High Level Synthesis for Digit Recognition Model

Group 7

Siddhartha (210101027) **Koushik** (210101069) **Sai Srinivas** (210101070)
Navadeep (210101072) **Nitish Kumar** (210101125)

---

1 Description of the model

The model chosen has .h5 file of size 1.2 MB. It is trained on MNIST dataset of 60,000 samples, each being an image of 28 x 28 pixels.

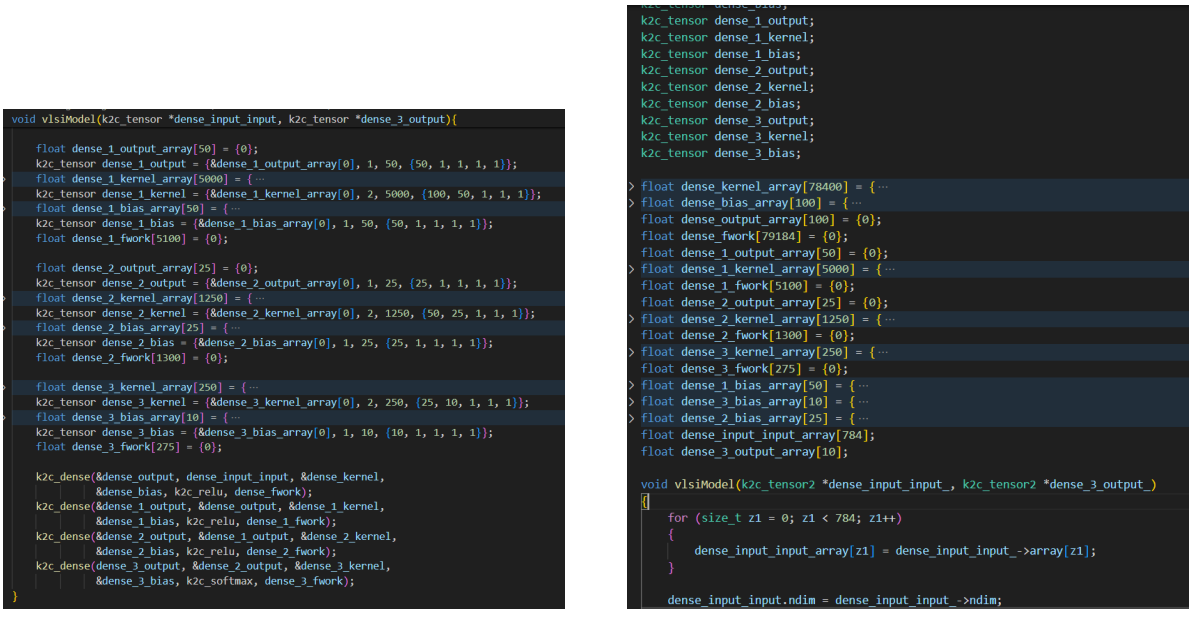
Task of model	Number of layers	Type of Layers	Other details
Multi-class classification of handwritten digits.	4	Dense layers	Activation functions used are relu and softmax

Table 1: Model Specifications

Model link: MNIST Digit Recognition

2 Changes made to keras2c generated files

- As size of the array must be known at compile time for c-synthesis in VivadoHls. The k2ctensor's float* is changed to float array (with maximum size, to keep the array access in limit, here 78400).
- Function pointers are not synthesisable, hence normal c functions are written instead.

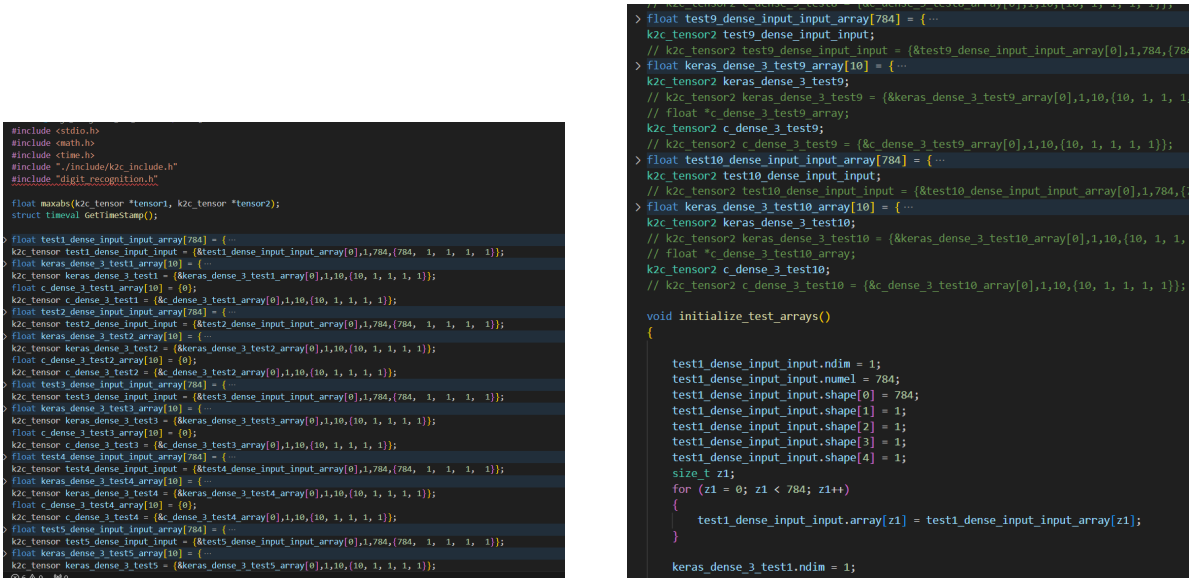


(a) without optimizations

(b) with optimizations

Figure 1: optimizations in keras2c .c files

- All k2tensors are filled with arrays individually, instead of directly passing the arrays pointer to it.Similarly, k2ctensors have been changed in test_suite file also.
- k2ctensors have been declared globally, as the local function memory is not sufficient to store all them. This was done to remove Abnormal program termination error.



(a) without optimizations

(b) with optimizations

Figure 2: optimizations in keras2c test suite

- As same for loop iterator is not allowed, all of them have been removed, and named uniquely.

3 Changes made to generate HLS4ML Report

- It was observed that only a particular python environment is supporting the vivado synthesis and working of hls4ml. We have tried with various environments and the supported versions, the particular environment which resulted in proper execution of hls4ml were python=3.10.10 pydot=1.4.2 scikit-learn=1.2.2 pip=23.0.1 tensorflow=2.11.1.
- The configuration generation and compilation of the given model was succesfull, but the build has shown config_schedule error.This was removed by commenting out a line config_schedule in build_prj.tcl. The line was made for synthesis in vivado 2019.2, but the vivado version we are using is vivado 2018.2.
- The board which was mentioned by TA, was set in project.tcl file and clock period was set to 10ns, for comparision with vivado synthesis of keras2c generated c-files.
- The error observed was while loop unrolling and sufficient memory and computing resources were not available.Hence we have removed
  - `PRAGMA HLS PIPELINE`. This was removed to reduce use of memory.
  - `PRAGMA HLS ARRAY_PARTITION` variable=mult complete. This was removed to avoid error of "Partitioned elements number (78400) has exceeded the threshold (4096), which may cause long run time"
  - `PRAGMA HLS ARRAY_PARTITION` variable=weights complete . This was removed to avoid same error as before.

– `PRAGMA HLS PIPELINE II=CONFIG_T` . This was removed to avoid error of "Stop unrolling loop 'Product1' because it may cause large runtime and excessive memory usage due to increase in code size.

4 Optimisations

In order to enhance the efficiency and performance, several optimizations were implemented. Each modification was chosen carefully to solve specific issues faced during synthesis while also improving resource consumption and execution speed.

- **Memory Optimization**: To mitigate memory consumption during synthesis, adjustments were made to the data structures. The array was removed from the `k2c tensor` structure, and instead, `float *` pointers were utilized wherever applicable. This modification significantly reduced the memory usage, making better use of available resources.
- **Code Motion Optimizations**: Techniques such as matrix multiplication optimization, loop merging, loop invariant removal were employed. These optimizations aimed to streamline the execution flow and minimize redundant computations. By rearranging code segments and eliminating unnecessary operations, the efficiency of the synthesized design was markedly improved.Code optimisations done are-
  - **Loop merging**: As various loops had same loop bound, we merged them. We have tried to fixed the bounds for for loops (so that pipeline can work) , but it didn't change the latency much, hence we didn't include in the final code.
  - **Loop invariant removal**: To avoid recomputations, we have removed loop invariants.
  - **Array accesses reduction**: In situations where only one array access was repeatedly called, the final computaion is stored in a temporary variable and array is accessed only once to update the final value.
- **HLS Pipeline Optimization**: In critical sections like matrix multiplications, the HLS `PIPELINE` pragma was applied. This optimization technique allows for concurrent processing of loop iterations, thereby increasing throughput and reducing latency. By parallelizing computations, the overall performance of the synthesized hardware was enhanced.
- **Array Partitioning**: Large arrays were partitioned using the HLS `ARRAY_PARTITION` pragma. This optimization strategy facilitates efficient memory utilization by distributing array elements across multiple memory banks. By partitioning arrays, the synthesis tool can better optimize memory access patterns, leading to reduced BRAM (Block RAM) utilization and improved resource utilization efficiency.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	52
FIFO	-	-	-	-
Instance	834	537	60965	48432
Memory	40	-	64	5
Multiplexer	-	-	-	964
Register	-	-	107	-
Total	874	537	61136	49453
Available	730	740	269200	129000
Utilization (%)	119	72	22	38

(a) without `ARRAY_PARTITION` pragma

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	52
FIFO	-	-	-	-
Instance	645	552	70538	56320
Memory	40	-	64	5
Multiplexer	-	-	-	964
Register	-	-	107	-
Total	685	552	70709	57341
Available	730	740	269200	129000
Utilization (%)	93	74	26	44

(b) with `ARRAY_PARTITION` pragma

Figure 3: HLS `ARRAY_PARTITION`

5 Results

Design	LUT	FF	DSP	BRAM	Latency (min/max)	Clock period
Baseline	41974	49842	329	870	936630/936630	9.122
Artix 7 - 1	57314	70709	552	685	436380/436380	9.286
Artix 7 - 2	67550	83716	627	591	436380/436380	9.286
HLS4ML	24408	39126	14	239	429892/429892	8.555

*Artix 7 - 2 has been optimized to have less BRAMs compared to Artix 7 - 1 by using ARRAY_PARTITION on dense_fwork array but Artix 7 - 2 has more LUTs, FFs, DSPs compared to Artix 7 - 1.

Design	Compared to	LUT	FF	DSP	BRAM	Latency (min/max)	Clock period
Artix 7 - 1	Baseline	136.54	141.86	167.78	78.73	46.59/46.59	101.79
Artix 7 - 2	Baseline	160.93	167.96	190.57	67.93	46.59/46.59	101.79
Artix 7 - 1	HLS4ML	234.81	180.72	3942.85	286.61	101.50/101.50	108.54
Artix 7 - 2	HLS4ML	276.75	213.96	4478.57	247.28	101.50/101.50	108.54

*The above table shows comparisions of Artix 7 - 1 and Artix 7 - 2 compared to Baseline and HLS4ML. The utilizations are in percentage.From the above table it is clear that latency has been reduced to 46% of baseline.

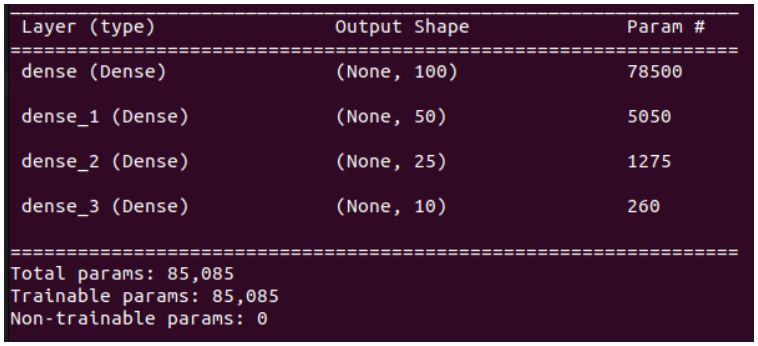


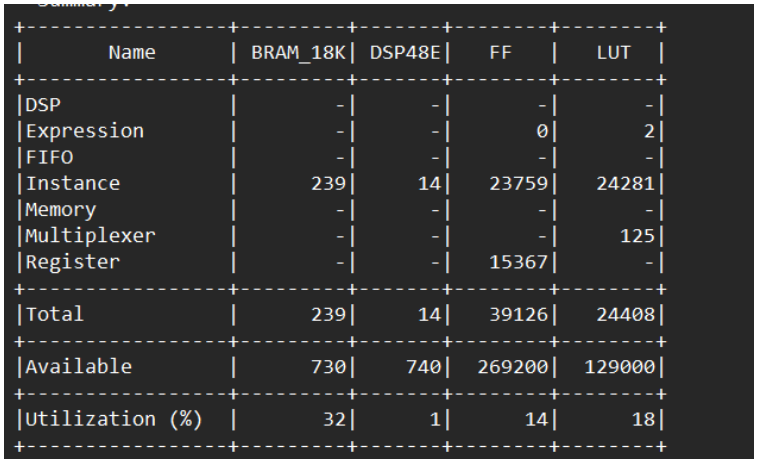
Figure 4: Model Summary

```
    for (size_t i = 0; i < outrows; ++i)
    {
        const size_t outrowidx = i * outcols;
        const size_t inneridx = i * innerdim;
        for (size_t k = 0; k < innerdim; ++k)
        {
            float temp = A[inneridx + k];
            size_t temp2 = k * outcols;
            for (size_t j = 0; j < outcols; ++j)
            {
#pragma HLS PIPELINE
                C[outrowidx + j] += temp * B[temp2 + j];
            }
        }
    }
```

Figure 5: Pipeline pragmas applied are given in the above picture

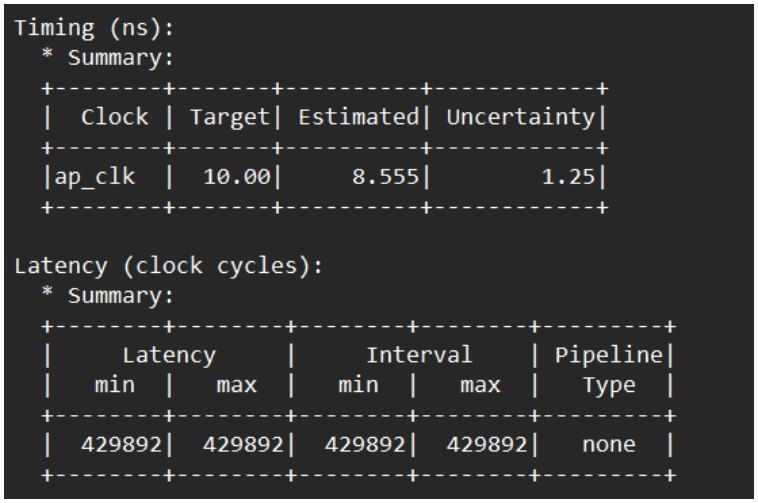
```
#define K2C_MAX_NDIM 5
struct k2c_tensor
{
    /** Pointer to array of tensor values flattened in row major order. */
    // float array[80000];

    /** 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];
};
typedef struct k2c_tensor k2c_tensor;

struct k2c_tensor2
{
    /** Pointer to array of tensor values flattened in row major order. */
    float array[784];

    /** 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];
};
typedef struct k2c_tensor2 k2c_tensor2;
```

Figure 6: Divided tensor structure.



Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2
FIFO	-	-	-	-
Instance	239	14	23759	24281
Memory	-	-	-	-
Multiplexer	-	-	-	125
Register	-	-	15367	-
Total	239	14	39126	24408
Available	730	740	269200	129000
Utilization (%)	32	1	14	18

Figure 7: Resource utilisation observed through hls4ml



Timing (ns):
  * Summary:

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.555	1.25

Latency (clock cycles):
  * Summary:

Latency		Interval		Pipeline
min	max	min	max	Type
429892	429892	429892	429892	none

Figure 8: Timing results obtained through hls4ml