**HLS for Neural Classification Model**

**Group Number:** **5**

**Group Members:**

Akshay Bhosale - 234101006

Ritik Koshta - 234101044

Rumit Gore - 236101045

Prarbdh Tiwari - 234101038

1. **Description:**

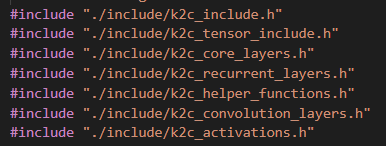
**Model Link**

<https://drive.google.com/file/d/1gCzSk0O1OdZAo-VBt-kccDNIh_j1A0xZ/view?usp=drivesdk>

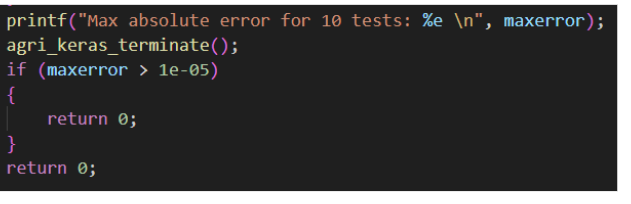
|  |  |  |  |
| --- | --- | --- | --- |
| **Task of Model** | **Number of layers** | **Type of Layers** | **Other Details** |
| To train, and evaluate a neural network model for binary classification using a synthetic dataset. The model's architecture consists of several dense layers with ReLU activation functions, and it is trained using the Adam optimizer and binary crossentropy loss. | 5 | 4 dense layers with ReLU activation functions and 1 dense sigmoid layer as output layer. | Hidden Neurons in the layers:  1st layer: 128  2nd layer: 64  3rd layer: 32  4th layer: 16  5th layer: 1 |

1. **Changes made to make keras2c generated files synthesizable and a brief description of the change made.**
2. It generated 3 files sample.c, sample.h, sample\_test\_suite.c
3. For C Simulation

* All missing functions are included after repetitive iterations (Initially we included total 6 files later we bring the required functions in sample.c file itself).



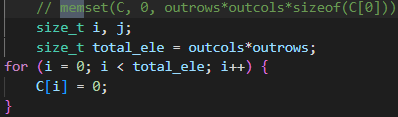
* All redefinition of “size\_t” data type variables errors resolved from all files by declaring the variable one only in the function at top.
* All Segmentation errors are also removed by declaring required array globally.
* All other small errors of missing parenthesis and missing ";" are resolved.
* Returning non-zero value error also resolved by changing the return statement from “1” to “0” in test bench file’s main function.



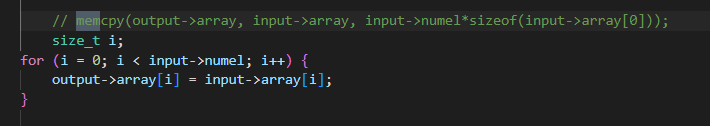
* Simulation succeed after it.

1. For Synthesis Part

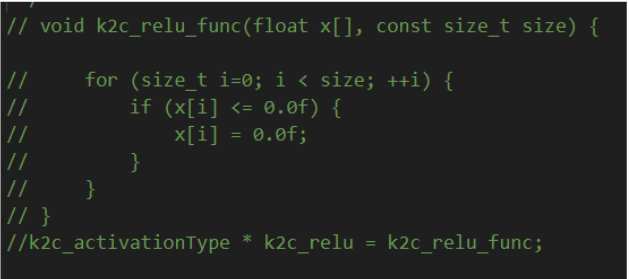
* All memset() functions changed using a loop in all files (took many multiple iterations).

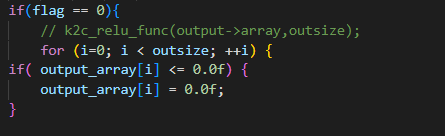


* All memcpy() functions changed using a loop in all files (took many multiple iterations).

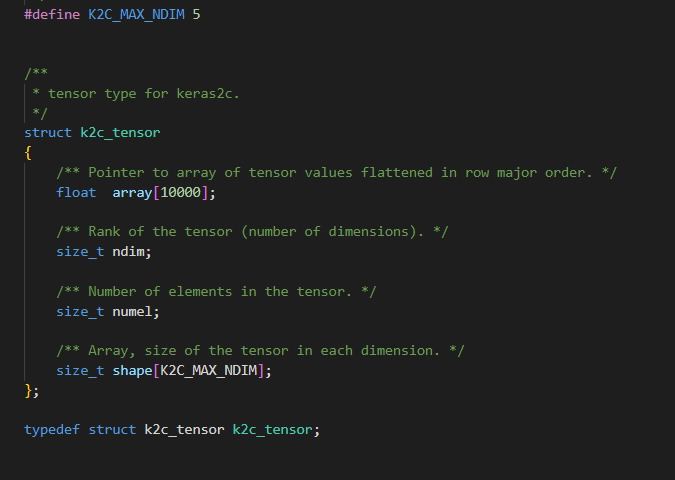


* All function pointers errors are resolved (by calling function directly instead of passing a pointer).

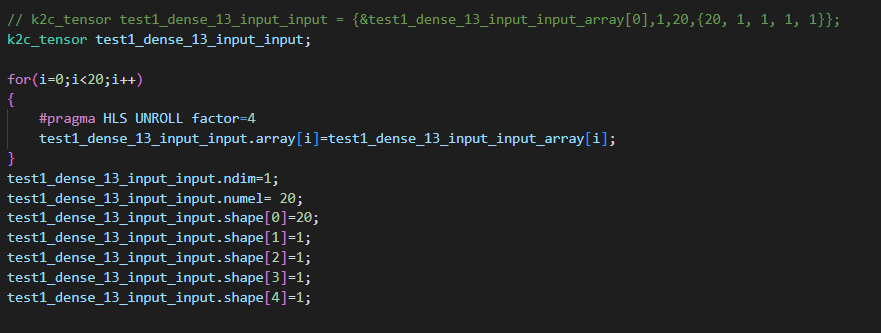


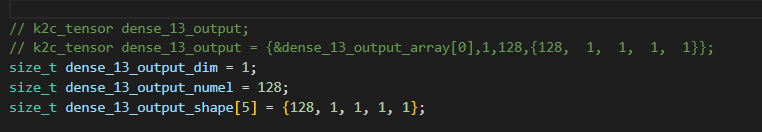


* All double pointers errors are resolved.
* Structure is redefined (Later for resource optimization we didn’t use the structure. We manually created required size arrays for individual instance).

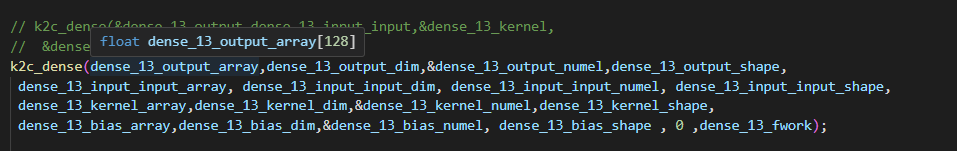


* Structure objects created manually instead of using functions (around 50 such occurrences).





* All dynamic memory allocation converted to static.
* Segmentation fault resolved here also (Array declared Globally).
* All function definition’s also changed.



* Synthesis succeed after it and all required reports generated.

1. C/RTL Simulation Performed after it and we get the Latency.
2. **Changes made to generate HLS4ML reports if a pragma is removed in this process. For each of the removed pragmas, a valid argument must be mentioned.**

We have not removed any pragmas. But we have done some   
 changes to optimise the code.

1. We have removed ***#config\_schedule -enable\_dsp\_full\_reg=false*** from build\_prj.tcl as it was throwing a warning and interrupting the flow.
2. We faced problems while importing hls4ml and we had to downgrade the tensorflow version 14.1 to make hls4ml run.
3. We also faced problems of system crashing again and again due to high memory requirements and less resources available on our system. We used TA’s lab system to run our code for synthesis of HLS4ML.
4. It took around 30 mins to run the synthesis on HLS4ML.
5. We have used the FPGA board as: **xc7a200t-fbg676-2 (artix7)**
6. The configuration of the board is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BRAM\_18K** | **DSP48E** | **FF** | **LUT** | **URAM** |
| 730 | 740 | 269200 | 134600 | 0 |

1. **In a markdown cell of jupyter notebook mention all the issues that are faced(dependencies and versions) and solutions to resolve.**

The dependencies and libraries installed to run and generate the overall   
 HLS4ML reports are saved in the markdown cell of jupyter notebook.   
 The same are presented here:

* pip3 install hls4ml[profiling]
* pip3 install tensorflow==2.15.1
* pip3 install torch torchvision --index-url <https://download.pytorch.org/whl/cpu>
* pip3 install --upgrade pip==22.3
* pip3 install tensorrt
* pip3 install tf\_keras
* pip3 install pydot
* pip3 install pydotplus
* pip3 install graphviz
* !export CUDA\_VISIBLE\_DEVICES=0

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

* **#pragma HLS unroll**

We are using it for unrolling the loop by required factor, which is reducing latency at the cost of resource.

* **#pragma HLS array\_partition**

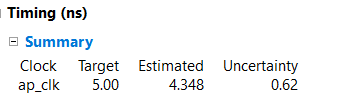
First we have cyclic partitioned with block factor 8 in dimension 1 of the arrays A and B then we have performed unrolling of the loops, which helps in reduction of latency and also we have applied pipeline II=1 in the innermost loop which helps in further reduction in latency.

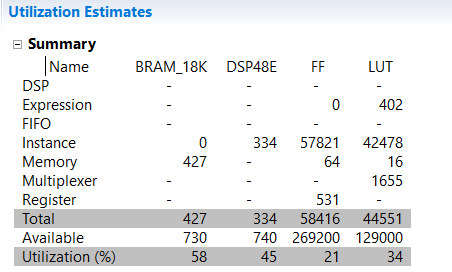
* **#pragma HLS PIPELINE**

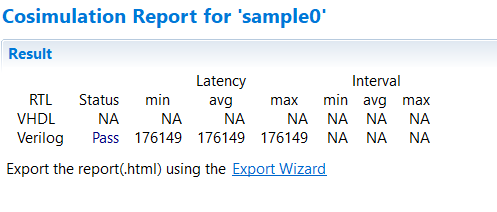
In the code we have applied pipeline II=1 in the inner most loop which helps in reduction of latency.



1. **Results:**
   1. **Latency and area overhead table for Baseline (Unoptimized).**

****

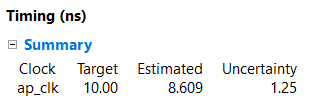
****

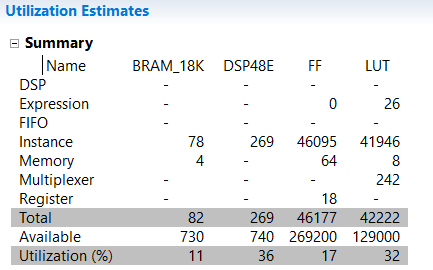
****

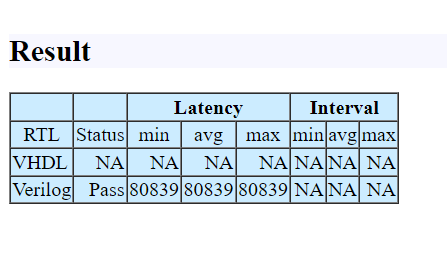
* 1. **Latency and area overhead table for Optimised (if there are**

**multiple versions like various tradeoffs, give all of them).**

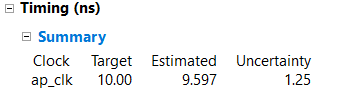
* **Resource Optimized**

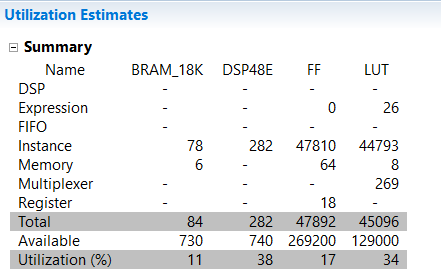
****

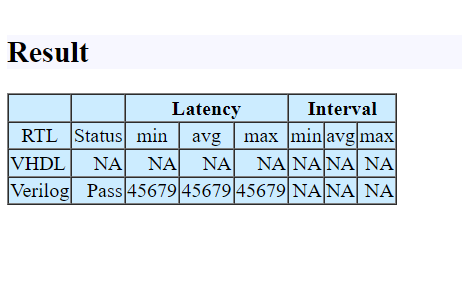
****

****

* **Latency Optimization (version 1)**

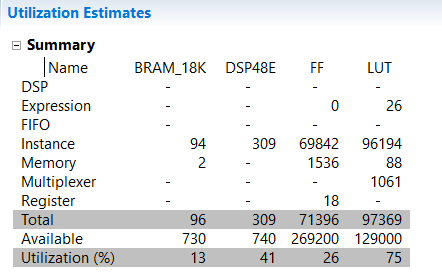
****

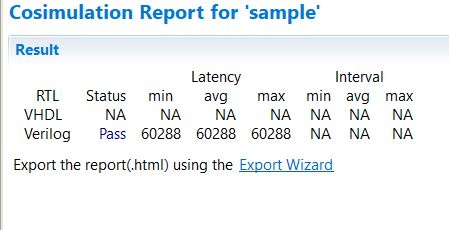
****

****

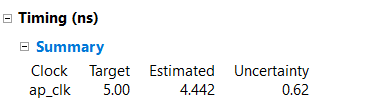
* **Latency Optimization (Version 2)**

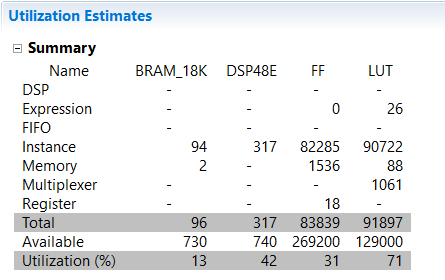
****

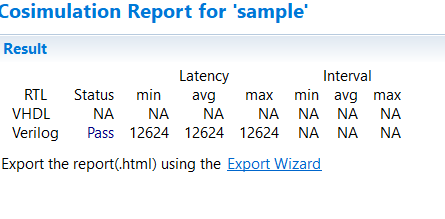
****

****

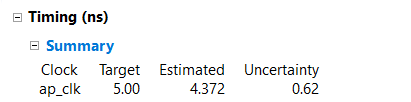
* **Latency Optimization (Version 3)**

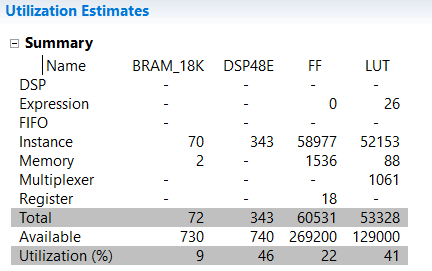
****

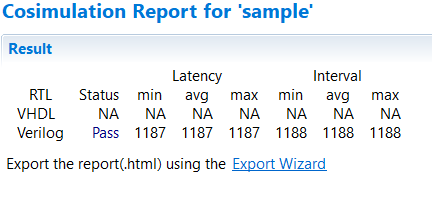
****

****

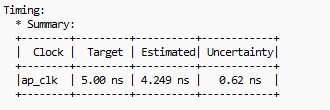
* **Latency Optimization (Final)**

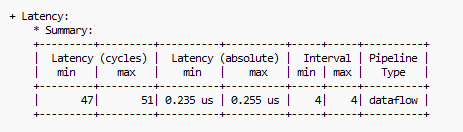
****

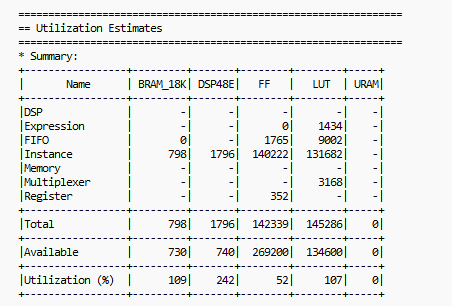
****

****

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

****

****

****

* 1. **Finally, a comparison report of both Optimised and HLS4ML generated reports.**

In Vivado, we have significantly optimized the BRAM, registers, LUTs, flip-flops, and DSPs compared to HLS4ML. This optimization has led to improved resource utilization. However, as a result of these enhancements, there is a slightly higher latency, which we are actively addressing to further enhance overall performance.

In Vivado we have used array partitioning, piplining and unrolling to achieve these results.

In our each try we were trying to bring the results of Vivado and HLS4ML closer and closer to each other. We had tried many ways and mentioned only the ones which worked the best among the many. As we saw an increase in resources we got a lesser latency and vice versa as expected. We have tried to reach the most optimal and balanced result in Vivado.

And hence overall we reached better utilization of resources in Vivado than HLS4ML.

The same can be seen in the table below, where the last row is for HLS4ML.

**Provided the HLS results for each of the above scenarios in the below**

**table.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Design** | **LUT** | **FF** | **DSP** | **BRAM** | **Latency (min/max)** | **Clock Period Estimated** |
| Unoptimized | 44551 | 58416 | 334 | 427 | 176149/  176149 | 4.348 |
| Resource Optimized | 42222 | 46177 | 269 | 82 | 80839/  80839 | 8.609 |
| Unroll | 45096 | 47892 | 282 | 84 | 45679/  45679 | 9.597 |
| Array Partition | 97369 | 71396 | 309 | 96 | 60288/  60288 | 5 |
| Optimised array partition | 91897 | 83839 | 317 | 96 | 12624/  12624 | 4.442 |
| Optimized Matrix Multiplication Tiling | 53328 | 60531 | 343 | 72 | 1187/1188 | 4.372 |
| HLS4ML | 145286 | 142339 | 1796 | 798 | 47/51 | 4.249 |

1. **All the modified keras2c generated files and HLS4ML jupyter note-**

**book must be uploaded in a zip format.**

Done and Submitted.