HLS for Hand-Written Digit Recognition

Group Number: 26

Group member	**Roll Number**
Rupnarayan Kumar	234101046
Sunny Priyadarshi	234101051
Manish Karmakar	234101028
Mukesh Barpete	234101030
Kumar Sanu	234101025

Guided By: **Dr. Chandan Karfa**
Allocated TA: **Manoj Choudhary**

1.Description of the model : -

Task : - This model trains a convolutional neural network (CNN) using the MNIST dataset for handwritten digit recognition.

Dataset :- MNIST training dataset, consisting of 60,000 grayscale images of handwritten digits.

Layer Description : -

- Input Layer: The model takes grayscale images of size 28x28 pixels as input.

- Convolutional Layers :- The convolutional layer has 16 filters of size 3x3, using the ReLU activation function. In the convolutional layer:
  (3 * 3 * 1 + 1) * 16 = 160 parameters.

- Pooling Layer : - A max-pooling layer with a pool size of 2x2 follows the convolutional layer. Here no parameters.

- Flatten Layer :- Flattens the output from the convolutional layers into a 1D vector. Here there are no parameters to learn.

- Dense Layer :- A fully connected layer with 10 units and the softmax activation function, which outputs probabilities for each digit class (0-9).
  In the output dense layer: (13*13*16) * 10 = 27040 parameters

- Output Layer : - generating the probability distribution over the possible digit classes and classify in (0-9)class.

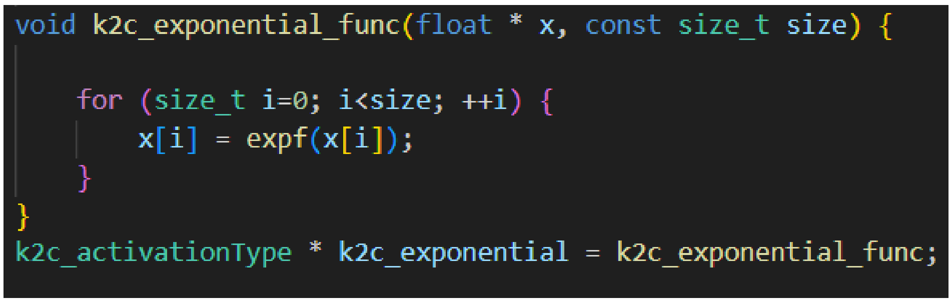
2. Changes made to make keras2c generated files synthesizable and a brief description of the change made.

A. Removed Function's pointer

- In HLS, function pointers can pose challenges because HLS tools need to analyse and synthesise the code into hardware. Function pointers introduce dynamic behaviour, making it harder for the HLS tool to determine the hardware structure and behaviour accurately.

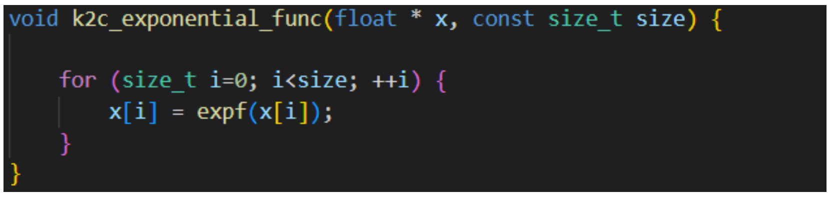
Solution: we have taken the functions inside the main file.

Before:



```
void k2c_exponential_func(float * x, const size_t size) {

    for (size_t i=0; i<size; ++i) {
        x[i] = expf(x[i]);
    }
}
k2c_activationType * k2c_exponential = k2c_exponential_func;
```

After:



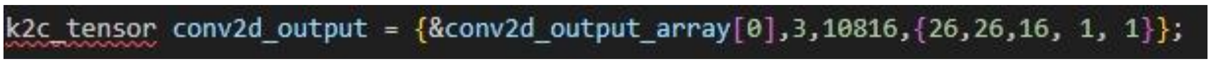
```
void k2c_exponential_func(float * x, const size_t size) {

    for (size_t i=0; i<size; ++i) {
        x[i] = expf(x[i]);
    }
}
```

B. Removed Pointer to Pointer access

- Pointer to Pointer Access is causing the error of dynamic size array which is not supported by vivado synthesis.

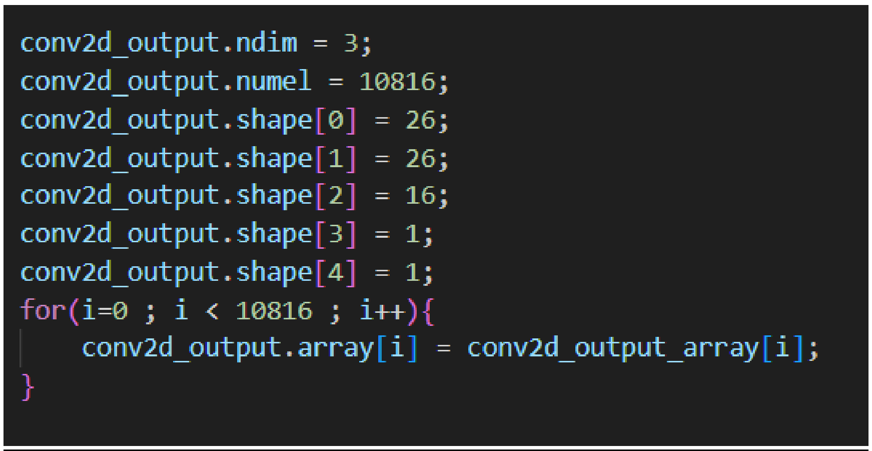
Solution: Flattened the structure shown as below:

Before:

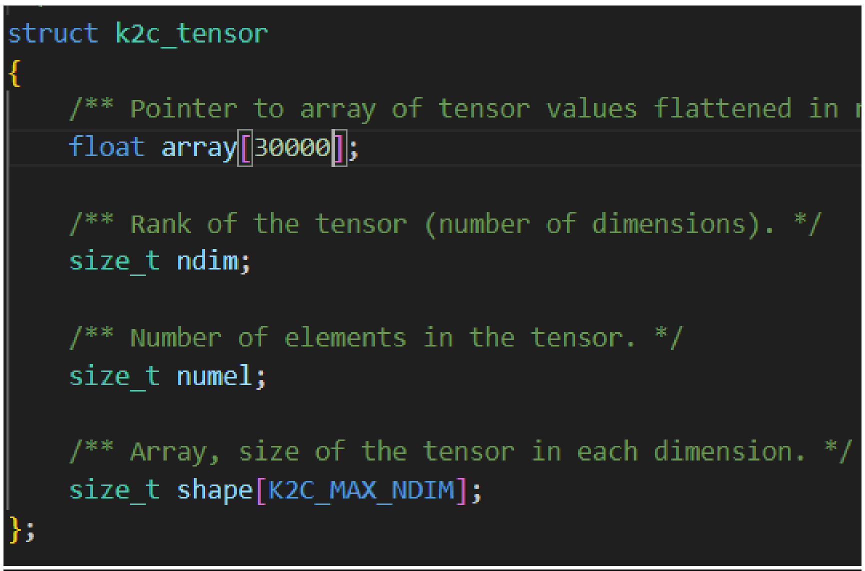


```
k2c_tensor conv2d_output = {&conv2d_output_array[0],3,10816,{26,26,16, 1, 1}};
```

1st Optimization:

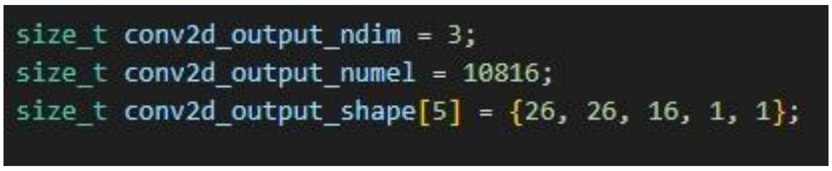


```c
conv2d_output.ndim = 3;
conv2d_output.numel = 10816;
conv2d_output.shape[0] = 26;
conv2d_output.shape[1] = 26;
conv2d_output.shape[2] = 16;
conv2d_output.shape[3] = 1;
conv2d_output.shape[4] = 1;
for(i=0 ; i < 10816 ; i++){
    conv2d_output.array[i] = conv2d_output_array[i];
}
```



```c
struct k2c_tensor
{
    /** Pointer to array of tensor values flattened in r
    float array[30000];

    /** Rank of the tensor (number of dimensions). */
    size_t ndim;

    /** Number of elements in the tensor. */
    size_t numel;

    /** Array, size of the tensor in each dimension. */
    size_t shape[K2C_MAX_NDIM];
};
```

2nd Optimization



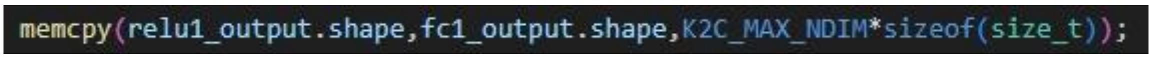
```c
size_t conv2d_output_ndim = 3;
size_t conv2d_output_numel = 10816;
size_t conv2d_output_shape[5] = {26, 26, 16, 1, 1};
```

C. Removed Dynamic Functions(memcpy)

- memcpy() is not directly supported in HLS primarily because it's a library function that involves complex dynamic memory operations, which are challenging to synthesise into hardware efficiently. Here's why memcpy() isn't directly supported in HLS:

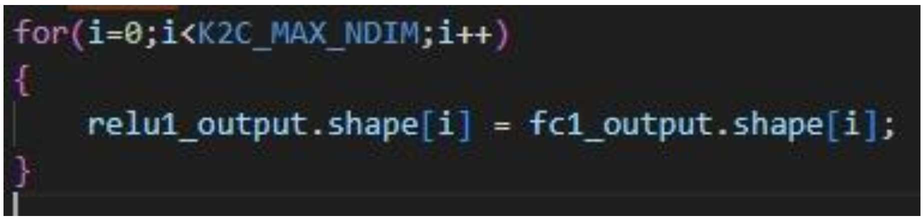
Solution: Rewritten in the form of a loop.

<u>Before:</u>



```
memcpy(relu1_output.shape,fc1_output.shape,K2C_MAX_NDIM*sizeof(size_t));
```

<u>After:</u>



```
for(i=0;i<K2C_MAX_NDIM;i++)
{
    relu1_output.shape[i] = fc1_output.shape[i];
}
```

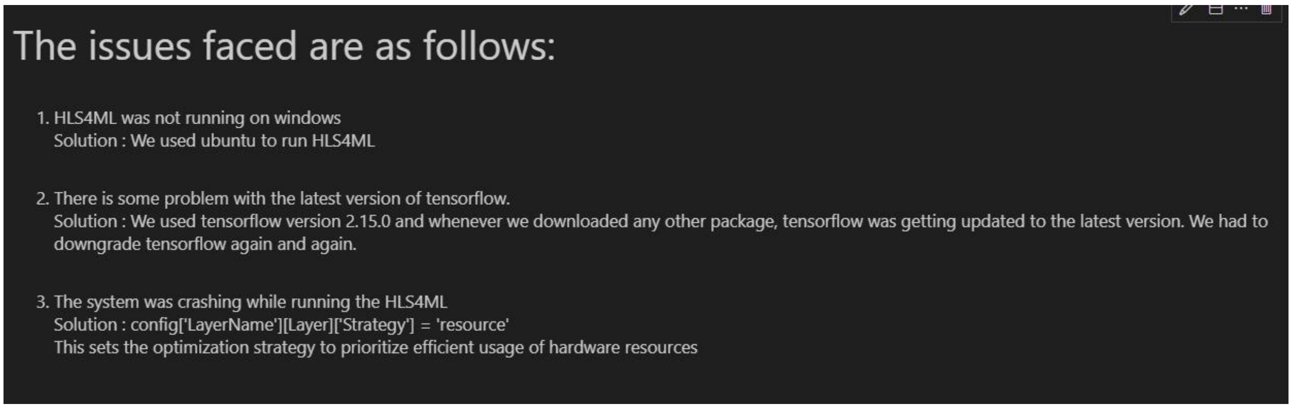
D. Removed Segmentation Fault error while Simulation

Solution: Took all the arrays outside the functions.

3. Changes made to generate HLS4ML report if a pragma is removed in this process. For each of the removed pragma, a valid argument must be mentioned.

Solution: We have not removed any pragmas while generating the HLS4ML report.

4. Mention all the issues that are faced(dependencies and versions) and solutions to resolve.



The issues faced are as follows:

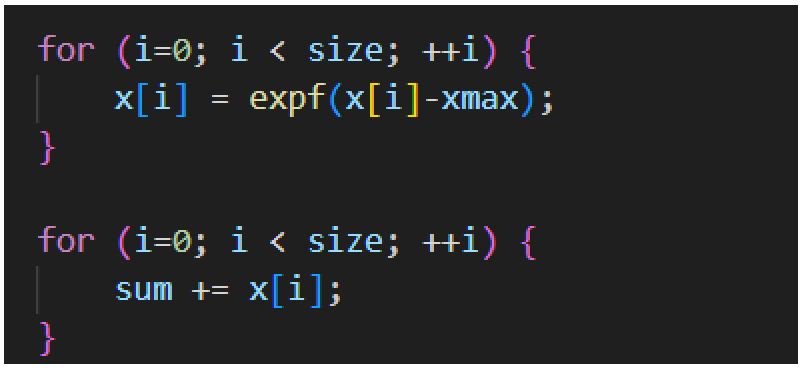
1. HLS4ML was not running on windows
   Solution : We used ubuntu to run HLS4ML

2. There is some problem with the latest version of tensorflow.
   Solution : We used tensorflow version 2.15.0 and whenever we downloaded any other package, tensorflow was getting updated to the latest version. We had to downgrade tensorflow again and again.

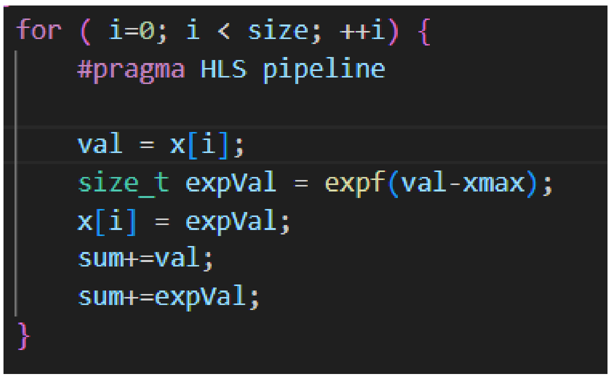
3. The system was crashing while running the HLS4ML
   Solution : config['LayerName'][Layer]['Strategy'] = 'resource'
   This sets the optimization strategy to prioritize efficient usage of hardware resources

5.Optimizations: For each optimization applied (pragma), justify why it has been used.

Loop merging, reduced array access and pipeline.

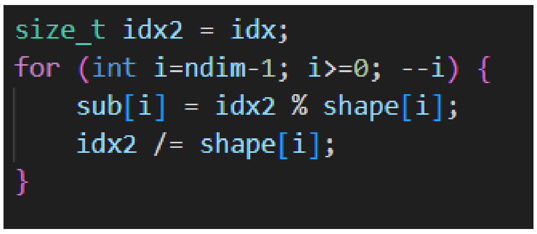


```
for (i=0; i < size; ++i) {
    x[i] = expf(x[i]-xmax);
}

for (i=0; i < size; ++i) {
    sum += x[i];
}
```

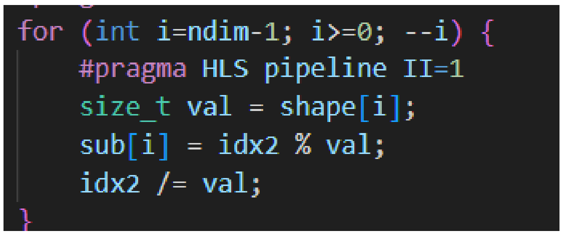


```
for ( i=0; i < size; ++i) {
    #pragma HLS pipeline

    val = x[i];
    size_t expVal = expf(val-xmax);
    x[i] = expVal;
    sum+=val;
    sum+=expVal;
}
```

Reduced array access and applied pipeline

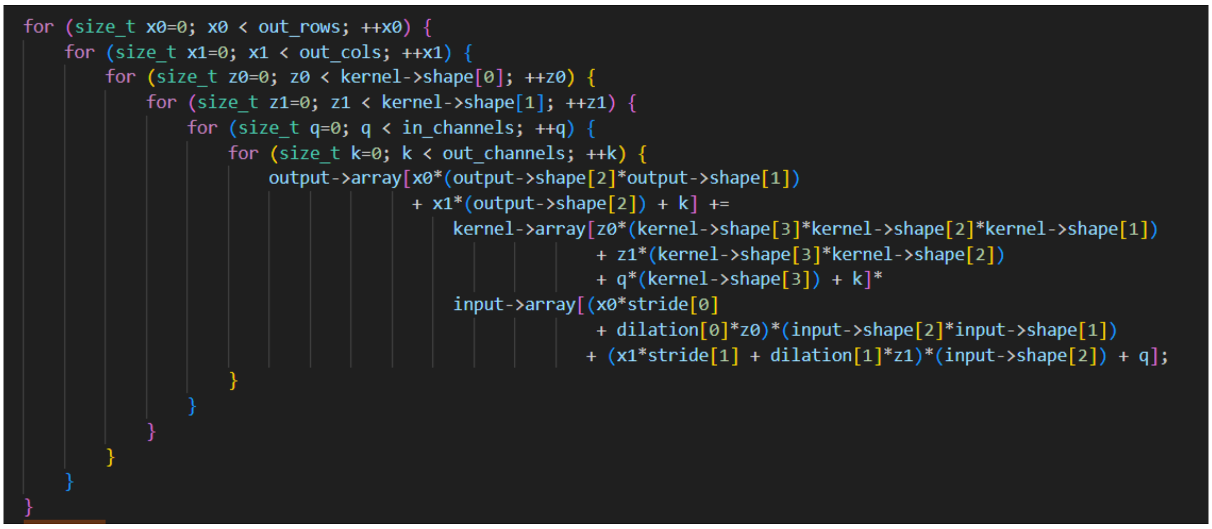


```
size_t idx2 = idx;
for (int i=ndim-1; i>=0; --i) {
    sub[i] = idx2 % shape[i];
    idx2 /= shape[i];
}
```

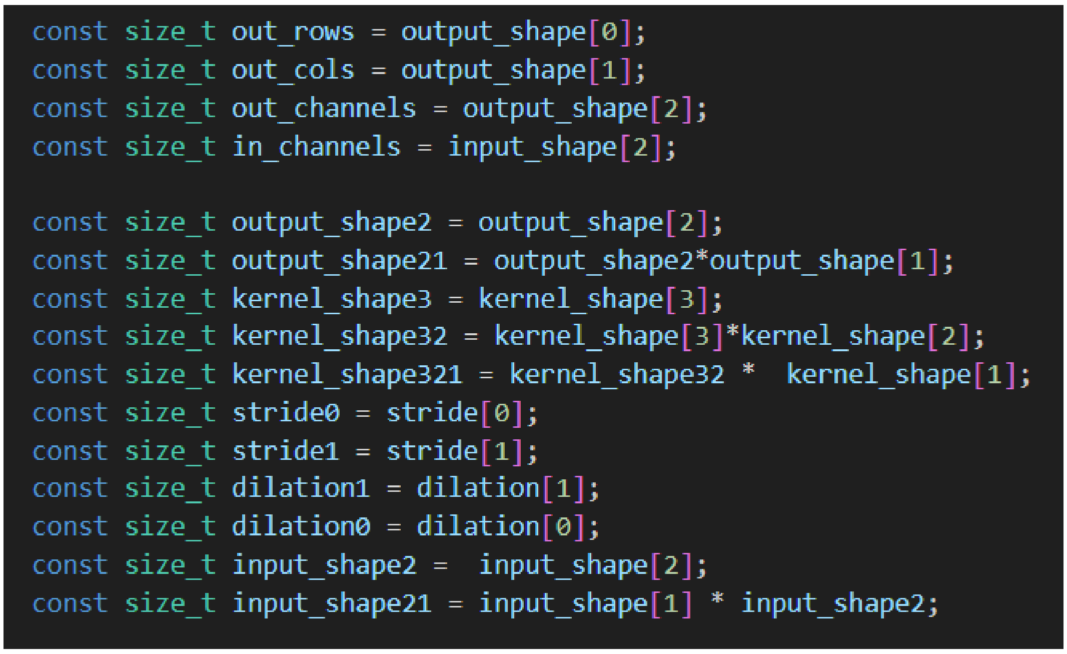


```
for (int i=ndim-1; i>=0; --i) {
    #pragma HLS pipeline II=1
    size_t val = shape[i];
    sub[i] = idx2 % val;
    idx2 /= val;
}
```

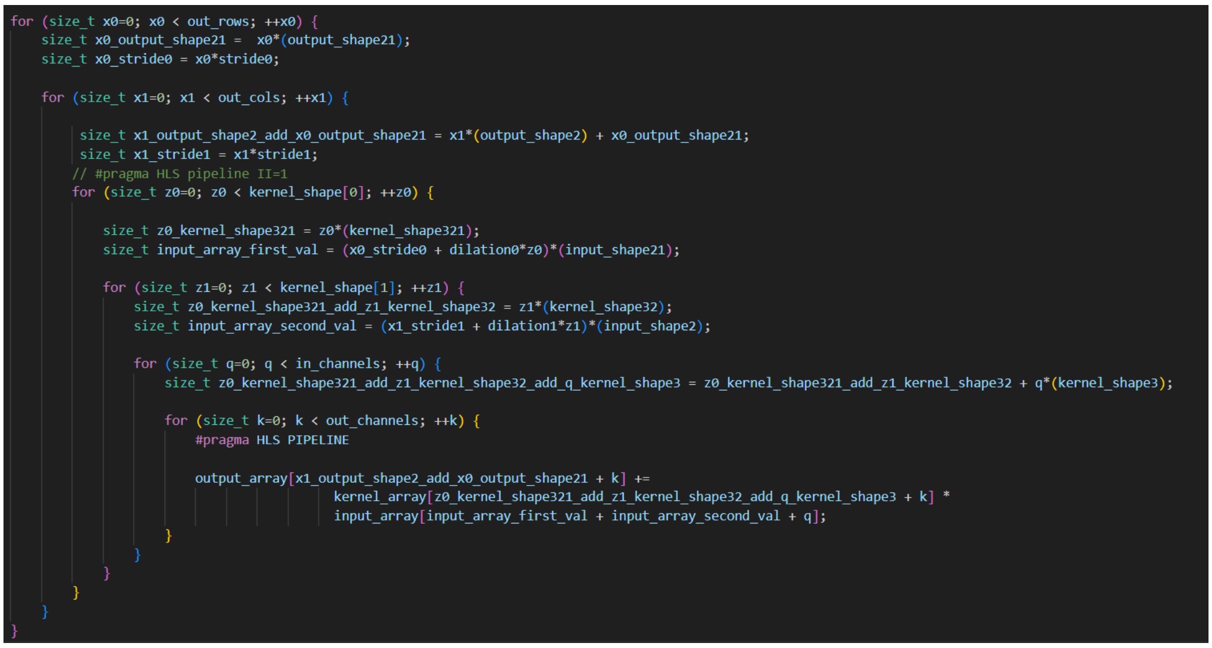
Code Motion



```
for (size_t x0=0; x0 < out_rows; ++x0) {
    for (size_t x1=0; x1 < out_cols; ++x1) {
        for (size_t z0=0; z0 < kernel->shape[0]; ++z0) {
            for (size_t z1=0; z1 < kernel->shape[1]; ++z1) {
                for (size_t q=0; q < in_channels; ++q) {
                    for (size_t k=0; k < out_channels; ++k) {
                        output->array[x0*(output->shape[2]*output->shape[1])
                                   + x1*(output->shape[2]) + k] +=
                                     kernel->array[z0*(kernel->shape[3]*kernel->shape[2]*kernel->shape[1])
                                                  + z1*(kernel->shape[3]*kernel->shape[2])
                                                  + q*(kernel->shape[3]) + k]*
                                     input->array[(x0*stride[0]
                                                  + dilation[0]*z0)*(input->shape[2]*input->shape[1])
                                                 + (x1*stride[1] + dilation[1]*z1)*(input->shape[2]) + q];
                    }
                }
            }
        }
    }
}
```

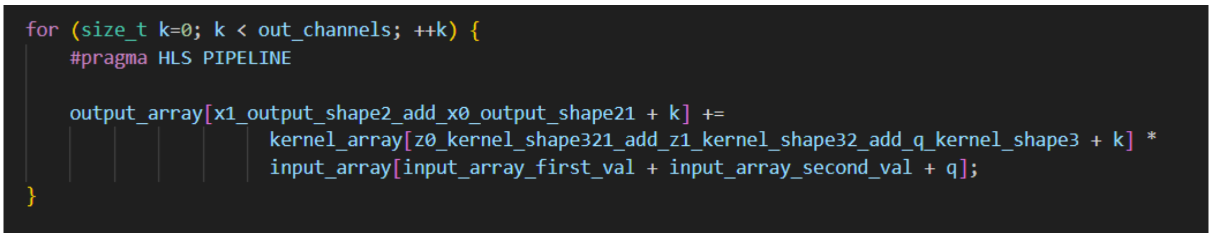


```
const size_t out_rows = output_shape[0];
const size_t out_cols = output_shape[1];
const size_t out_channels = output_shape[2];
const size_t in_channels = input_shape[2];

const size_t output_shape2 = output_shape[2];
const size_t output_shape21 = output_shape2*output_shape[1];
const size_t kernel_shape3 = kernel_shape[3];
const size_t kernel_shape32 = kernel_shape[3]*kernel_shape[2];
const size_t kernel_shape321 = kernel_shape32 *  kernel_shape[1];
const size_t stride0 = stride[0];
const size_t stride1 = stride[1];
const size_t dilation1 = dilation[1];
const size_t dilation0 = dilation[0];
const size_t input_shape2 =  input_shape[2];
const size_t input_shape21 = input_shape[1] * input_shape2;
```



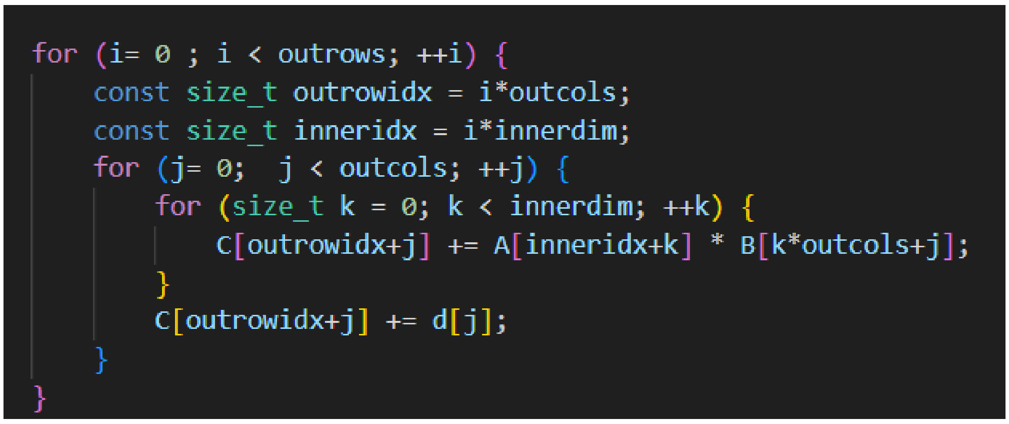
```
for (size_t x0=0; x0 < out_rows; ++x0) {
    size_t x0_output_shape21 =  x0*(output_shape21);
    size_t x0_stride0 = x0*stride0;

    for (size_t x1=0; x1 < out_cols; ++x1) {

         size_t x1_output_shape2_add_x0_output_shape21 = x1*(output_shape2) + x0_output_shape21;
         size_t x1_stride1 = x1*stride1;
        // #pragma HLS pipeline II=1
        for (size_t z0=0; z0 < kernel_shape[0]; ++z0) {

            size_t z0_kernel_shape321 = z0*(kernel_shape321);
            size_t input_array_first_val = (x0_stride0 + dilation0*z0)*(input_shape21);

            for (size_t z1=0; z1 < kernel_shape[1]; ++z1) {
                size_t z0_kernel_shape321_add_z1_kernel_shape32 = z1*(kernel_shape32);
                size_t input_array_second_val = (x1_stride1 + dilation1*z1)*(input_shape2);

                for (size_t q=0; q < in_channels; ++q) {
                    size_t z0_kernel_shape321_add_z1_kernel_shape32_add_q_kernel_shape3 = z0_kernel_shape321_add_z1_kernel_shape32 + q*(kernel_shape3);

                    for (size_t k=0; k < out_channels; ++k) {
                        #pragma HLS PIPELINE

                        output_array[x1_output_shape2_add_x0_output_shape21 + k] +=
                                          kernel_array[z0_kernel_shape321_add_z1_kernel_shape32_add_q_kernel_shape3 + k] *
                                          input_array[input_array_first_val + input_array_second_val + q];
                    }
                }
            }
        }
    }
}
```

And then applied pipeline

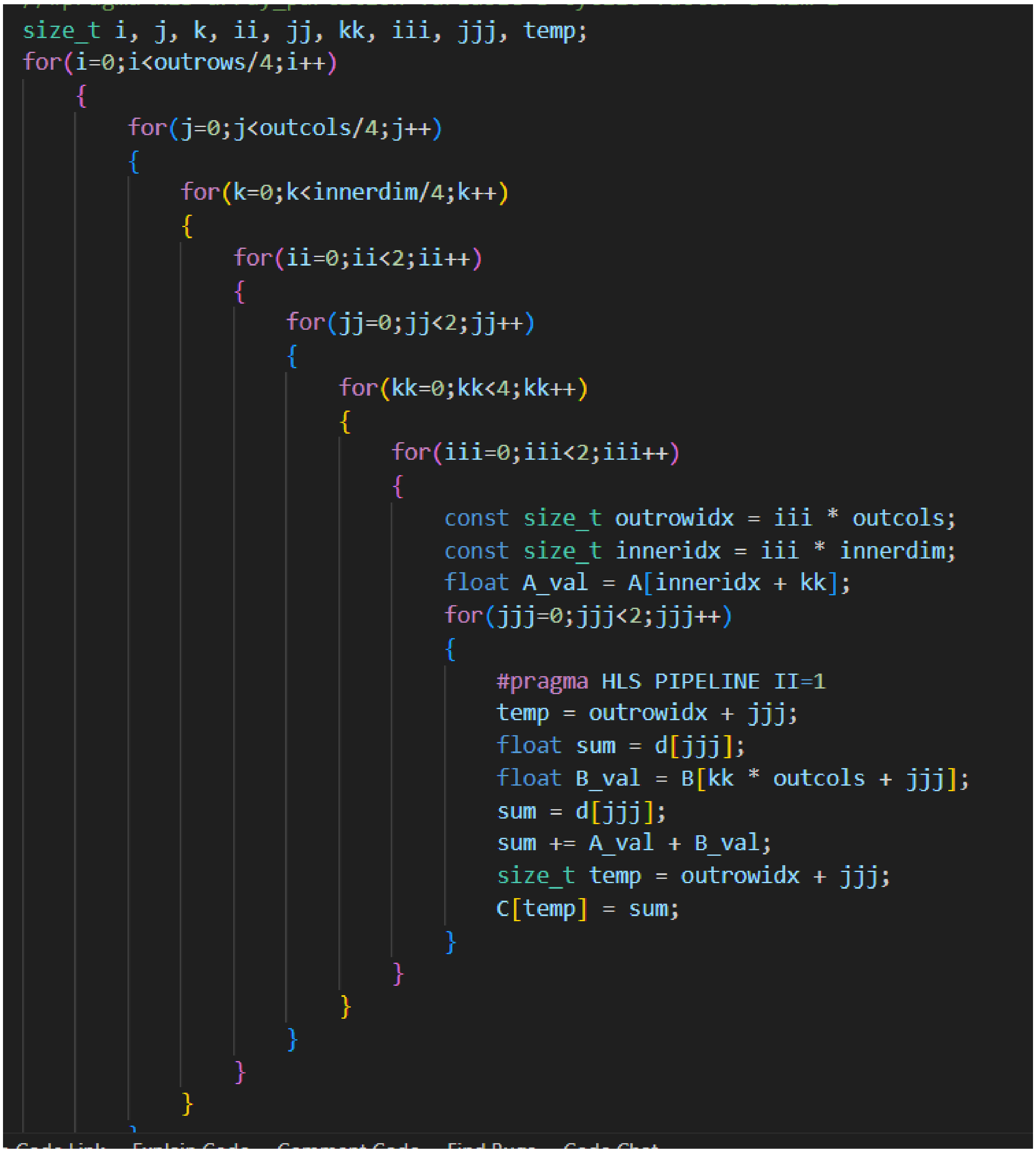


```
for (size_t k=0; k < out_channels; ++k) {
    #pragma HLS PIPELINE

    output_array[x1_output_shape2_add_x0_output_shape21 + k] +=
                      kernel_array[z0_kernel_shape321_add_z1_kernel_shape32_add_q_kernel_shape3 + k] *
                      input_array[input_array_first_val + input_array_second_val + q];
}
```

Loop tiling



```
for (i= 0 ; i < outrows; ++i) {
    const size_t outrowidx = i*outcols;
    const size_t inneridx = i*innerdim;
    for (j= 0;  j < outcols; ++j) {
        for (size_t k = 0; k < innerdim; ++k) {
            C[outrowidx+j] += A[inneridx+k] * B[k*outcols+j];
        }
        C[outrowidx+j] += d[j];
    }
}
```



```
size_t i, j, k, ii, jj, kk, iii, jjj, temp;
for(i=0;i<outrows/4;i++)
    {
        for(j=0;j<outcols/4;j++)
        {
            for(k=0;k<innerdim/4;k++)
            {
                for(ii=0;ii<2;ii++)
                {
                    for(jj=0;jj<2;jj++)
                    {
                        for(kk=0;kk<4;kk++)
                        {
                            for(iii=0;iii<2;iii++)
                            {
                                const size_t outrowidx = iii * outcols;
                                const size_t inneridx = iii * innerdim;
                                float A_val = A[inneridx + kk];
                                for(jjj=0;jjj<2;jjj++)
                                {
                                    #pragma HLS PIPELINE II=1
                                    temp = outrowidx + jjj;
                                    float sum = d[jjj];
                                    float B_val = B[kk * outcols + jjj];
                                    sum = d[jjj];
                                    sum += A_val + B_val;
                                    size_t temp = outrowidx + jjj;
                                    C[temp] = sum;
                                }
                            }
                        }
                    }
                }
            }
```

Inline small functions
a) k2c_relu_function
b) k2c_bias_add
c) k2c_add_bias
d) k2c_flatten

A. K2c_softmax_func
   - Loop1: minimised array access by code motion.

   - Loop2: minimised array access by code motion and applied pipeline. Also merged loop2 and loop3 as it runs same number of iterations.
     Array access and exponential computation takes much clocks , which can be done parallely, so we applied pipeline.

   - Loop4: minimised array access and applied pipeline. Multiplier takes more cycle and assignment takes 1 cycle. So, we can apply pipelining and save clocks.

B. K2c_relu_func
   - Loop1: array access, condition checking then assignment which takes 3 cycles. We can apply pipeline and save latency.

C. K2c_bias_add
   - Loop1: array access, addition and assignment. By applying pipelining we can save latency.

D. K2c_conv2d
   - Loop1: used unroll factor=2. For array assignment

   - Loop2: It has 6 nested loop and inside loops there are much array access and computation and those arrays are not updating. So, we can minimise array access and computation by applying code Motion.

In the innermost loop, we used pipeline to save cycles in computation.

E. K2c_maxpool2d

- There are much array access and computation on array which are not updating so we can save array access and computation by applying code motion. In the inner loop array access, condition checking then assignment which can be parallelized somewhat. So, we applied pipelining.

F. K2c_affine_matmul

- Here matrix multiplication is being calculated. So we can apply loop tiling to increase locality of reference.

G. K2c_idx2sub

- Here we minimised array access and three computations are being done which can be parallelised . So we used pipeline.

H. K2c_sub2idx

- Here array access and two operations are being done. We can parallelise the computation so applied pipeline here

I. K2c_matmul

- Array access and computations are being done. Computations are constant so we applied code motion, and some computations can be parallelised so applied pipelining.

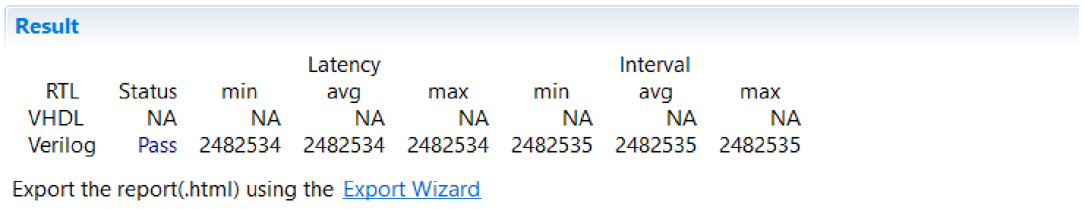
J. K2c_dot

- Here multiple array access and computations were done. So, to parallelise the computation we used pipelining.

Here redundant array access and computations were being done, so we used code motion and pipelining.

6. Results:

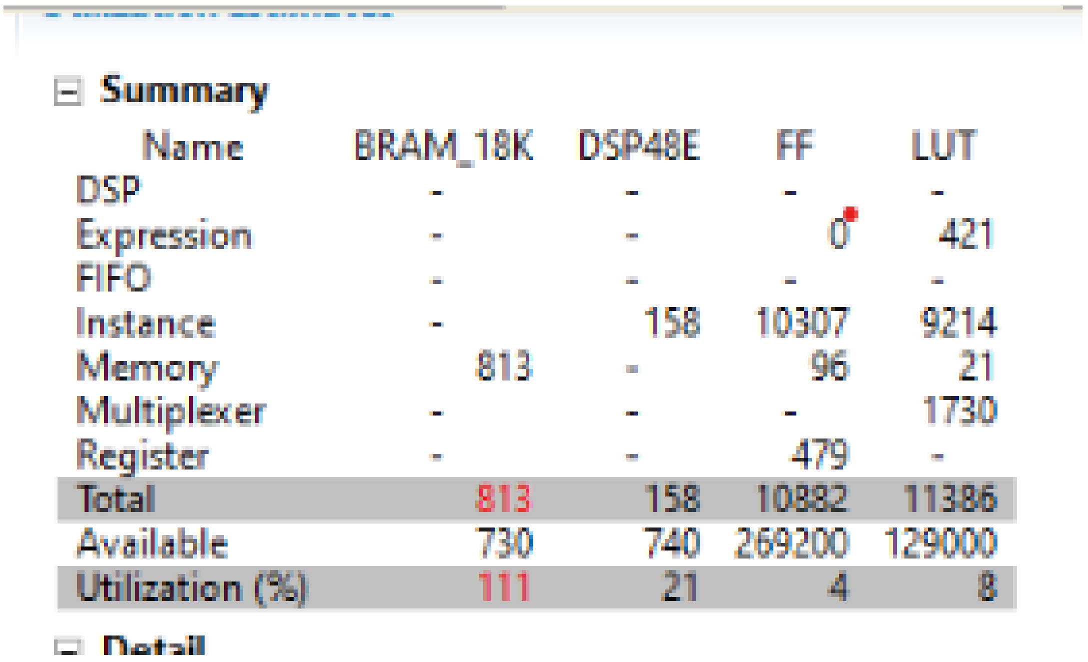
- **Latency and area overhead table for Baseline (Unoptimized).**



Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	2482534	2482534	2482534	2482535	2482535	2482535

Export the report(.html) using the Export Wizard

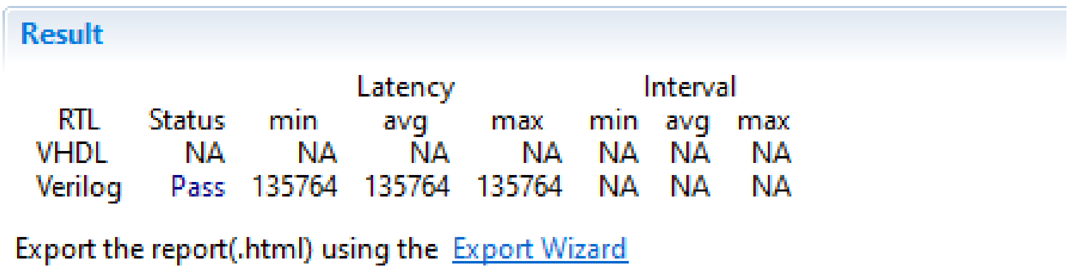


Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	421
FIFO	-	-	-	-
Instance	-	158	10307	9214
Memory	813	-	96	21
Multiplexer	-	-	-	1730
Register	-	-	479	-
Total	813	158	10882	11386
Available	730	740	269200	129000
Utilization (%)	111	21	4	8

Detail

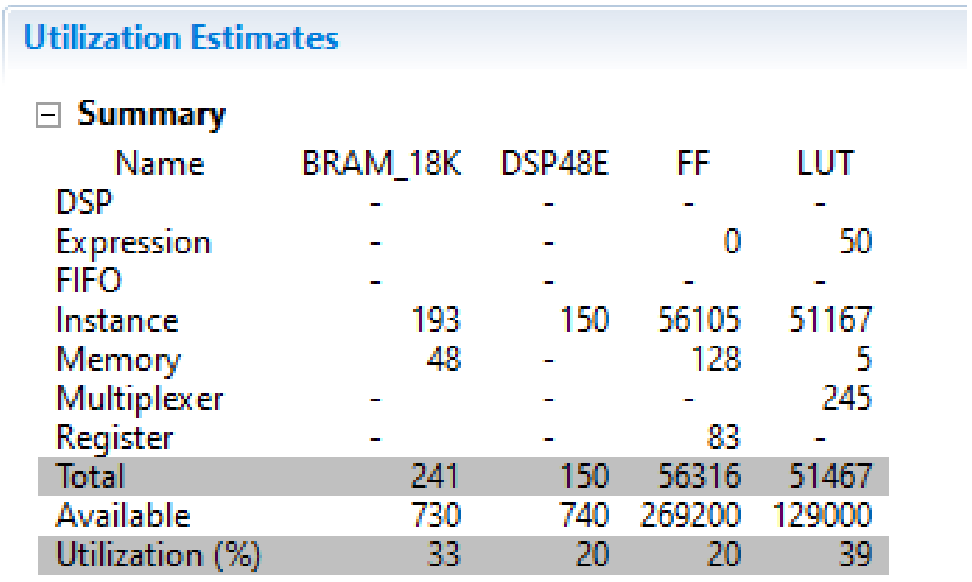
- **Latency and area overhead table for Optimized**



Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	135764	135764	135764	NA	NA	NA

Export the report(.html) using the Export Wizard

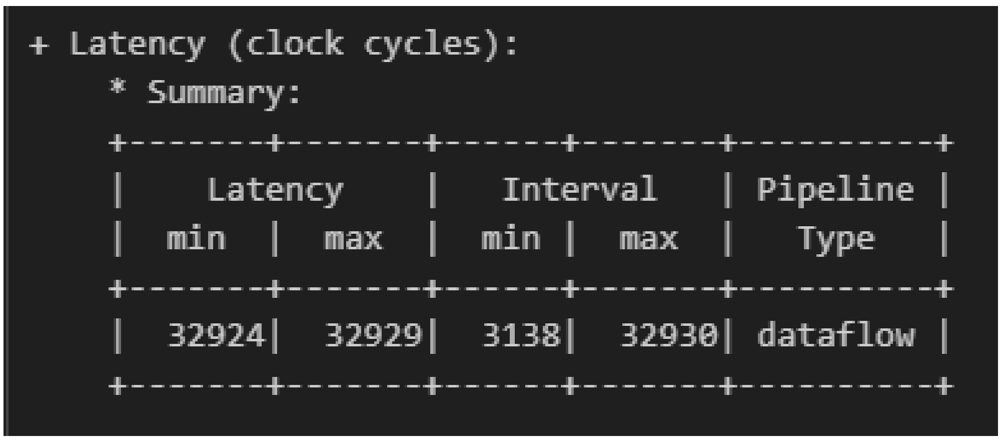


Utilization Estimates

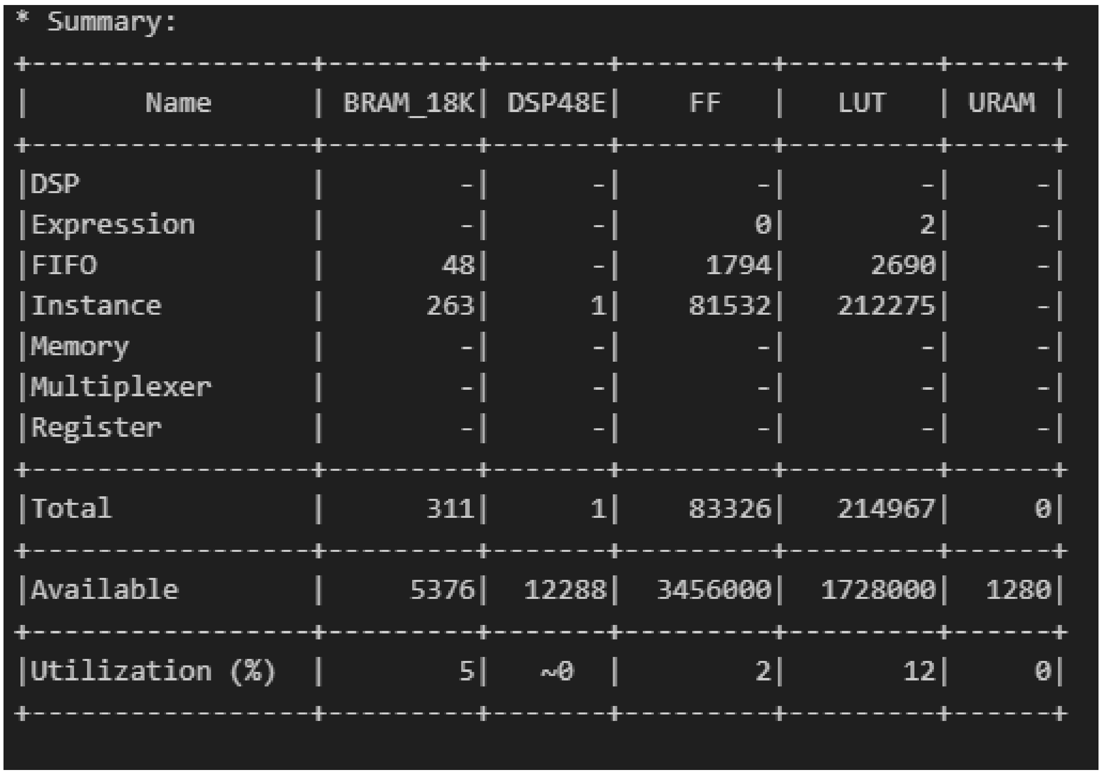
⊟ **Summary**

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	50
FIFO	-	-	-	-
Instance	193	150	56105	51167
Memory	48	-	128	5
Multiplexer	-	-	-	245
Register	-	-	83	-
Total	241	150	56316	51467
Available	730	740	269200	129000
Utilization (%)	33	20	20	39

- **HLS4ML generated Latency and area overhead table.**



```
+ Latency (clock cycles):
    * Summary:
    +-------+-------+------+-------+----------+
    |    Latency    |   Interval   | Pipeline |
    |  min  |  max  |  min |  max  |   Type   |
    +-------+-------+------+-------+----------+
    |  32924|  32929|  3138|  32930| dataflow |
    +-------+-------+------+-------+----------+
```



```
* Summary:
+-----------------+---------+-------+---------+---------+------+
|       Name      | BRAM_18K| DSP48E|    FF   |   LUT   | URAM |
+-----------------+---------+-------+---------+---------+------+
|DSP              |        -|      -|        -|        -|     -|
|Expression       |        -|      -|        0|        2|     -|
|FIFO             |       48|      -|     1794|     2690|     -|
|Instance         |      263|      1|    81532|   212275|     -|
|Memory           |        -|      -|        -|        -|     -|
|Multiplexer      |        -|      -|        -|        -|     -|
|Register         |        -|      -|        -|        -|     -|
+-----------------+---------+-------+---------+---------+------+
|Total            |      311|      1|    83326|   214967|     0|
+-----------------+---------+-------+---------+---------+------+
|Available        |     5376|  12288|  3456000|  1728000|  1280|
+-----------------+---------+-------+---------+---------+------+
|Utilization (%)  |        5|   ~0  |        2|       12|     0|
+-----------------+---------+-------+---------+---------+------+
```

- **Comparison report of both Optimized and HLS4ML generated report.**

HLS4ML Table:

Design	LUT	FF	DSP	BRAM	Latency(min/max)	Clock Period
xcvu13p-flga2577-2-e	214967	83326	1	311	32924 / 32929	5

Vivado Optimized Table:

	Design	LUT	FF	DSP	BRAM	Latency(min/max)	Clock Period/Estimated
Unoptimized	Artix7	11386	10882	158	813	24825534/24825534	10/8.345
Final Optimized	Artix7	51467	56316	150	241	135764 / 135764	5/5.681

Thank You