CS 577 : C-based VLSI Design

HLS for Brain Tumour Detection

Group 6

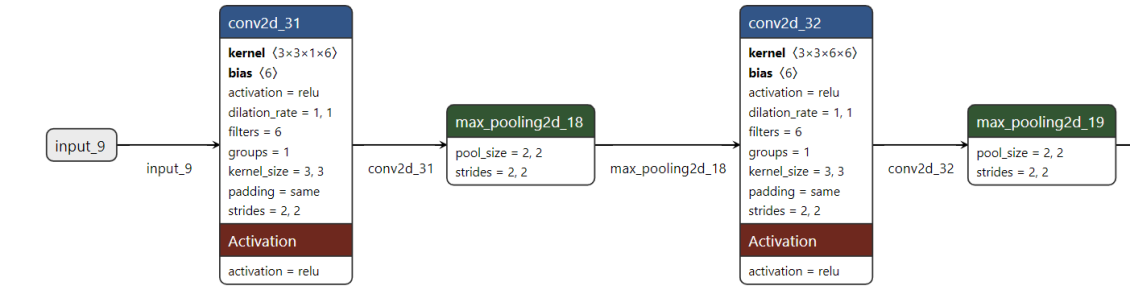
- Abhi Agarwal - 210101002
- Achyut Dhiman - 210101006
- Balaji S - 210101028
- Satyarth Gupta - 210101095
- Hrishikesh - 210102038

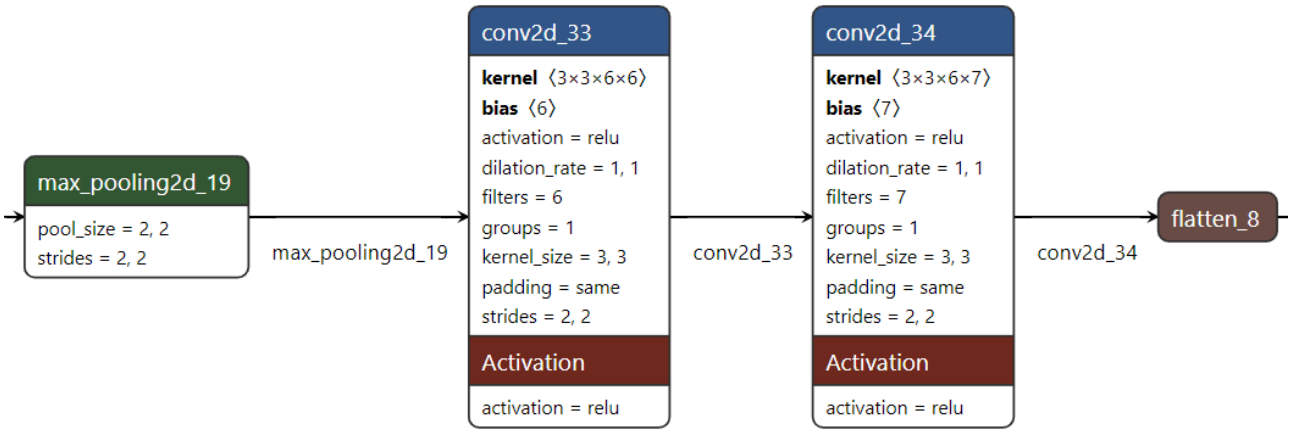
Model Description

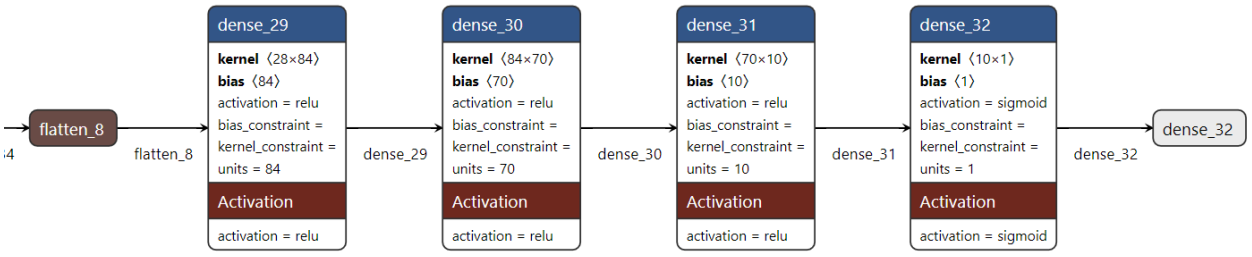
The model we chose was sourced from https://github.com/Sarah-Hesham-2022/Doctork-Application/tree/main and the model chosen was **BrainTumor_CNN.h5**. The project was based on Health Care System with GUI, based on Deep Learning and Images Classification. The model we chose has a goal of detecting MRI of brain and classify them (to brain with & without tumor). The model was moderately sized (**190 KB**) and had **8 layers**.

Task of model	No. of Layers	Type of Layers	Other Details
Brain Tumour Detection	8	1.Padding (2D) 2.Convolution (2D) 3.MaxPooling (2D) 4.Flatten (1D) 5.Dense (Fully-Connected)	This CNN model for Brain Tumour detection comprises 8 layers, including 4 2D convolutional layers with padding and max-pooling, followed by a flattening layer and 4 fully connected dense layers. The model processes the input through convolutional operations, downsamples with max-pooling, and uses dense layers for classification, aiming to detect brain tumours based on the provided data.

Model Structure- Brain Tumour







Changes made to HLS4ML model

The hls4ml was used to create firmware implementations of machine learning algorithms. HLS4ML was used to convert keras model to HDL; suitable for Vivado High Level Synthesis.

We used compatible tensorflow version 2.13.1 to run the model.

We used following script to run HLS4ML on our model and build it using Vivado HLS.

```
import os
new_path = '/opt/Xilinx/Vivado/2018.2/bin'
os.environ['PATH'] += os.pathsep + new_path

from tensorflow.keras.models import load_model
import hls4ml
model = load_model('BrainTumor_CNN.h5')
config = hls4ml.utils.config_from_keras_model(model, granularity='name')
config['Model']['ReuseFactor'] = 4
#config['Model']['Strategy'] = 'resource'
#config['Model']['Precision'] = 'fixed<>'

hls_model = hls4ml.converters.convert_from_keras_model(model, hls_config=
config , io_type = 'io_stream', part = 'xc7z020clg400-1' , backend = 'Vivado')
# hls_model.compile()
hls_model.build(csim = False)


# Print out the report if you want
hls4ml.report.read_vivado_report('my-hls-test')
```

The script first updates the system PATH to include the Vivado toolchain directory, then loads the Keras model, configures conversion parameters, and finally performs the conversion and synthesis. It generated the output folder my-hls-test which had all required C++ files; utility libraries and configuration files for HLS synthesis. We faced many build issues; which required visiting the documentation exhaustively and also trying various hit and trial changes. Problematic pragmas were identified through exhaustive search; by first removing each and every pragma, then adding them till all problematic

pragmas were found. In the end, to successfully build the model, following changes were made.

1. Commenting my-hls-test/build_prj.tcl line 167

In **build_prj.tcl** the line 167 corresponds to following directive

config_schedule -enable_dsp_full_reg=false

In Vivado 2018.2, it shows the following error

ERROR: [HLS 200-101] 'config_schedule': Unknown option '-enable_dsp_full_reg=false'.

Upon reviewing through 2018.2 Vivado Design Suite User Guide:High-level Synthesis (UG902); we found the only options for config_schedule is -effort. Enable_dsp_full_reg is an option in Vivado 2023.2 (UG1399). The default there is false for this option. So in build_prj.tcl we commented this particular pragma; and re-run the code.

Options

`-effort (high|medium|low)`

Specifies the effort used during scheduling operations.

- The default is `Medium` effort.
- A `Low` effort optimization improves the run time and might be useful when there are few choices for the design implementation.
- A `High` effort optimization results in increased run time, but typically provides better results.

`-verbose`

Prints out the critical path when scheduling fails to satisfy any directives or constraints.

`-relax_ii_for_timing`

This option allows scheduling to relax the II on a pipelined loop or function in order to satisfy timing requirements. In general, scheduling might create a design that fails to meet timing, allowing logic synthesis to be used to ensure the timing requirements are met. This option informs scheduling to always meet timing and relax the throughput target (II) in order to ensure the design meets its timing requirements.

Pragma

There is no pragma equivalent.

Examples

Changes the default schedule effort to `Low` to reduce run time.

```
config_schedule -effort low
```

2. Commenting line 30 of my-hls-test/firmware/nnet_utils/nnet_dense_latency.h

The line 30 of **nnet_dense_latency.h** is
#pragma HLS ARRAY_PARTITION variable=mult complete
This was commented out because of following error

ERROR: [XFORM 203-103] Array 'mult.V' (firmware/nnet_utils/nnet_dense_latency.h:17): partitioned elements number (5880) has exeeded the threshold (4096), which may cause long run-time.

Other solutions were tried as well like changing ARRAY_PARTITION type from complete to block (factor 2), cyclic but the error still persisted.

The exact cause of this error seems to be the size of weights of layer dense_30; which has 5880 elements. The maximum allowed number of partitions was 4096 according to default configurations.

3. Commenting line 18 of my-hls-test/firmware/nnet_utils/nnet_dense_stream.h

The line 18 of **nnet_dense_stream.h** corresponds to
#pragma HLS PIPELINE II=CONFIG_T::reuse_factor

It belongs to dense_wrapper; the code snippet of which is shown here.

```
template <class data_T, class res_T, typename CONFIG_T>
void dense_wrapper(data_T data[CONFIG_T::n_in], res_T res[CONFIG_T::n_out],
          typename CONFIG_T::weight_t weights[CONFIG_T::n_in *
CONFIG_T::n_out],
          typename CONFIG_T::bias_t biases[CONFIG_T::n_out]) {
  #pragma HLS INLINE recursive
  if (CONFIG_T::strategy == nnet::latency) {
    // #pragma HLS PIPELINE II=CONFIG_T::reuse_factor
    dense_latency<data_T, res_T, CONFIG_T>(data, res, weights, biases);
  } else {
    dense_resource<data_T, res_T, CONFIG_T>(data, res, weights, biases);
  }
}
```

It seems like the dense_wrapper invokes dense_latency; which is called inline recursively with pipelining which somehow causes enormous number of load/store instructions. This results in following set of warnings and errors

```
WARNING: [ANALYSIS 214-1] Tool encounters 11760 load/store instructions to
analyze which may result in long runtime.
ERROR: [XFORM 203-1403] Unsupported enormous number of load/store
instructions: 'nnet::dense_wrapper<ap_fixed<16, 6, (ap_q_mode)5, (ap_o_mode)3,
0>, ap_fixed<16, 6, (ap_q_mode)5, (ap_o_mode)3, 0>, config15>' .
ERROR: [HLS 200-70] Failed building synthesis data model.
```

Also 11760 happens to be double of 5880 which is length of weights of layer dense_30; which might happen to cause this issue., because of huge number of elements to load.

4. Increasing clock _period in my-hls-test/project.tcl

We increased clock period to 10 ns as earlier the default target of 5 ns was being exceed by estimated clock period.

```
    + Timing (ns):
        * Summary:
        +--------+-------+----------+------------+
        |  Clock | Target| Estimated| Uncertainty|
        +--------+-------+----------+------------+
        |ap_clk  |   5.00|     5.421|        0.62|
        +--------+-------+----------+------------+
```

HLS4ML generated latency and area overhead table

```
================================================================
== Vivado HLS Report for 'myproject'
================================================================
* Date:           Tue Apr 23 20:35:34 2024

* Version:        2018.2 (Build 2258646 on Thu Jun 14 20:25:20 MDT 2018)
* Project:        myproject_prj
* Solution:       solution1
* Product family: zynq
* Target device:  xc7z020clg400-1


================================================================
== Performance Estimates
================================================================
+ Timing (ns):
  * Summary:
  +--------+-------+----------+------------+
  |  Clock | Target| Estimated| Uncertainty|
  +--------+-------+----------+------------+
  |ap_clk  | 10.00|    9.353|        0.62|
  +--------+-------+----------+------------+

+ Latency (clock cycles):
  * Summary:
  +--------+--------+-------+--------+----------+
```

```
    |   Latency   |   Interval  | Pipeline |
    |  min |  max | min |  max |   Type   |
    +--------+--------+-------+--------+----------+
    | 202971| 202971| 40810| 191000| dataflow |
    +--------+--------+-------+--------+----------+

================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+-------+
|       Name      | BRAM_18K| DSP48E|   FF   | LUT |
+-----------------+---------+-------+--------+-------+
|DSP              |        -|     -|      -|     -|
|Expression       |        -|     -|      0|     2|
|FIFO             |      100|     -|   4220|  14383|
|Instance         |       27|   275|  43190|  74316|
|Memory           |        -|     -|      -|     -|
|Multiplexer      |        -|     -|      -|     -|
|Register         |        -|     -|      -|     -|
+-----------------+---------+-------+--------+-------+
|Total            |      127|   275|  47410|  88701|
+-----------------+---------+-------+--------+-------+
|Available        |      280|   220|  106400|  53200|
+-----------------+---------+-------+--------+-------+
|Utilization (%)  |       45|   125|     44|   166|
+-----------------+---------+-------+--------+-------+
```

Design	LUT	FF	DSP	BRAM	Latency(min/max)	Clock period
HLS4ML	88701	47410	275	127	202971/202971 clock cycles	9.353 est. 10 target

Changes made to Keras2c generated files

1) In struct k2c Tensor, the array pointer was changed so as to represent array[MAX_SIZE] (MAX_SIZE=30,000)
2) Removed Function Pointers using flags since they are not synthesizable in HLS.

```
if(flag==0){
    k2c_softmax_func(output->array, output->numel);
}
if(flag==1){
 k2c_relu_func(output->array, output->numel);

}
```

```
k2c_dense(&dense_29_output,&flatten_8_output,&dense_29_kernel,
    &dense_29_bias,1,dense_29_fwork);
k2c_dense(&dense_30_output,&dense_29_output,&dense_30_kernel,
    &dense_30_bias,1,dense_30_fwork);
k2c_dense(&dense_31_output,&dense_30_output,&dense_31_kernel,
    &dense_31_bias,1,dense_31_fwork);
k2c_dense(dense_32_output,&dense_31_output,&dense_32_kernel,
    &dense_32_bias,0,dense_32_fwork);
 }
```

3) Memcpy was removed
   memcpy(&output->array[offset], &input->array[i*num], num*sizeof(input->array[0]))
   Instead copy using for loop is implemented

```
for (size_t ji = 0; ji < num; ji++) {
        output->array[offset + ji] = input->array[i * num + ji];
 }
```

4) Memset was removed for similar reasons
   memset(output->array,0,output->numel*sizeof(output->array[0]))
   Was replaced by

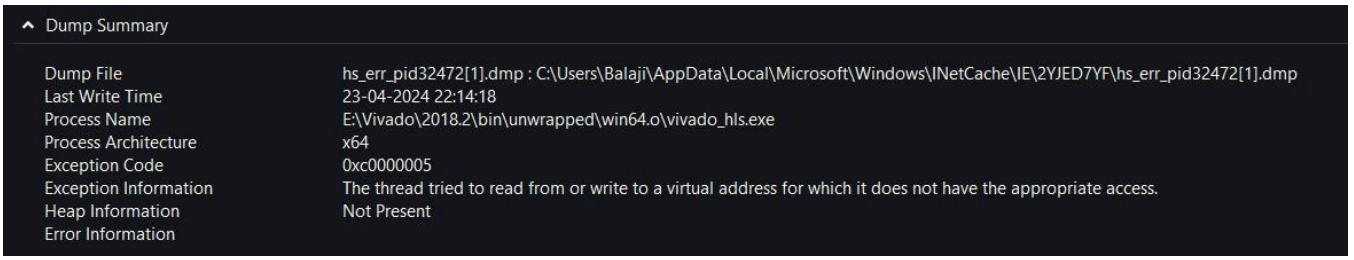
```
size_t i;
for ( i = 0; i < output->numel; ++i) {
        output->array[i] = 0.0f; // Set each element to zero
```

```
}
```

5) The structure initialization was unfolded due to type mismatch between float* and float

```
 dense_32_kernel.ndim=2;
dense_32_kernel.numel=10;
dense_32_kernel.shape[0]=10;
dense_32_kernel.shape[1]=1;
dense_32_kernel.shape[2]=1;
dense_32_kernel.shape[3]=1;
dense_32_kernel.shape[4]=1;
for(i=0;i<10;i++){
    #pragma HLS unroll factor=4
        dense_32_kernel.array[i] = dense_32_kernel_array[i];
}
```

6) Abnormal program termination was handled by increasing the size of arrays , which was caused by out of bounds access



7) Also, some basic initial changes (versions and type) were made to use keras2C.. To generate the C file malloc==false was used, since malloc and related system calls are not supported by HLS., Tensorflow version 2.15.0 and numpy version 1.26.4 were used.

Changes made to optimize the Resource Utilisation and Latency

1) Some of the larger arrays were removed and instead a direct initialization of the arrays inside the k2c Tensor structure was done.
2) Output array in k2c_conv2d was already initialised to 0, then there is no sense in re-initialising inside the k2c_conv2d function. We removed the initialization that was outside and only the initialization iniside remains
3) The same changes were done to the output k2c_pad2d.
4)

```
for (size_t i=0; i < size; ++i) {
    #pragma HLS unroll factor=4
    if (x[i] <= 0.0f) {
```

```
        x[i] = 0.0f;
    }
}
```

```
Report time       : Tue Apr 23 17:20:53 IST 2024.
Solution          : solution1.
Simulation tool   : xsim.

+----------+----------+-----------------------------------------------+-----------------------------------------------+
|          |          |                    Latency                    |                    Interval                   |
+   RTL    +  Status  +-----------------------------------------------+-----------------------------------------------+
|          |          |      min      |      avg      |      max      |      min      |      avg      |      max      |
+----------+----------+-----------------------------------------------+-----------------------------------------------+
|      VHDL|        NA|             NA|             NA|             NA|             NA|             NA|             NA|
|   Verilog|      Pass|        1226411|        1226411|        1226411|             NA|             NA|             NA|
+----------+----------+-----------------------------------------------+-----------------------------------------------+
with unroll factor=4
```

Pragma unroll factor 4 was used wherever possible which was the optimal. Unroll factor 16 lead to an increase in resource utilization while at the same time not causing much change to the latency.

```
Report time       : Tue Apr 23 17:44:32 IST 2024.
Solution          : solution1.
Simulation tool   : xsim.

+----------+----------+-----------------------------------------------+-----------------------------------------------+
|          |          |                    Latency                    |                    Interval                   |
+   RTL    +  Status  +-----------------------------------------------+-----------------------------------------------+
|          |          |      min      |      avg      |      max      |      min      |      avg      |      max      |
+----------+----------+-----------------------------------------------+-----------------------------------------------+
|      VHDL|        NA|             NA|             NA|             NA|             NA|             NA|             NA|
|   Verilog|      Pass|        1220701|        1220701|        1220701|             NA|             NA|             NA|
+----------+----------+-----------------------------------------------+-----------------------------------------------+
with unroll factor=16
```

5)

```
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) {
        for (size_t j = 0;  j < outcols; ++j) {
          #pragma HLS pipeline
           C[outrowidx+j] += A[inneridx+k] * B[k*outcols+j];
        }
      }
   }
```

Pragma HLS pipeline was used in the innermost loop wherever possible since this leads to reduction in latency

6)

```
for(i=0;i<10;i++){
   #pragma HLS unroll factor=4
        dense_31_bias.array[i] = dense_31_bias_array[i];
}
```

For small loops direct loop unroll was used whereas

```
for(i=0;i<700;i++){
   #pragma HLS unroll factor=4
   #pragma HLS PIPELINE II=1
         dense_31_kernel.array[i] = dense_31_kernel_array[i];
}
```

For large loops unroll with pipeline is used.

Initial Reports

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1351
FIFO	-	-	-	-
Instance	0	157	35156	30245
Memory	1771	-	512	43
Multiplexer	-	-	-	4934
Register	-	-	940	-
Total	1771	157	36608	36573
Available	5040	2880	5037120	2518560
Available SLR	1680	960	1679040	839520
Utilization (%)	35	5	~0	1
Utilization SLR (%)	105	16	2	4

```
Report time       : Sat Apr 20 20:40:20 IST 2024.
Solution          : solution1.
Simulation tool   : xsim.

+----------+----------+-----------------------------------------+-----------------------------------------+
|          |          |                 Latency                 |                 Interval                |
+   RTL    +  Status  +-----------------------------------------+-----------------------------------------+
|          |          |     min     |     avg     |     max     |     min     |     avg     |     max     |
+----------+----------+-----------------------------------------+-----------------------------------------+
|      VHDL|        NA|           NA|           NA|           NA|           NA|           NA|           NA|
|   Verilog|      Pass|      2493405|      2493405|      2493405|      2493406|      2493406|      2493406|
+----------+----------+-----------------------------------------+-----------------------------------------+
```

Results

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1606
FIFO	-	-	-	-
Instance	86	356	66302	69049
Memory	889	-	416	33
Multiplexer	-	-	-	5573
Register	-	-	1652	-
Total	975	356	68370	76261
Available	5040	2880	5037120	2518560
Available SLR	1680	960	1679040	839520
Utilization (%)	19	12	1	3
Utilization SLR (%)	58	37	4	9

```
Report time       : Tue Apr 23 17:20:53 IST 2024.
Solution          : solution1.
Simulation tool   : xsim.

+----------+----------+-----------------------------------------+-----------------------------------------+
|          |          |                 Latency                 |                 Interval                |
+   RTL    +  Status  +-----------------------------------------+-----------------------------------------+
|          |          |      min      |      avg      |      max      |      min      |      avg      |      max      |
+----------+----------+-----------------------------------------+-----------------------------------------+
|      VHDL|        NA|             NA|             NA|             NA|             NA|             NA|             NA|
|   Verilog|      Pass|        1226411|        1226411|        1226411|             NA|             NA|             NA|
+----------+----------+-----------------------------------------+-----------------------------------------+
with unroll factor=4
```

Comparison

Design	Unoptimized	Optimised	HLS4ML
LUT	36573	76261	88701
FF	36608	68370	47410
DSP	157	365	275
BRAM	1771	975	127
latency	2493405	1226411	202971
Clock Period	8.44	10.493	9.353

Conclusion

In conclusion, the optimized hardware design, achieved through meticulous adjustments and optimization strategies applied during the conversion process with HLS4ML, demonstrated superior performance compared to the unoptimized version. By fine-tuning configuration parameters, resolving build issues, and implementing optimization techniques such as pragma unrolling and pipeline utilization, the optimized design significantly improved resource utilization and reduced latency, meeting project requirements effectively. This underscores the effectiveness of leveraging HLS4ML for transforming deep learning models into hardware-friendly implementations suitable for FPGA deployment, ultimately providing a robust solution for brain tumour detection in medical imaging applications.

References

- https://support.xilinx.com/s/question/0D52E00006ihQPSSA2/warning-opmode-input-warning-the-opmode-0110x0x-with-carryinsel-000-to-dsp48e1-instance-is-invalid?language=en_US
- https://fastmachinelearning.org/hls4ml/
- https://github.com/Sarah-Hesham-2022/Doctork-Application
- https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction