EE/CE 6370 ANN Project

Khoa Diep
Dismas Ezechukwu
Matthew Nigh

ANN - Loading and Cleaning the Dataset

- Used Python (Jupyter Notebook) for the designing and training the ANN that predicts handwritten numbers.
- Loaded the data from the MNIST dataset with a Keras function call.
- Scaled each image with pixel values from (0 to 255) to binary (0,1).
  - Allows single bit to represent each pixel with less communication overhead

```
# Scale images to [0, 1]
x_train = np.round(x_train.astype("float32") / 255).astype("int")
x_test = np.round(x_test.astype("float32") / 255).astype("int")
```

- Trimmed each 28x28 image to a 20x20 image which is flattened.
  - Reduces image size by eliminating portion of image which is normally white space.

```
trim = 4
x_train_ = x_train[:, trim:28-trim, trim:28-trim].reshape(x_train.shape[0],-1)
x_test_ = x_test[:, trim:28-trim, trim:28-trim].reshape(x_test.shape[0],-1)
input_shape = x_train_.shape[1:]
```

ANN - Creating the Model

```
model = Sequential()
model.add(keras.Input(shape=input_shape, name="input0"))


model.add(
    QDense(
        10,
        name='fc1',
        kernel_quantizer=quantized_bits(6, 0, alpha=1),
        bias_quantizer=quantized_bits(6, 0, alpha=1),
        kernel_initializer='lecun_uniform',
        kernel_regularizer=l1(0.0001),
    )
)
model.add(QActivation(activation=quantized_relu(6), name='relu1'))
model.add(
    QDense(
        10,
        name='fc2',
        kernel_quantizer=quantized_bits(6, 0, alpha=1),
        bias_quantizer=quantized_bits(6, 0, alpha=1),
        kernel_initializer='lecun_uniform',
        kernel_regularizer=l1(0.0001),
    )
)
model.add(QActivation(activation=quantized_relu(6), name='relu2'))
model.add(
    QDense(
        num_classes,
        name='fc3',
        kernel_quantizer=quantized_bits(6, 0, alpha=1),
        bias_quantizer=quantized_bits(6, 0, alpha=1),
        kernel_initializer='lecun_uniform',
        kernel_regularizer=l1(0.0001),
    )
)
model.add(Activation(activation='softmax', name='softmax'))
model.summary()
```

Summary of the Model

Layer (type)	Output Shape	Param #
fc1 (QDense)	(None, 10)	4010
relu1 (QActivation)	(None, 10)	0
fc2 (QDense)	(None, 10)	110
relu2 (QActivation)	(None, 10)	0
fc3 (QDense)	(None, 10)	110
softmax (Activation)	(None, 10)	0

ANN - Optimal Model and Accuracy

- Created a minimal model with at least 90% training/testing accuracy.
- Training Accuracy: 90.87%

```
Epoch 15/15
422/422 [==============================] - 1s 2ms/step - loss: 0.4336 - accuracy: 0.9087
```

- Testing Accuracy: 90.58%

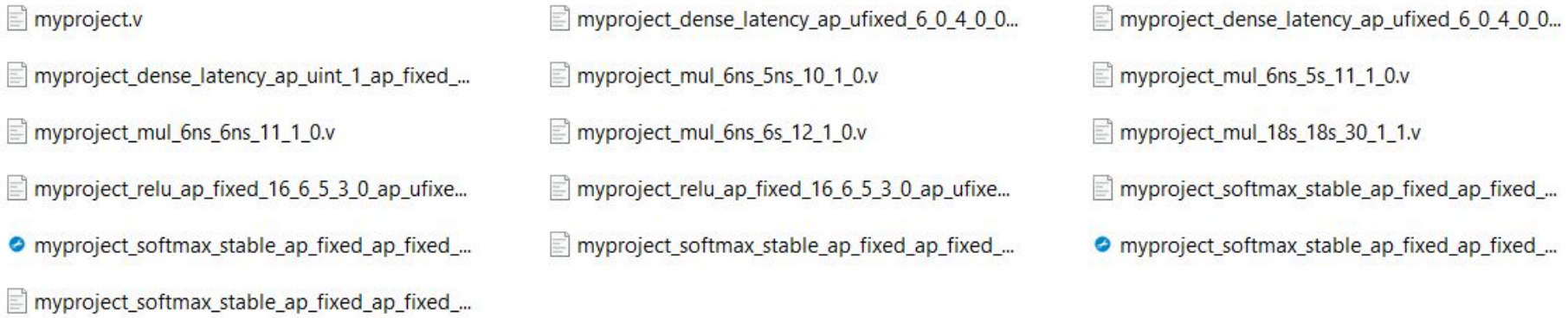
```
Test loss: 0.4460456669330597
Test accuracy: 0.9057999849319458
```

- Total Parameters: 4230

```
Total params: 4230 (16.52 KB)
Trainable params: 4230 (16.52 KB)
Non-trainable params: 0 (0.00 Byte)
```

ANN to RTL - Converting to Verilog Files

- The package, hls4ml, created several Verilog and .dat files which is the translated ANN.
- The top file is called myproject.v.
- Other files translates various aspects (such as the layers) of our model.
- Files generated:



RTL - Simulation

- Created a testbench that tests the output of the RTL ANN to ensure that the prediction of the RTL ANN and Keras ANN matches with each other.
- Testbench Result:

```
Time resolution is 1 ps
Time =                   0, max_i =  x, data_in=00000000000000000000000000000000, nn_done=0
Time =                 110, max_i =  x, data_in=00010000000000000000001111100000, nn_done=0
Time =                 120, max_i =  x, data_in=00100000000011100111100000000001, nn_done=0
Time =                 130, max_i =  x, data_in=00111000000111000000001000011100, nn_done=0
Time =                 140, max_i =  x, data_in=01001100000001000000010001100000, nn_done=0
Time =                 150, max_i =  x, data_in=01010100000001000110000001000000, nn_done=0
Time =                 160, max_i =  x, data_in=01100100001000000100000001000010, nn_done=0
Time =                 170, max_i =  x, data_in=01110000010000001100001000000100, nn_done=0
Time =                 180, max_i =  x, data_in=10000001100000010000000111111000, nn_done=0
Time =                 190, max_i =  x, data_in=10010010000000001110000000100000, nn_done=0
Time =                 200, max_i =  x, data_in=10100000000000000110000000000000, nn_done=0
Time =                 210, max_i =  x, data_in=10110000010000000000000000000100, nn_done=0
Time =                 220, max_i =  x, data_in=11000000000000000000110000000000, nn_done=0
Time =                 230, max_i =  x, data_in=11010000000010000000000000000001, nn_done=0
Time =                 240, max_i =  x, data_in=11101000000000000000000000000000, nn_done=0
Time =                 430, max_i =  6, data_in=11101000000000000000000000000000, nn_done=1
```

- Correct result is shown by max_i when nn_done == 1

FPGA - Resolving DAT files

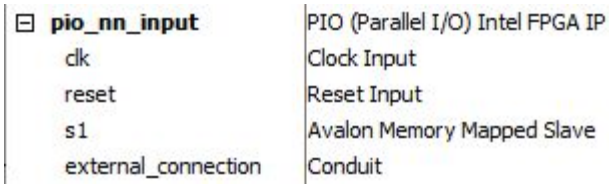
- Created a Python script converting the generated .DAT files (from hls4ml) to .MIF files.
- Utilized the RAM Initializer IP from Quartus to generate memory that contains the weights of the neural network from the .MIF files.
- These steps resolved the fact that the .DAT files are not synthesizable on the FPGA.
- Example from myproject_softmax_stable_ap_fixed_ap_fixed_16_6_5_3_0_ softmax_config10_s_exp_table_ROM_bkb.v:

```
ram1 u2(
    .clock(clk),
    .dataout(dataout),
    .init(1'b1),
    .init_busy(ram_init_busy),
    .ram_address(ram_address),
    .ram_wren(ram_wren)
);

always @(posedge clk) begin
    rom0[ram_address] <= dataout;
    rom1[ram_address] <= dataout;
    rom2[ram_address] <= dataout;
    rom3[ram_address] <= dataout;
    rom4[ram_address] <= dataout;
end
```

FPGA - Driver File

- Modified the ghrd_top.v file from HW9 to handle receiving and sending data from HPS. In Platform Designer, created new Parallel I/Os.
- pio_nn_input represents the a 32 bit inout port which represents both the output of the pixel values of the cleaned image from HPS to the FPGA and the input of the predicted value of the sent image (inout is in the perspective of the HPS).



- The predicted value is calculated by taking the max of the output layer and then sent to the HPS. It is also displayed on the seven segment display.

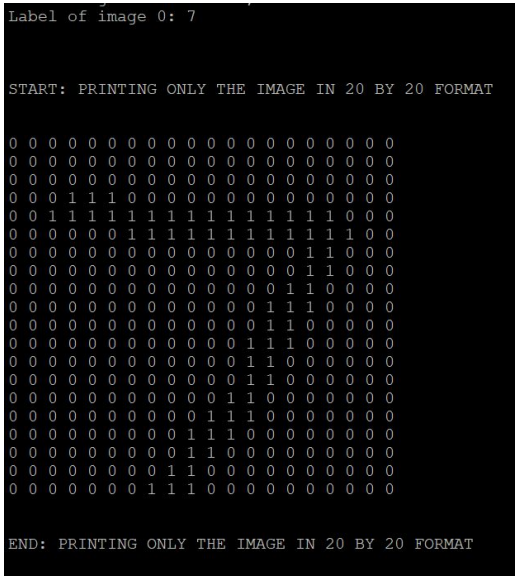
FPGA - Synthesis Results

- Resources used:
  - Summary shown →
- Fmax: 80 MHz → 12.5 ns

Quartus Prime Version	22.1std.2 Build 922 07/20/2023 SC Lite Edition
Revision Name	soc_system
Top-level Entity Name	ghrd_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	7,505 / 32,070 (23 %)
Total registers	5436
Total pins	368 / 457 (81 %)
Total virtual pins	0
Total block memory bits	765,952 / 4,065,280 (19 %)
Total DSP Blocks	36 / 87 (41 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	1 / 4 (25 %)

Embedded C - Data Processing

- Loaded image(s) from the MNIST testing dataset based on the argument(s) given by the user on command line.
  - Argument 1 represents the starting ID of the MNIST dataset
  - Argument 2 represents the ending ID of the MNIST dataset
- Processed each loaded image to match the input of the ANN (20x20 binary pixels flattened).



Embedded C - FPGA Communication

- Batches of 32 bits are needed to send all 400 pixel bits to the FPGA for each image.
- The first 4 bits of the data sent to pio_nn_input represents the offset address.
- The last 28 bits of the data sent to pio_nn_input represents the pixel values of the cleaned image with a total of 15 sent batches.
- Read the predicted data sent from the FPGA and compared it to the actual value of the image.

Conclusions

- YouTube link - (499) CE 6370 ANN Project - YouTube
- For future improvements, we could add more PIOs decrease the time needed to send the values of each image pixel. It would increase the parallelism of the overall project, since the communication between the HPS and the FPGA is the slowest factor of the entire project.
- The package, hls4ml, was a useful tool to transition a neural network from QKeras to C to Verilog, albeit with some frustrations with understanding the documentation and synthesizing the .DAT files.
- Using an FPGA to accelerate the ANN's prediction was faster than Python/Keras prediction, with the tradeoff that it was much more involved and difficult to implement.