Introduction

This is the fourth and last part of my series about the work I do on this years internship. If you haven't already, you can read the third part here.

Implementing TFLite quantized Network in C

Because I don't know how to read the TFLite format (network structure, variables, etc.) I opted for a small Python program to export everything I need in a C struct.

The only problem: Printing the layers of the TFLite model in Tensorflow is sorted alphabetically by name. So my program is nothing more than a hack, to ensure correct order of the layers.

To make sure that all weights were exported correctly by that program, I decided to write the network inference with floats.

I included the *network.h* file in my C Code and dequantized all weights and biases according to the available quantization parameters.

Calculating the dot-product of the input and the weights for every layer, adding the bias and performing the activation function (ReLu) gives me the correct result.

Going one step further I tried to calculate the inference with the quantized values. There I had to keep the conversions because of the different quantization parameters in mind (scale, zero point).

I got pretty good results (after fixing the saturation range from [-128; 127] to [-127; 127]), but somehow the error is a bit bigger than that of the TFLite implementation. And I have a 32 bit * 16 bit multiplication per neuron, which is also not that great.

Nevertheless I decided to go on with the HLS as I can fix these errors later on (if there is time left).

HLS - High Level Synthesis

HLS in Vivado basically takes a C/C++ program and converts it to a HDL like Verilog or VHDL. But of course there are certain constraints in order to successfully convert a regular C/C++ program.

When compiling a program via gcc in Vivado HLS I got the following error:

> In file included from ../../../network_tb.cpp:1:0: /usr/include/stdio.h:27:36: fatal error: bits/libc-header-start.h: Datei oder Verzeichnis nicht gefunden #include <bits/libc-header-start.h>

Which was fixed by installing:

> sudo apt install gcc-multilib g++-multilib

But after that I was greeted by the next error:

> /path/to/Vivado/2019.2/tps/lnx64/binutils-2.26/bin/ld: cannot find crt1.o: No such file or directory /path/to/Vivado/2019.2/tps/lnx64/binutils-2.26/bin/ld: cannot find crti.o: No such file or directory /path/to/Vivado/2019.2/tps/lnx64/binutils-2.26/bin/ld: cannot find -lpthread /path/to/Vivado/2019.2/tps/lnx64/binutils-2.26/bin/ld: cannot find -lm collect2: error: ld returned 1 exit status make: *** [Makefile.rules:402: csim.exe] Error 1

Apparently this happens because the Linker looks for libraries via LIBRARY_PATH and not LD_LIBRARY_PATH. AR# 69355

To fix this set the environment variable:

> export LIBRARY_PATH=/usr/lib/x86_64-linux-gnu

But for me, in order to get it to work, I had to call the executable over the terminal:

> export LIBRARY_PATH=/usr/lib/x86_64-linux-gnu /path/to/Vivado/2019.2/bin/vivado_hls

---

After adjusting my C program to run with Vivado HLS and convert it to a HDL I got the following report:

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
161	4705	1.610 us	47.050 us	161	4705	none

The max cycles are a bit too high for my taste, let's do something about that.

Optimizations

While designing the Neural Network I had in mind, that the content of every layer executes in parallel. This is not the case, as Vivado HSL executes loops synchronous. To change that we have to tell Vivado which loops to unroll:

> #pragma HLS unroll

And after doing that for every loop we get to:

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
184	184	1.840 us	1.840 us	184	184	none

To decrease the maximum latency even more, we should tell Vivado to pipeline the design:

#pragma HLS pipeline

Now the bigger operations are broken down into smaller ones, which can be processed independent of the complete operation.

This gives us:

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
11	11	0.110 us	0.110 us	1	1	function

Of course by using these optimizations we trade clock cycles for hardware, there is always a downside.

This pragma sets the target cycles for a given function, loop, or region of code. This can either save hardware (if the design is under the minimum cycles) or Vivado will try to meet the maximum constraints.

#pragma HLS latency min=4 max=8

But then it could be, that the target clock constraints cannot be met.

#pragma HLS dataflow

This pragma should also reduce the latency, but it doesn't work with loop unrolling and pipelining. So in this design I can't use it.

#pragma HLS ALLOCATION instances=mul limit=10 operation
#pragma HLS ALLOCATION instances=add limit=10 operation

With these pragmas you can restrict resource usage, for example by only allowing 10 multipliers and adders in your design, and Vivado HLS will automatically modify the HDL to respect this - but of course the overall cycles will increase.

After implementing both model weights into the code to predict both the discharge and charging curve, I came to the following summary:

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
7	7	0.700 us	0.700 us	5	5	function

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	58	0	11903	-
FIFO	-	-	-	-	-
Instance	62	-	2125	10368	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	16323	-
Register	-	-	2178	-	-
Total	62	58	4303	38594	0
Available	2060	2800	607200	303600	0
Utilization (%)	3	2	~0	12	0

The model weights are now basically part of the design, but they should be assigned over parameters.

By chance, while occasionally Googling, I stumbled upon this project which is very similar to what I want to achieve: hls4ml

hsl4ml takes a Neural Network in different formats and converts it into C++ code, readily to be synthesized by Vivado HLS.

Maybe I can find a trick or two to use on my model code.

To be continued...