Building and Investigating an Application Design Space using HLS

ECE 1373: Assignment #1

Due: February 15, 2018

In this assignment, you will learn how to use different features of the Xilinx Vivado High-Level Synthesis (HLS) tool to build hardware engines. You will construct and optimize two key computations in Convolutional Neural Networks (CNNs), namely the convolution and fully-connected layers. During this assignment, you will exercise the use of HLS directives on the hardware engine and analyze the effects of these directives on the structure of the hardware engine. We will also expect you to have done a brief survey of other implementations of neural networks on FPGAs to understand what has worked well in the past. You are welcome to try some of those approaches, but you have to do it through HLS. Lastly we expect you to explore the design space by trying different implementations, justifying the purpose of each implementation, and to compare all your design implementations with each other and against published implementations on FPGAs. The quality of your grade is not only based on the quality of results but on the justification of your chosen architectures.

You will need to submit a report discussing the selection of the algorithm, a justification of the directives used and a description of their effect on the implementation; **read Sec. 4 and Sec. 5** carefully to understand what to submit. Please submit the report as a pdf file and a zip file containing the Vivado HLS project as a private post on Piazza by **February 15, 2017** at 11:59pm. **Late submissions will be penalized by 10% per day.** You may submit the code as a link to a file-sharing service if it is too large to attach on Piazza.

If you have any questions, please start by posting on Piazza. Make the questions public so that everyone can benefit from the responses.

1 Prerequisites

It is required that you have completed the **Vivado HLS tutorials** [1] to use the tools and are able to navigate through the tools with ease. In addition, by having completed the tutorial, you should have a thorough understanding of the various directives and their effects in hardware to guide the design of your hardware engine. It is also recommended that you have completed the tutorials [2] [3] [4] covering the FPGA design flow in the Vivado tool suite. These tutorials will provide an understanding of the method for mapping circuits written in hardware description languages (HDL), such as Verilog and VHDL, to FPGA primitives. While you do not need to know these other tool flows to do this assignment, it will give you some context for where the HLS tools fit. However, Assignment 2 will require you to be able to use the other tools, and you will have only two weeks for that assignment, so it is recommended that you start now. Chapter 1 of the High-Level Synthesis user guide [5] contains useful details for optimizing your design. Please note that for this assignment, you are required to use Vivado version **2017.2**.

2 Background: Neural Networks

A neural network is a type of system that transforms input data to produce outputs. For example, a classical application is handwritten digit recognition where an image consisting of the pixel values of a hand-drawn digit from 0 to 9 is fed into the network and the output identifies which digit was drawn. To perform this task, the input is often fed through successive *layers* of neurons that each transform the input. An example

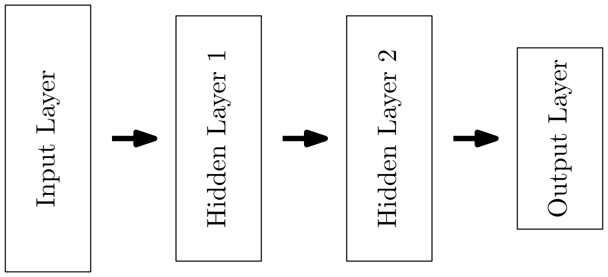

Figure 1: Structure of a Neural Network

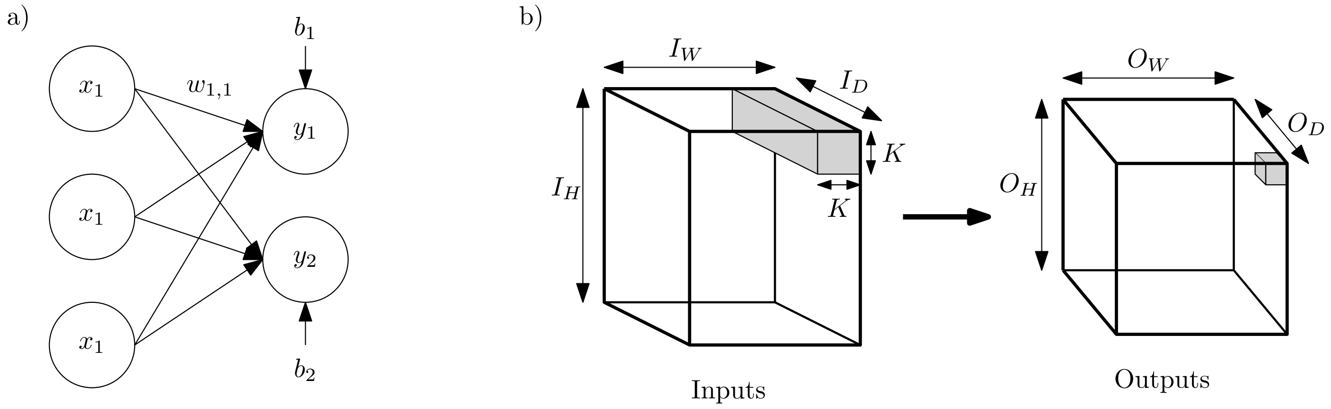

Figure 2: Neural Network Layer Structures. a) Fully-Connected layer connections b) Convolutional Layer structure

of this structure is shown in Fig. 1 where the input layer may be the colour values of an image, hidden layers contain transformed data and the output layer may encode the classification prediction. Parameters, called weights, between each layer are determined during a *training* phase such that the output layer produces the correct result when a new input is introduced. Running an input forward through the network is called *inference*.

In this assignment we will consider inference where the computation can be broken down into computing the transformation performed between each layer. You will create implementations for two kinds of layers: *fully-connected* and *convolutional*. A good reference for neural network concepts are online class notes [6].

2.1 Fully-Connected Layers

A fully connected layer connects N_i input nodes to N_o output nodes where each node contains a real-valued number, an example is shown in Fig. 2 where $N_i = 3$ and $N_o = 2$. The value of an output is determined as a function of the weighted sum of the inputs and a bias. That is,

$$y_j = \sigma\big((\sum_{i}^{N_i} w_{i,j}x_i) + b_j\big). \tag{1}$$

Thus, there are $N_i \times N_o$ weights connecting the inputs and outputs in this layer and N_o bias terms. The non-linear function $\sigma(\cdot)$ is called the activation function and can take different forms. In this assignment, we will consider the Rectified Linear Unit (ReLU) shown in Eqn. 2.

$$\sigma(x) = \max(0, x) \tag{2}$$

2.2 Convolutional Layers

A convolutional neural network layer is similar to the fully-connected case, but differs in how the inputs are mapped to output nodes. Rather than a flat structure, the inputs and outputs are arranged into volumes as shown in Fig. 2b. The input nodes are organized as I_D *feature maps* with width I_W and height I_H. This input is convolved with filters of size $K \times K \times I_D$ to produce output feature maps. During convolution, the filters may also have a stride S. For instance, a stride of 1 would mean that the filter slides across the input one unit at a time whereas a stride of 2 would move the filter two units before producing the next output. The width of an output feature map is $O_W = (I_W - K)/S + 1$ and the height is $O_H = (I_H - K)/S + 1$.

A filter volume is shown as the grey rectangular prism in the input of Fig. 2b. One application of this filter produces a corresponding value in the output volume, shown by the grey cube. The output may have a different number of feature maps (O_D) than the inputs and each output feature map would have a unique convolution filter (and set of weights).

As a filter is being shifted across the input volume, each output is produced as a function of a weighted sum where the weights are the filter values. The output node value at row x, column y and output feature map f_o is computed by:

$$\text{out}(x, y, f_o) = \sigma\Big(\big(\sum_{f_i}^{I_D}\sum_{i}^{K}\sum_{j}^{K} w(i, j, f_i, f_o) * \text{in}(x * S + i, y * S + j, f_i)\big) + b_{f_o}\Big), \tag{3}$$

where 'in' and 'out' index the input and output volumes respectively by width, height and feature map, $w(\cdot)$ indexes the weights, b_{f_o} is the bias and σ is the activation function (Eqn. 2). Note that a single bias is applied for the entire output feature map f_o.

2.3 Batching

The previous descriptions have considered transforming a single set of input nodes to a set of output nodes. However, networks often handle multiple inputs simultaneously to obtain additional parallelism. Thus, the full computation that should be considered for a given layer is transforming B sets of input nodes to B sets of output nodes where B is the batch size.

3 Assumptions & Goal

You are to develop two kinds of High-Level Synthesis accelerators, one for fully-connected layers and one for convolutional layers. You will be provided with parameters and test data for each of the scenarios.

For this assignment, make the following assumptions:

- The maximum batch size B is 10
- All weights, input and output data are real-valued
- The design must fit the targeted FPGA
- The maximum number of nodes (N_o and N_i) for fully-connected layers is 1024
- The convolution window size K is either 1, 3, 5 or 7
- The convolution stride is either 1 or 2
- The maximum dimensions (I_D, I_W, I_H) for convolution inputs is (3, 230, 230)
- The maximum dimensions (O_D, O_W, O_H) for convolution inputs is (64, 112, 112)

Your layers should conform to the following function signatures:

Fully Connected Layer

```
void fc_layer(float weights[MAX_INPUT_SIZE*MAX_OUTPUT_SIZE],
              float biases[MAX_OUTPUT_SIZE],
              float input[MAX_INPUT_SIZE*MAX_BATCH],
              float output[MAX_OUTPUT_SIZE*MAX_BATCH],
              const int batch_size,
              const int num_inputs,
              const int num_outputs);
```

`batch_size` refers to the size of the batch (number of images). `num_inputs` and `num_outputs` define the number of input and output nodes respectively. `weights` is the array that contains the `num_inputs` $\times$ `num_outputs` weights for the layer. The values are organized as `num_outputs` blocks of `num_inputs` weights, such that the first `num_inputs` values corresponding to the weights for the first output node. `input` is the array of input values and `output` is the array of output values. For one of your explorations (Sec. 4 Step 6) you may change the representation of data values from `float`.

Convolution Layer

```
void conv_layer(
float weights[MAX_INPUT_DIMS*MAX_OUTPUT_DIMS*MAX_KERNEL_SIZE*MAX_KERNEL_SIZE],
float biases[MAX_OUTPUT_DIMS],
float input[MAX_CONV_INPUT_SIZE*MAX_BATCH],
float output[MAX_CONV_OUTPUT_SIZE*MAX_BATCH],
const int b,
const int od,
const int ox,
const int oy,
const int id,
const int ix,
const int iy,
const int s,
const int k);
```

`b` refers to the size of the batch (number of images) `od` refers to the number of output feature maps, `ox` refers to the width of the output plane, `oy` refers to the height of the output plane, `id` refers to the number of input feature maps, `ix` refers to the width of the input plane, `iy` refers to the height of the input plane, `s` refers to the size of the stride, `k` refers to the width and height of the convolution window. `weights` is the array that contains the `id` $\times$ `od` $\times$ `k` $\times$ `k` weights for the layer. The values are organized as `od` blocks of `id` $\times$ `k` $\times$ `k` weights, such that the first `k` $\times$ `k` values corresponding to the weights for the first output node. `input` is the array of input values and `output` is the array of output values. `MAX_CONV_INPUTS` refers to the input size of the largest convolution which is 3*230*230 and `MAX_CONV_OUTPUTS` refers to the output size of the largest convolution which is 64 * 112 *112. `|MAX_BATCH|`refers to the maximum size of the batch which is 10. For one of your explorations (Sec. 4 Step 6) you may change the representation of data values from `float`.

4 What to Do

The general steps you will be performing are outlined here. Before starting, be sure to read Section 5.1 to see exactly what you will need to present in your report.

1. Perform a literature review of at least three other works that implement neural networks in hardware.
2. **Without** applying directives to the function interface or changing data representation, implement fully-connected and convolution layers in Vivado HLS targeting the XCVU095-FFVC1517-2-E FPGA.
3. Use C simulation to verify your implementation functions correctly. You will be provided with some test inputs to help you get started and as a basis for measuring performance. These inputs will not cover the full range of input sizes. You may wish to create additional inputs and outputs to further verify your hardware.

4. **Optimization** Refine your implementations using code changes and HLS directives to obtain the best implementations in terms of minimum estimated wall-clock time (cycles $\times$ clock period) to process a batch of data. Ensure that your implementations remain functionally correct through simulation. Obtain final utilization and timing results by running the **evaluate** step when exporting the RTL.

5. Verify your optimized hardware performs correctly and matches your performance expectations using RTL Co-simulation. You may wish to do this at multiple stages in your design process.

6. **Data Representation**: Explore more optimizations by changing the data type of the weights, input and output data. As with your previous designs, verify functional correctness with co-simulation and estimate how the changes affect the overall computational accuracy of your outputs. **If the data representation affects the accuracy, ensure that the mean squared error of the outputs is less than** 0.001 for the test data provided. Obtain final utilization and timing results by running the **evaluate** step when exporting the RTL.

Note: There will be prizes for the highest performing designs using the floating point representation (Step 4) and modified data representation (Step 6). The designs must still meet the target area and mean squared error constraints defined above.

5 Deliverable and Report Breakdown

5.1 Report Outline

Prepare a report, no more than **6** pages of text (figures do not count towards this limit). The report should follow an academic paper format and style. You can use a predefined conference template. We recommend the IEEE format template [7]. A part of the grade will be based on the quality of the presentation of your work in a style appropriate for an academic paper.

The report should include, but is not limited to:

1. Abstract
   - A brief summary of the key points of the report.
2. Introduction
   - Very briefly describe neural networks and how they may be mapped to FPGAs.
3. Related Work
   - A literature review of other implementations of neural networks on FPGAs. Please review at least **three** other works. Briefly comment on their hardware architecture, and justifications of their architectural decisions.
4. Case Studies
   - The focus of this section is for you to do a thorough design space exploration of your hardware cores and show that you understand how you use HLS to create different types of architectures.
   - As you work towards your best designs in Step 4 of Section 4, you will be trying different optimizations that will result in many different implementations. For both fully-connected and convolutional accelerators, pick two intermediate implementations plus your best implementation and compare them in tables showing area and performance metrics. Report performance using the provided test cases. Comment on the results.
   - For each of the six chosen implementations, use a figure to show the resulting hardware architecture and describe your reasoning for your choice of modifications. Simply saying **"Pragma X has given me the best results"** will **not** suffice. A better justification would include a statement like, "Pragma Y creates a 4-stage pipeline increasing the throughput by a factor of four, as shown in Figure 2 and Table 3."
   - In Step 6 of Section 4 you modify the data representation for your designs. Compare your best design using a modified data representation with your best floating-point implementation. Discuss any further modifications that were performed in addition to changing the data representation. Finally, discuss the tradeoffs between these two designs.
5. Conclusion
   - For your best designs, compare and contrast with related work. How do your architectures differ? How do the results differ (you may not be able to directly compare performance)?
   - Lastly given related work, and what you implemented, what limitations did you face? Did you notice an architectural design that you could not implement in HLS? What approaches do you think you can use to circumvent these limitations (e.g break the circuit into multiple circuits).
6. Appendix
   - Figures, tables, charts, code and other additional information.

Note: The Appendix does not count in the **6** page limit.

5.2 File Submission Format

For each HLS Project that you have (you may have multiple as modifications to the source code requires a separate project), archive the project by clicking **File** and then **Archive Project**. This will open a new dialog box. Ensure that you have **unchecked** the option *Active Solution Only* and **checked** the option *Include Run Results*. Once you have archived all your HLS projects, zip all of these projects into one zip directory.

References

[1] *High-Level Synthesis Flow on Zynq using Vivado HLS*, `http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-high-level-synthesis-flow-zynq.html`.

[2] *Embedded System Design Flow on Zynq using Vivado*, `http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-embedded-design-flow-zynq.html`.

[3] *FPGA Design Flow using Vivado*, `http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-embedded-design-flow-zynq.html`.

[4] *HDL Design using Vivado*, `http://www.xilinx.com/support/university/vivado/vivado-teaching-material/hdl-design.html`.

[5] *Vivado Design Suite User Guide: High-Level Synthesis*, `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug902-vivado-high-level-synthesis.pdf`.

[6] *CS231n Convolutional Neural Networks for Visual Recognition*, `https://cs231n.github.io/`.

[7] *Manuscript Templates for Conference Proceedings*, `http://www.ieee.org/conferences_events/conferences/publishing/templates.html`.