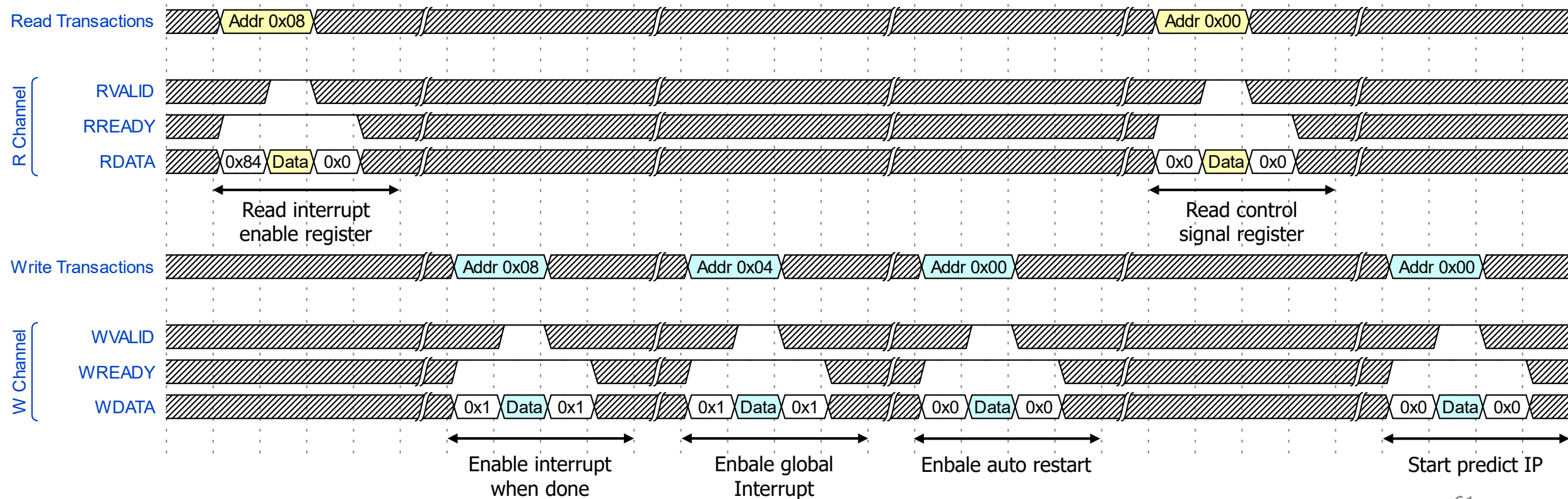
# HW Design of LeNet-5 (Programmable Logic), Cont'd

- Waveform in `dmaInit()` part obtained through Integrated Logic Analyzer (ILA).
- Read and write DMA IP config.
- ARM Core ascertain that read channel is open by reading and writing MM2S control register.
- ARM Core ascertain that write channel is not open by reading and writing S2MM control register.



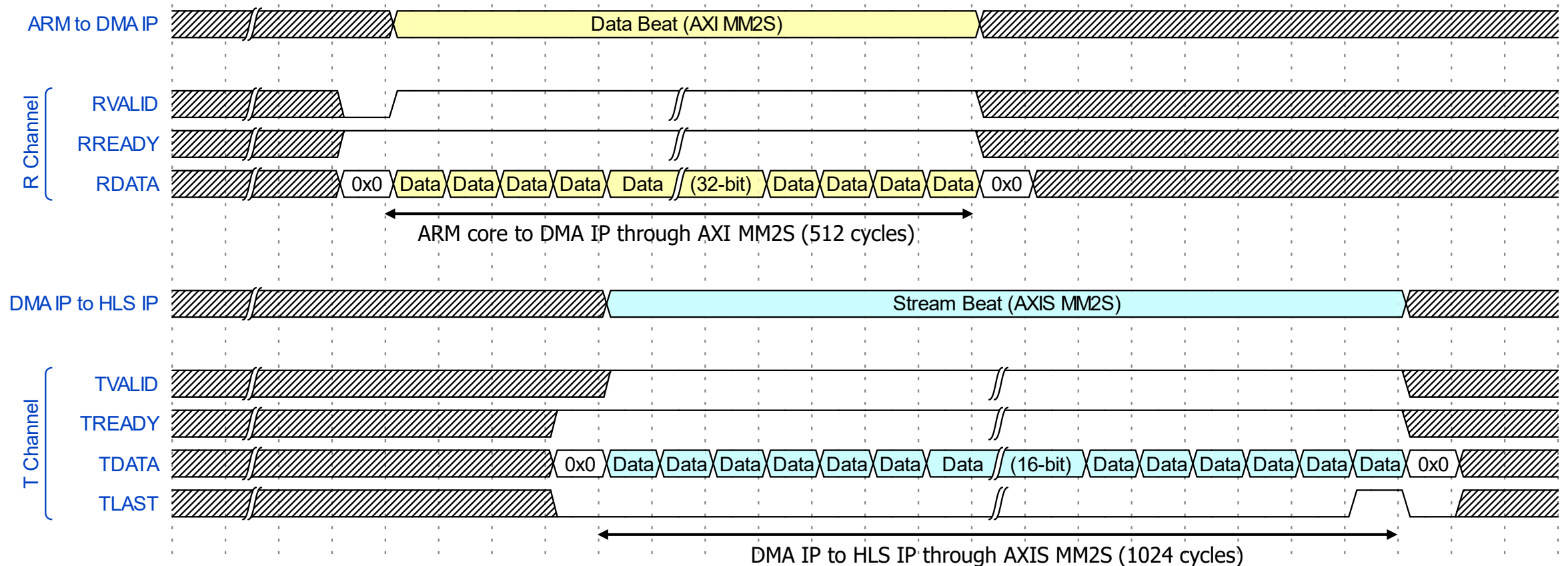
# HW Design of LeNet-5 (Programmable Logic), Cont'd

- Waveform in `predictInit()` part obtained through Integrated Logic Analyzer (ILA).
- Read and write predict IP config.
- Set the interrupt signal to be HIGH when the application done.
- Enable predict IP automatically restart and start.



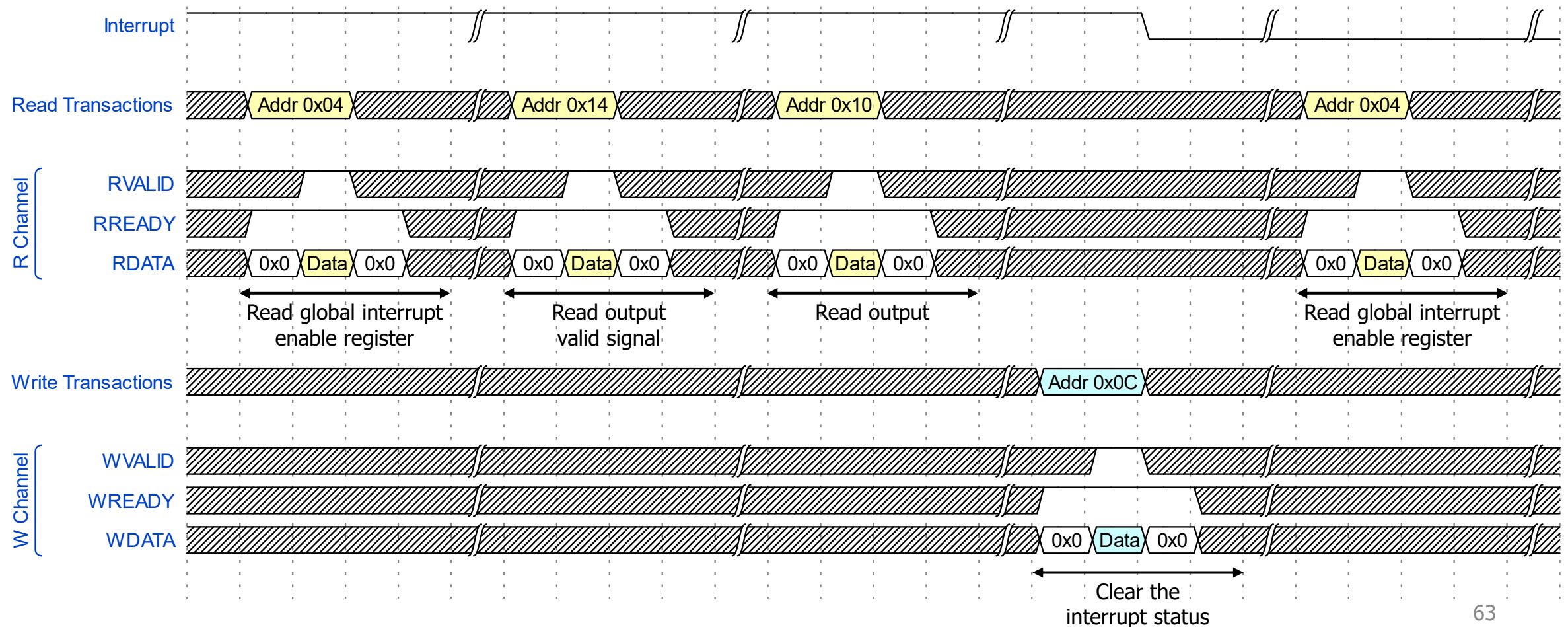
# HW Design of LeNet-5 (Programmable Logic), Cont'd

- Waveform in `dataTx()` part obtained through Integrated Logic Analyzer (ILA).
- Data transfer from ARM Core to HLS IP through DMA IP.
- ARM to DMA IP reads data in 32-bit units, while DMA IP to HLS IP reads data in 16-bit units.



# HW Design of LeNet-5 (Programmable Logic)

- Waveform in `interruptRx()` part obtained through Integrated Logic Analyzer (ILA).
- Read predict IP output valid signal and output data when interrupt signal asserted.
- After read output, clear the interrupt status.



# SW Driver of LeNet-5 (Processing System)

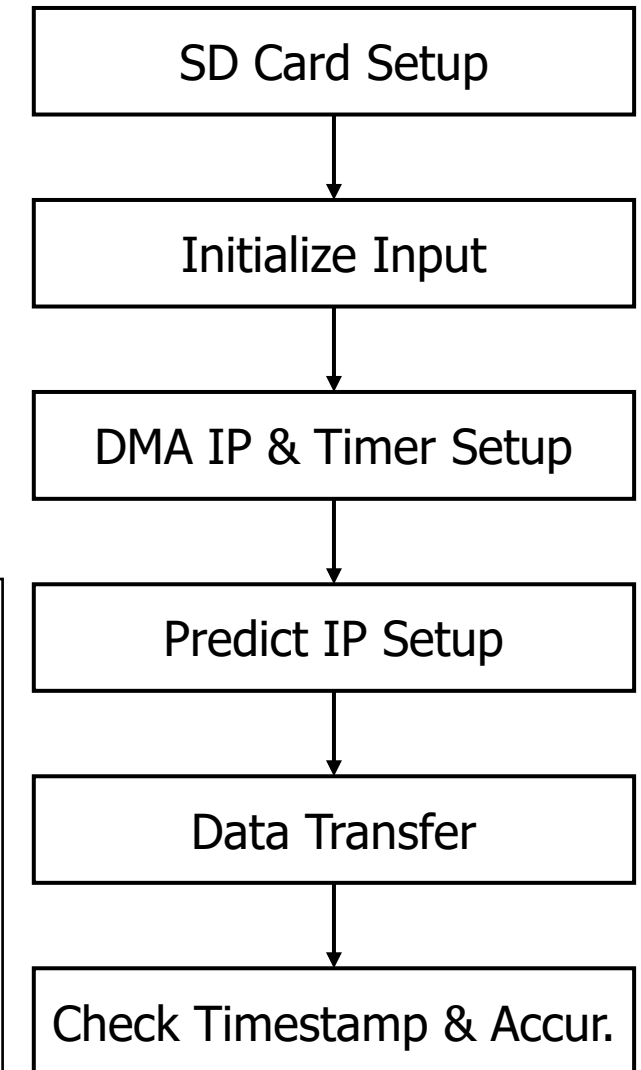
- Read the input binary file stored in SD card and initialize.
- Input is initialized through the data read from the SD card.
- Configure to use DMA IP, AXI Timer IP, and predict IP.
- Before transferring data, cache flush the data to be transferred.
- Data is transferred using `XAxiDma_SimpleTransfer()` Xilinx API.
- Measure DMA transfer latency, predict IP operation latency, etc..
- Check the accuracy to make sure predict IP implemented correctly.

## # UART Terminal output

```
----- LeNet-5 Start -----  
[INFO] SD Init PASSED  
[INFO] Read 0 binary file PASSED  
[INFO] Read 8 binary file PASSED  
[INFO] SD Card Read PASSED  
[INFO] Input initialize PASSED  
[INFO] SD Eject PASSED  
[INFO] DMA initialize PASSED  
[INFO] AXI Timer initialized PASSED  
[INFO] Predict Core initialized PASSED  
...  
[INFO] Timer setup time      : 29 cycles.  
[INFO] Loop iteration time  : 44 cycles. (16)  
...
```

## # UART Terminal output

```
...  
[INFO] Total [HW] run time   : 1742312 cycles.  
[INFO] One sample run time  : 108894 cycles.  
...  
[INFO] Start point          : 27  
[INFO] 0 DMA Tx Start point  : 92  
...  
[INFO] 15 HW ACC Start point : 1231250  
[INFO] 15 HW ACC End point   : 1742310  
...  
[INFO] End point             : 1742412  
...  
[HW Result]  
Accuracy : 100.00 [%]  
Error    : 0 [cases]
```





# Performance analysis, ARM core vs. predict IP, Cont'd

- Result of running the Original model and the Lite model with different compile options on ARM core.

[Original LeNet-5 on ARM core]

No. of input	-O0	-O1	-O2	-O3
1	33.10 <i>ms</i>	6.13 <i>ms</i>	6.12 <i>ms</i>	6.61 <i>ms</i>
4	132.36 <i>ms</i>	24.69 <i>ms</i>	24.25 <i>ms</i>	26.00 <i>ms</i>
16	529.41 <i>ms</i>	96.84 <i>ms</i>	96.86 <i>ms</i>	103.59 <i>ms</i>
64	2116.83 <i>ms</i>	393.79 <i>ms</i>	386.63 <i>ms</i>	414.97 <i>ms</i>
256	8465.10 <i>ms</i>	1576.09 <i>ms</i>	1544.76 <i>ms</i>	1659.93 <i>ms</i>
Average	33.08 <i>ms</i>	6.15 <i>ms</i>	6.05 <i>ms</i>	6.50 <i>ms</i>

⇒ 81%↓ ⇒ 1%↓ ⇒ 7%↑

[Light LeNet-5 on ARM core ]

No. of input	-O0	-O1	-O2	-O3
1	13.15 <i>ms</i>	2.31 <i>ms</i>	1.43 <i>ms</i>	1.30 <i>ms</i>
4	52.56 <i>ms</i>	9.20 <i>ms</i>	5.71 <i>ms</i>	5.18 <i>ms</i>
16	210.33 <i>ms</i>	36.76 <i>ms</i>	22.85 <i>ms</i>	20.68 <i>ms</i>
64	839.48 <i>ms</i>	147.09 <i>ms</i>	91.39 <i>ms</i>	82.66 <i>ms</i>
256	3355.89 <i>ms</i>	587.93 <i>ms</i>	364.92 <i>ms</i>	330.59 <i>ms</i>
Average	13.13 <i>ms</i>	2.30 <i>ms</i>	1.43 <i>ms</i>	1.30 <i>ms</i>

⇒ 82%↓ ⇒ 38%↓ ⇒ 9%↓

## # UART Terminal output

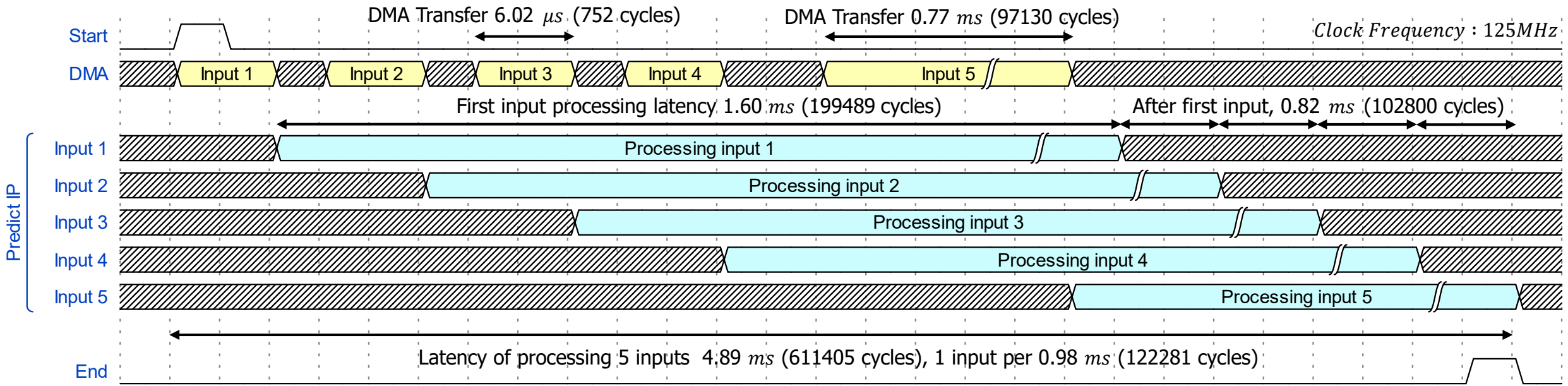
```
----- LeNet-5 Start -----  
...  
[ORG Result]  
Accuracy : 94.55 [%]  
Error    : 545 [cases]  
----- LeNet-5 End -----
```

## # UART Terminal output

```
----- LeNet-5 Start -----  
...  
[Lite Result]  
Accuracy : 97.47 [%]  
Error    : 253 [cases]  
----- LeNet-5 End -----
```

# Performance analysis, ARM core vs. predict IP, Cont'd

- Start and end points are checked through AXI Timer IP.
- For the first input, the latency is high, but it decreases from the second input onwards.
- Predict IP is taking high latency to receive the 5<sup>th</sup> input as it has not yet consumed the data.
- Because `ap_ctrl_chin` block level protocol is applied to predict IP, the 5<sup>th</sup> input is back pressured.
- As the number of inputs increases, the latency per one input will decrease.

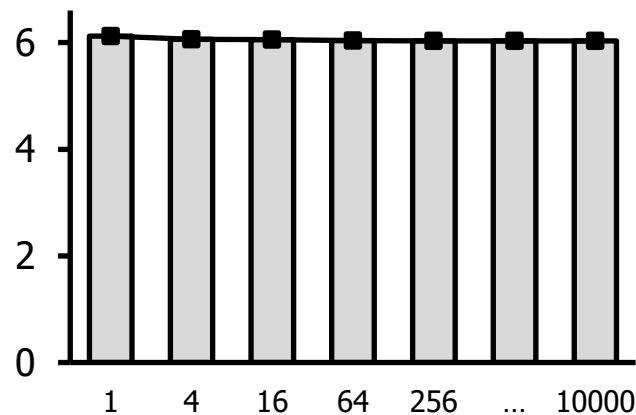


# Performance analysis, ARM core vs. predict IP, Cont'd

- Although the ARM core operates at  $667MHz$  and predict IP operates at  $125MHz$ ,
- Predict IP on programmable logic has the best performance. (**36%** faster than Lite on ARM core)

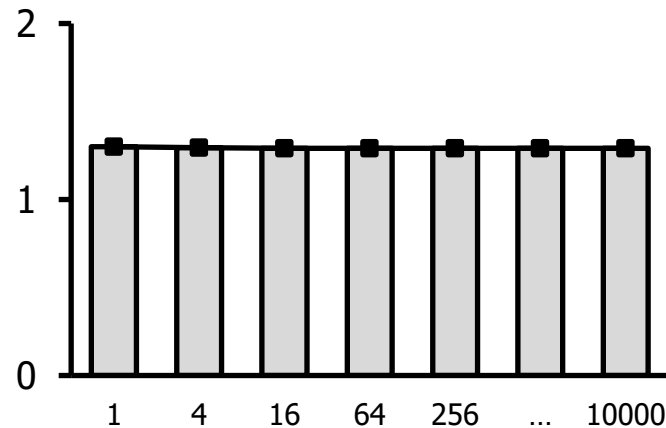
No. of input	-O2 (Org)	Average
1	6.12 ms	6.12 ms
4	24.25 ms	6.06 ms
16	96.86 ms	6.05 ms
64	386.63 ms	6.04 ms
256	1544.76 ms	6.03 ms
...	24699.98 ms	6.03 ms
10000	60323.51 ms	6.03 ms

Original Average (-O2)



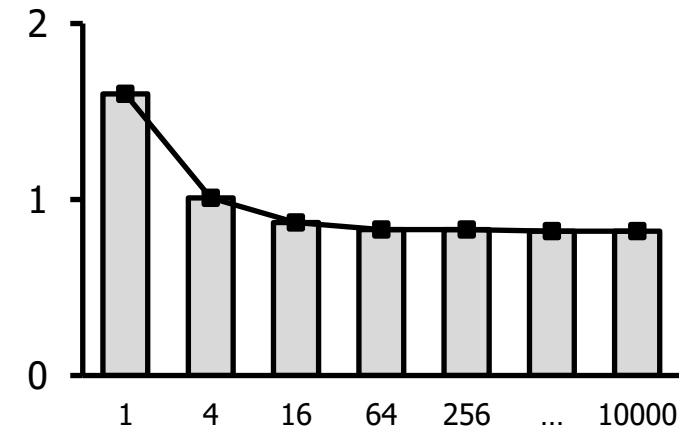
No. of input	-O3 (Lite)	Average
1	1.30 ms	1.30 ms
4	5.18 ms	1.30 ms
16	20.68 ms	1.29 ms
64	82.66 ms	1.29 ms
256	330.59 ms	1.29 ms
...	5290.26 ms	1.29 ms
10000	12915.67 ms	1.29 ms

Lite Average (-O3)



No. of input	-O0 (predict)	Average
1	1.60 ms	1.60 ms
4	4.06 ms	1.01 ms
16	13.93 ms	0.87 ms
64	53.41 ms	0.83 ms
256	211.31 ms	0.83 ms
...	3369.31 ms	0.82 ms
10000	8224.70 ms	0.82 ms

Predict IP

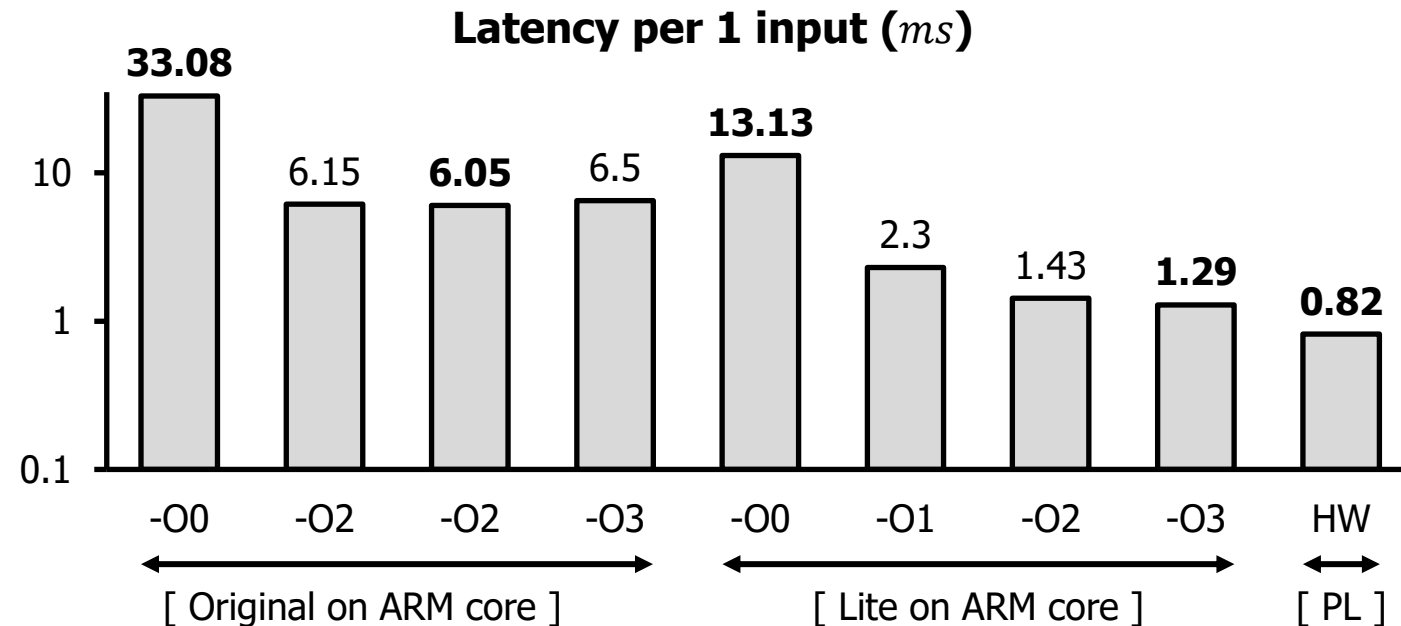


# Performance analysis, ARM core vs. predict IP, Cont'd

- Predict IP on PL is **40.34x** faster than Original with **-O0** compile option on PS.
- Predict IP on PL is **7.38x** faster than Original with **-O2** compile option on PS.
- Predict IP on PL is **16.01x** faster than Lite with **-O0** compile option on PS.
- Predict IP on PL is **1.57x** faster than Lite with **-O3** compile option on PS.

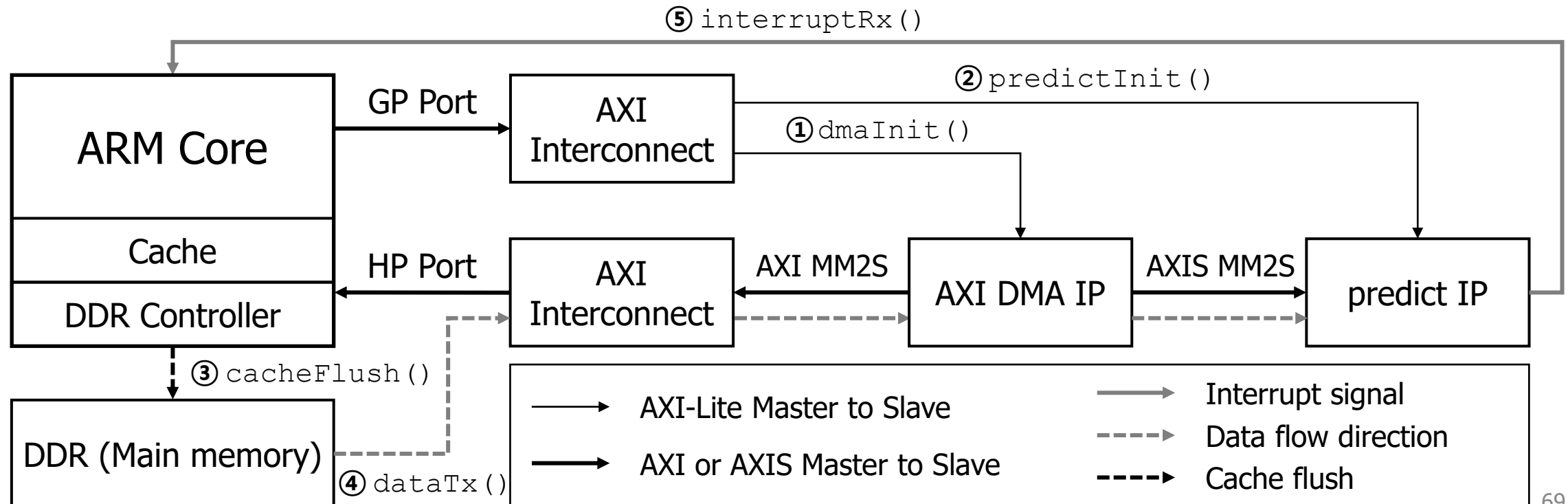
Model	vs. HW
Original (-O0)	<b>98 %</b> ↓
Original (-O1)	87 % ↓
Original (-O2)	<b>86 %</b> ↓
Original (-O3)	87 % ↓
Lite (-O0)	<b>94 %</b> ↓
Lite (-O1)	64 % ↓
Lite (-O2)	42 % ↓
Lite (-O3)	<b>36 %</b> ↓

	-O0	-O1	-O2	-O3	-O0	-O1	-O2	-O3	HW
Average	<b>33.08 ms</b>	6.15 ms	<b>6.05 ms</b>	6.50 ms	<b>13.13 ms</b>	2.30 ms	1.43 ms	<b>1.29 ms</b>	<b>0.82 ms</b>



# Overview of HW Design & SW Driver & Performance

- HW Design
  - Predict IP based on AXI Stream is connected to main memory (DDR) through DMA IP.
- SW Driver
  - When the interrupt signal becomes active HIGH, read the result and measure the latency.
- Performance Analysis
  - Comparing the result of different compile options in PS and the results through predict IP in PL.



# References (Xilinx documents)

- Vivado Design Suite User Guide (High-Level Synthesis), UG902 (v2020.1)
- Vivado Design Suite Tutorial (High-Level Synthesis), UG871 (v2015.4)
- Vivado Design Suite AXI Reference Guide, UG1037 (v4.0)
- Vivado Design Suite User Guide (Programming and Debugging) UG908 (v2022.2)
- AXI DMA (v7.1) LogiCORE IP Product Guide, PG021
- AXI Timer (v2.0) LogiCORE IP Product Guide, PG079
- Zynq-7000 SoC Technical Reference Manual, UG585 (v1.13)
- Zynq-7000 All Programmable SoC Software Developers Guide, UG812 (v12.0)
- A Zynq Accelerator for Floating Point Matrix Multiplication Designed with Vivado HLS, XAPP1170 (v2.0)

# References (etc. )

- The Zynq Book (Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC)
- Parallel Programming for FPGAs (The HLS Book)
- ZynqNet : An FPGA-Accelerated Embedded Convolutional Neural Network
- Eyeriss : A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks
- Deep Learning with Limited Numerical Precision
- Wavedrom (<https://wavedrom.com/>)
- Github (<https://github.com/wasinsangdam/LeNet-5>)

**THANK YOU!**