Integrating a Neural Network on an FPGA

ECE 1373: Assignment #2

Due: March 12, 2019

In this assignment, you are required to create a neural network using layers you implemented in HLS (engines from Assignment 1) into an environment with a PCIe connected FPGA. We will provide you an abstraction to communicate via PCIe to your FPGA application along with an interface to off chip memory. You are to implement the VGG-16 [1] Neural Network on the FPGA until the final fully connected layer. You will be responsible for creating an FPGA application that conforms to an AXI slave interface to receive input from PCIe and the AXI master interface to read data and write data from off chip memory. Furthermore you will also be expected to investigate different ways to partition your working memory space into sizable sections that will fit on-chip (or suffer off-chip penalties). During this assignment you will discover how to communicate via the AXI protocol to off-chip resources, interface with a processor and move and partition memory efficiently. You are expected to provide one design that physically implements one convolution layer, one max pool layer and one fully-connected layer and time multiplexes these layers for a simple design. Furthermore for a small percentage of your grade you can improve this design by implementing more specialized layers.

You will need to submit a report discussing your optimizations and justifications of the different system-architectures used; **read Sec. 3 and Sec. 5** carefully to understand what to submit. Please submit the report as a pdf file as a private post on Piazza by **March 12, 2019** at 11:59pm. **Late submissions will be penalized by 10% per day.**

If you have any questions, please start by posting on Piazza. Make the questions public so that everyone can benefit from the responses. Please refer to Section 5.2 for details on how to submit your project. For this assignment you will be submitting your code through your container that you will be given access to. Please provide tcl scripts so that we can regenerate your designs. Please verify that this tcl script works with the creation of the partial reconfiguration region. Refer to the README in the project repository for more information.

1 Prerequisites

It is required that you have completed the **Vivado HLS tutorials** [2] (from Assignment 1) and **Vivado Tutorial** for a design flow that covers the IP Integrator tool [3]. For additional context on hardware review refer to **HDL Tutorial** [4]. You should also be familiar with the Alphadata 8v3 FPGA Board [5]. You are expected to make a kernel that communicates to PCIe and off-chip memory via AXI. We have provided the abstractions but it is important to understand the AXI protocol. This information can be found in the AXI user guide [6]. In addition, Chapter 1 of the High-Level Synthesis user guide [7] contains useful details for optimizing your design. Please note that for this assignment, you are required to use Vivado version **2017.2**. We also provided a debug ILA module to debug your design in real time. For more information on how to use the ILA please refer to user guide [8]. Note you will not be making more debug ports, but you will use the ones we provided.

2 Assignment Overview

You are to implement the VGG-16 [1] Neural Network on the FPGA. In Assignment 1 you have created a fully connected and a convolution layer in HLS. We will use Caffe, an open source Machine Learning Framework to gather the input and verify the output [9].

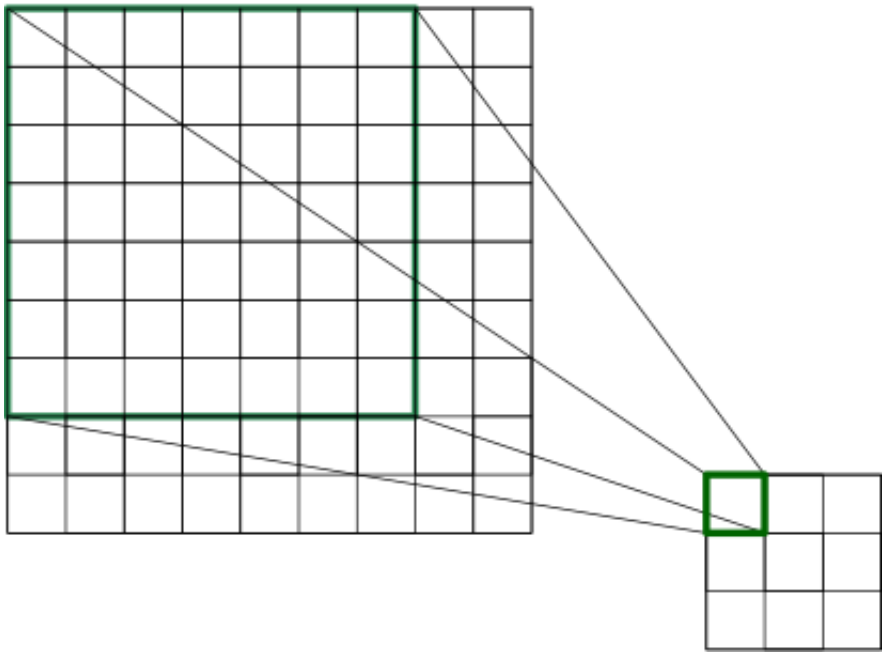

Figure 1: The green filter represents the window where we will output the maximum value in the input covered by the window. In this example the filter is moving at a stride of 1

To implement the full network, you will need to implement the Max Pooling layer. In this layer we slide a window across an input feature map and choose the maximum value in the window. The window slides according to the stride size. This is similar to the convolution layer, but instead of performing a matrix multiply on the weights associated with the filter, we perform the max operation on the input covered by the filter. An example of the pooling window is shown in Figure 1.

2.1 Provided Infrastructure

You will be provided access to a container with a connected Alpha Data 8V3 card. Instructions for accessing the container will be provided on a Piazza post. Inside the container you will find:

- Source code to compile bitstreams
- Example project that includes convolution layer running in hardware
- Python scripts to measure accuracy of result
- Python scripts to generate input for your driver

2.2 Provided FPGA Abstraction

We provide an FPGA Hypervisor abstraction for the FPGA that will be pre-programmed on the FPGA. This encapsulates a PCIe module, debug core and an off-chip memory controller. The Hypervisor and the interfaces you will have to communicate with the abstraction are shown in Figure 2. The PCIe module contains a Direct Memory Access (DMA) engine as well as a memory-mapped AXI Lite interface. Using the DMA engine, the PCIe IP can address the first 2GB of the DDR memory and example code is provided demonstrating how to write values into memory. The weights and input data will be provided in software and it will be your job to transfer the data onto the FPGA and efficiently access the data.

Your design will be implemented in the Application Region. This region has an AXI Lite slave port which can be addressed by the PCIe core. An address space of 512KB is available in which you can map your

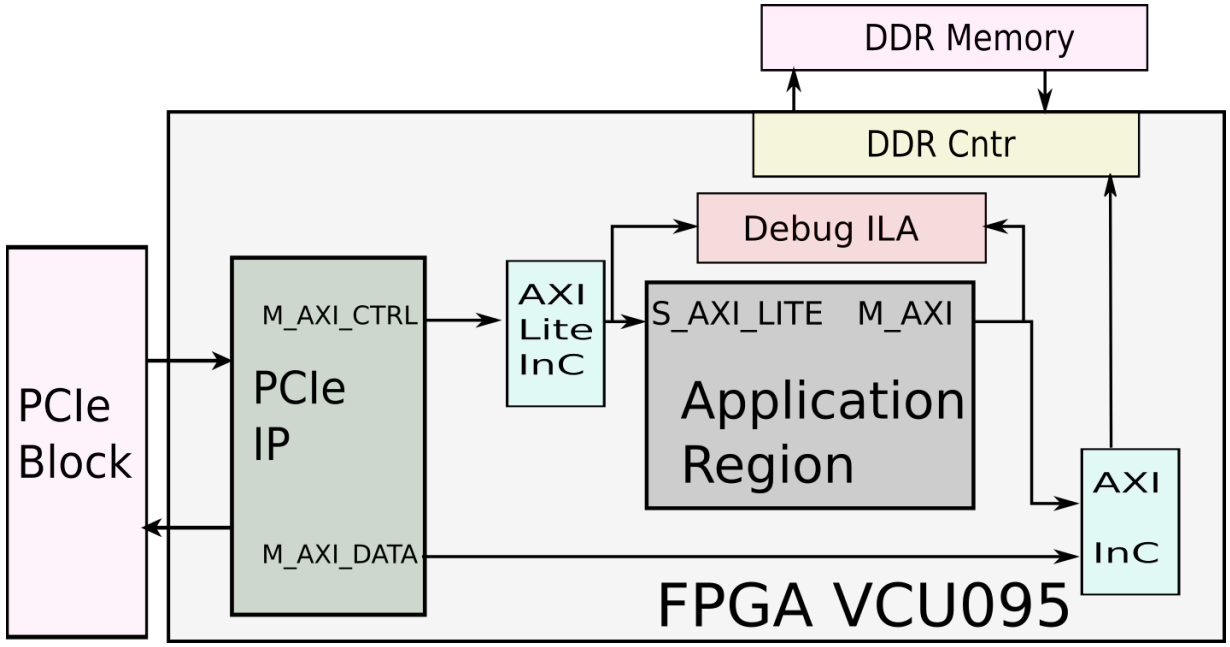

Figure 2: The FPGA Hypervisor abstraction provided. Main cores provided are the PCIe IP, DDR controller IP and a Debug ILA.

hardware control registers. A provided example demonstrates how to address into cores in the Application Region. The Application Region also exposes a master AXI port from which IPs can access the full 8GB of available external DDR memory. Note that to connect multiple IPs to the ports of the Application Region, you will need to create interconnects within the region and specify correct offsets and address spaces.

You will use partial reconfiguration [10] to program the Application Region into the FPGA (leaving the Hypervisor programmed on the FPGA). The Hypervisor also contains a debug ILA attached to the AXI control and data interfaces. This will help you debug signals in real time.

2.3 Function Interface and Data Formats

We are more flexible with the function interface in this assignment as you will handle the transfer of data between the CPU and the FPGA. To measure the accuracy of your implementation, you must use the provided floating point values as inputs to your design and will need to produce floating point outputs from your last layer which will be used to compute the predicted labels in Caffe. However, you may perform type conversion of these values in software and the time to do this conversion will not count as part of your accelerator runtime.

2.4 Provided Software Infrastructure

We provide a Python framework to extract layer information and input from Caffe. This input is gathered in batches, where the number of batches is programmable within the script. Refer to Section 2.5 for the number of images to be run. This data is then taken by your test application that will call a driver. The test application will read the binary of the batched data, DMA the data into the FPGA and start the application. We have provided an example test application that does this. Feel free to modify the test application to interface with your first layer and read back the results from the last layer. Lastly we also provide a Python framework that reads out the binaries output from your FPGA and gathers the accuracy of the prediction.

2.5 System Design

We provide an example system with a single convolution layer connected to PCIe and memory. Your tasks are as follows:

1. For the majority of your grades (80%) you will create a simple overlay by designing a system with one convolution layer, one fully connected layer and one max pool. This overlay can be designed to time-multiplex the different layers needed to run VGG-16. To do this you can modify our example to add the fully connected and max pool layers.
2. For the balance of your grades, you will investigate optimizing your first design by potentially adding more layers in hardware, investigating data formats, or using any other architectural enhancements.

You will be optimizing the design for overall run-time of your application on a batch of images provided. We will expect you to gather the run-time to process 100 images. The number of batches are up to you (and in-turn batch size). We will measure the time from when you start the application (apply the start signal to your first kernel) till you have read back the last bit of the output (including the time to DMA from the FPGA into the CPU). You will be implementing this on the provided boards within the cluster. You can modify the data representation of the VGG-16 Network as long as the Top-5 accuracy is above or equal to 95 %. This number may seem high but the data we have available is a subset of the training set so we expect higher than normal accuracy numbers due to overfitting (not ideal). If you notice variation in your run-time, please average the results over several runs.

3 What to Do

1. Refer to the VGG-16 paper [1] for the description of the layers in the neural network. Do not implement the Soft-max layer. Read the data from the final fully connected layer into software and use our driver to send this data into Caffe's implementation of Soft-max. **Also note that the final fully connected layer does not have an activation ReLU**.
2. Implement the Max Pooling layer in HLS and modify your existing Convolution and Fully Connected cores to integrate with the provided Hypervisor system.
3. Integrate your compute cores into the provided application region using Vivado and verify that they work correctly.
4. Explore different system and IP-level optimizations including modifications to your kernels, interconnect and additional IPs to improve the performance of your system. Ensure that your designs continue to meet the accuracy requirements described in Section. 2.5. Report performance on both unoptimized overlay and any optimizations you have made. If you are unable to implement any optimizations please comment on some optimizations you would attempt. In addition, comment on the area usage and performance/area trade-off you observed.

4 Hints

Here are some things you might want to consider:

1. Consider modifying your convolution kernel to create the max pool.
2. Consider how often you are accessing off-chip memory, maybe we can move things to on-chip memory?
3. If you add on-chip memory consider modifying your interconnects to support that.
4. What data representations would you like to consider?
5. How general does your particular layer implementation need to be?
6. How are multiple layers communicating with one another, and how do you signal the beginning of the next layer?

5 Deliverable and Report Breakdown

This section highlights the report breakdown and project submission instructions.

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 [11].

The report should include, but is not limited to:

1. Abstract
    - A brief summary of the key points of the report.
2. Introduction
    - Provide a brief high-level description of the different types of optimizations you attempted in your Case Studies.
3. Case Studies
    - The focus of this section is for you to build a large system, first a simple overlay and then to investigate optimizations.
    - Describe your basic overlay with utilization and performance results. Next describe optimizations (with figures). If you cannot implement further optimizations comment on optimizations you would like to try and comment on projected performance benefits.
4. Conclusion
    - Which type of optimizations gave you the best performance results?
    - What optimizations would you like to have tried that you have not had a chance to implement?
    - What challenges will you face if you were to scale your design to a much larger input data set?
    - How would you implement this system differently if you were to optimize for latency or throughput?
    - Other concluding remarks.
5. 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

To submit you will leave your code on the container you are given access to. Please archive your code in the home directory of your container and include a README instructions for us on how to run your projects including locations of all bitstreams. Note that one of us will be collecting your archived project and sending this to the other TA to grade, if the instructions are not clear for that TA to grade then this will reflect poorly on your assignment.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.

[2] *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.

[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] *ADM-PCIE-8V3 Product Page*, https://www.alpha-data.com/dcp/products.php?product=adm-pcie-8v3.

[6] *AXI Reference Guide*, https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.

[7] *Vivado Design Suite User Guide: High-Level Synthesis*, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_3/ug902-vivado-high-level-synthesis.pdf.

[8] *Vivado Design Suite User Guide: Integrated Logic Analyzer*, https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_2/pg172-ila.pdf.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, ACM, 2014, pp. 675–678.

[10] *Vivado Design Suite Tutorial: Partial Reconfiguration*, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_1/ug947-vivado-partial-reconfiguration-tutorial.pdf.

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