WES237C: Project 5 Binary Neural Network

Yi Song yis057@ucsd.edu PID# A53266440

Sebastian Nevarez <u>senevarez@ucsd.edu</u> PID# A59021322

BNN Design

The binary neural network is 3 layers long with an array of 25 unsigned integers (32-bit each) as input (800-bits total). The output is a 10-integer long array with 10 probabilities of the 10 digits (0-9). The highest probability index is the prediction of the input image. To implement each layer of the algorithm, we start by traversing the input array and the weights. A XNOR Popcount function is performed to do a bitwise XNOR on each element in the input array and weights and the number of ones in the result is accumulated. Then we do 2*(accumulated ones -16) - 784 to get a reduced middle layer of integers with size 128. Finally, a sign and quantize function is performed to prepare the array of 128 integers for the next layer. The sign function checks the sign of the integer and assigns +1 to positive numbers and -1 to negative numbers. The quantize function checks the signed numbers and assigns 0s to 1s and 1s to -1s. That changes the image background to all black with white digits written on the background. We repeat the first layer two more times with different array lengths. And finally at layer 3, we take the output before doing the sign and quantize function. That will give us an array of 10 probabilities and the index corresponding to the highest probability is the predicted number. As a result we can see the vitis version is really efficient in terms of hardware, it uses 0 DSP modules which means there are no multiplications inside the code, all bitwise manipulations. Our interval is 6098 cycles making the throughput 16.4kHz.

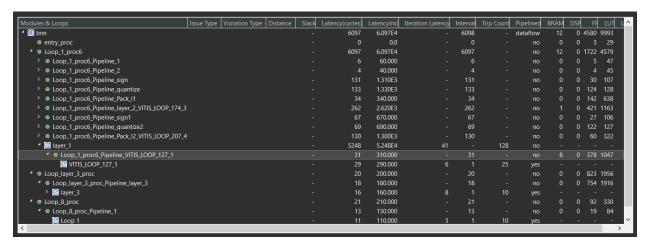


Figure 1: Performance and Resource Utilization

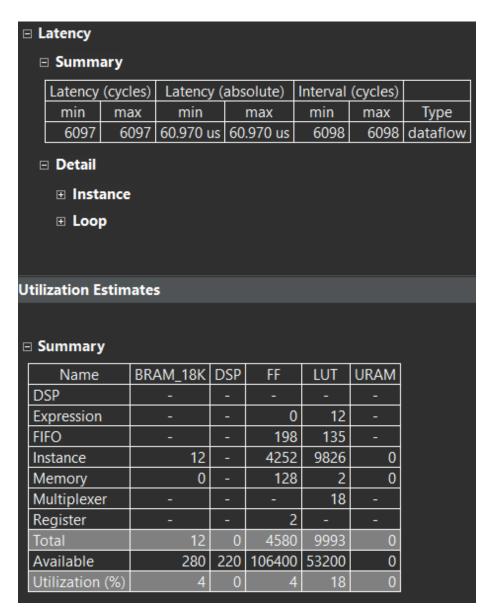


Figure 2: Latency and Resource Utilization Percentages

Python Demo

The main function or our demo starts by defining the run_option which was defined as 1 (XNOR), 2 (MAC), or 3 (HLS). For options 1 and 2, the function run_test is called to run the BNN and the timing is computed using a simple python timer. Option 3 calls the function hiscode to run our implemented BNN and also computes the runtime. See Table 1 below for a comparison of each implementation.

Implementation	Runtime (s)	Accuracy
XNOR	-	89.39
MAC	81.59540247917175	89.39
HLS	-	-

Table 1: Runtime & Accuracy Comparison

The function to run our HLS BNN implementation first loads the bitstream and the provided dataset. From there, our next step is to sanitize our input to match the HLS requirements. To sanitize, we first need to flip all the 0s to 1s and all the non-0s to 0s. The next step is to add 16 0s to the end to reach 800-bits which corresponds to 25 32-bit unsigned integers, the format used in the HLS project and how the weights are trained. Once the input has been sanitized, we are able to run our implementation for BNN. See Table 2 for a snippet of our hlscode function. After running the three layers in the BNN algorithm, we need to output the index with the highest probability as our prediction. A comparison will be made between the output and the label.

```
def hlscode(self):
"""This is a reference implementation for HLS.
Intentionally, left empty so that students implement the HLS ref design.
:return:
# Load the bitstream and dataset
ol=Overlay('./design_1_wrapper.bit')
mnist = np.load("dataset/mnist-original.npy", allow pickle=True)
X = mnist.item().get("data")
y = mnist.item().get("label")
# Sanitize the input
# Flip 0s to 1s and non-0s to 0s
# Add 16 0s to the end to reach 800-bits
# Call BNN
# Output the index with highest probability
# Compare output with the label
print("Done")
```

Table 2: hlscode()