FPGA Implementation Of A Neural Network For Image Recognition

Stanley Chen
stanley.chen0@gmail.com

Yiu Yung Wong
ymwong@ucdavis.edu

Kevin Wang
kzwang@ucdavis.edu

ABSTRACT

The aim of this project is to develop a robust neural network on a FPGA/HPS platform that is able to recognize and process image files. The overall goal of this paper is to discuss the advantages of using an FPGA implementation in complement with a Hard Processing System to implement the neural network instead of simply using single threaded cores to process the entire system. The neural network has a lot of repetitive and often similar calculations, which can be exploited through parallelism on the FPGA platform. The original goal was to increase the processing of every image from 0.2 seconds to 0.1 seconds and maintain an 95% accuracy rate. However, our group's implementation was able to push it much further past the original goal. Our final speed per image came 0.004 seconds per image with a final accuracy of 92-93%. Throughout this paper, we will evaluating the algorithm, as well as the development and implementation of our design.

1. INTRODUCTION

Neural network systems currently is one of the best solution to many large computing problems for image recognition. A neural network is a data processing model in the field of machine learning that consists of a large network of interconnected elements called neurons that act upon observational data. They are inspired by the human brain; they use conventional processing to mimic the neural network and create a system that can learn through observational data. The neural networks themselves can replicate one method of how humans learn, which is through trial and error.

Today, Field Programmable Gate Arrays, or FPGAs, have become very popular in certain areas of computers over core processors. They are reprogrammable chips that can implement a variety of functionality based on its HDL design. They typically run at lower frequencies compared to multi-core processors. Despite that, one of the prime reasons it is favored over CPU cores is that they can be configured to be parallel processing devices versus the sequential behavior of the traditional CPU, which allows them to possibly outperform their multi-core brethren depending on the task at hand. With enough parallelized tasks, a project may cost less in terms of power and still outperform a multi-core, high frequency processor.

In this project, FPGA is an alternative platform in order to implement the neural network because it has many repetitive calculations that can be done in parallel.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. ALGORITHM

The neural network we were given to implement consisted of 4 layers: an input layer consisting of 784 nodes corresponding to the 784 pixels of the image file, two hidden layers that consist of 200 nodes respectively that performs the calculations, and an output layer with 10 nodes corresponding to the 10 possible predicted values ranging from 1 through 10. A sample of our neural network design is shown in Figure 1.

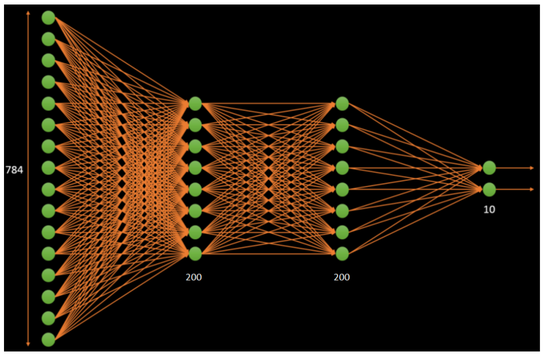


Figure 1: Neural Network Design

The neural network calculations are performed from layer to another layer until the final layer is reached. The input to each node in the next layer is calculated by multiplying the input value with the weights. These are summed up together and evaluated through an activation function, typically in the form of a sigmoid function. These values are used to calculate the following layer until we reach the output layer. The figure below shows one example of how to calculate a node. The predicated value is the maximum value out of all the inputs to the output layer. An example of how to perform a single calculation for a node in each next layer is provided in Figure 2.

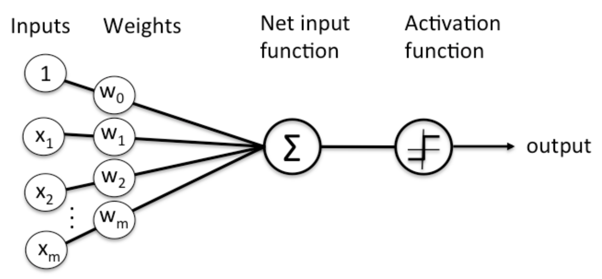


Figure 2: Example of Calculating a Single Node

In our neural network design, we made some slight modifications in the architecture in order to make some of the calculations easier. All data values (image data, weights, and biases) are all given in floating point numbers, which makes the value representation in bits much more difficult. Thus, our group made a team decision to round the values in order to keep them in 16-bit integer format, which increased the ease of development. Also, another customization we implemented was to use a modified version of the sigmoid function. Again, our plan was to move from floating numbers, which consume more processing power, to integers. Hence, we chose a step function instead, which is a steeper and binary version of the sigmoid function.

3. IMPLEMENTATION

For our design, our group transferred the entire neural network classification to be processed on the FPGA fabric instead of running any of it on the HPS side. The only thing that the HPS did was to validate the predicted values from the FPGA against the expected values. In this section, we will be detailing each of our main acceleration methods.

3.1 Data Cleaning

Earlier, it was discussed that we performed some customization on the provided data and as well as the algorithm within the neural network itself. The reasoning was simply to increase the simplicity of development. One of the main reasons is that the Altera DE1-SoC board is limited by 87 DSP to perform multiplication. Since every input value to each layer ranged from anywhere from 0 to 1, we simply rounded those values and used a multiplexer to perform the multiplication instead of having the DSP run an integer-based multiplication.

All data cleaning methods we chose to implement was tested on Matlab first to ensure that our design met the minimum requirements for accuracy. Since we were tinkering with the numbers, our group figured that we would lose some accuracy, but we were fine as long as it was not too debilitating. In the end, after all our data cleaning, we simply lost 1% of our accuracy, going from 98% to 97%.

3.2 Memory Management

The Altera DE1-SoC board provides two different types of memory: On-Chip Memory and SDRAM Memory. Each of these memory types have a trade-off between space and speed. The On-Chip Memory is extremely fast, but limited in space. The vice-versa is true for the SDRAM Memory. There is also a difference between the communication protocol on the On-Chip and SDRAM. In Figure 3 and Figure 4, it shows the difference between the two memory types. The On-Chip ensures that data is received on the next clock cycle after the request is sent out by the master module. The SDRAM does not have the same guarantee. Using the On-Chip, it will ensure that we have an *O(n)*. If we had used SDRAM to store these weights, we would be Ω*(n)* By storing certain data in different memory locations, we can make retrievals faster.

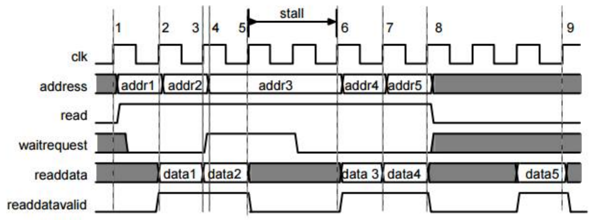


Figure 3: SDRAM Communication Protocol

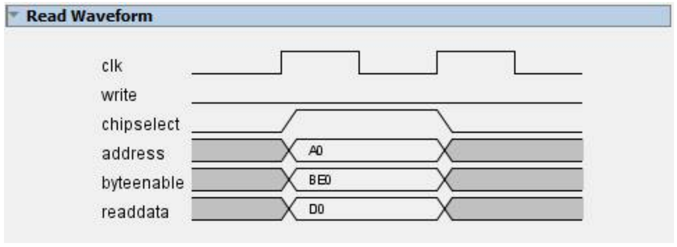
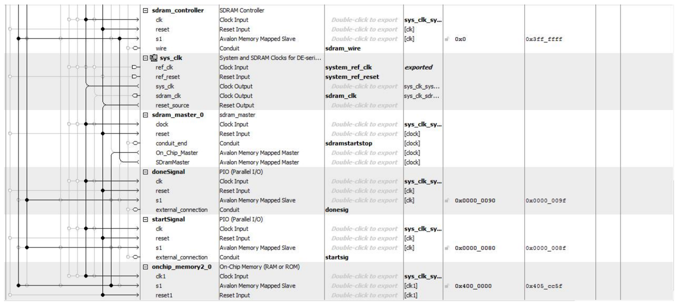


Figure 4: OnChip Communication Protocol

For our implementation, our designed the On-Chip Memory to be a completely read-only design and was used to primarily read in all weights to transition to the first hidden layer from the input layer. This transition held the about 79% of the entire architecture's weights. The idea was that we wanted to make the common case of accessing these weights faster. All other data was that was needed for calculations was stored on SDRAM.

3.3 Development of Custom IP

Our group developed a custom IP that managed the entire process. It has a Start PIO that is connected between FPGA and HPS, so that the HPS can communicate with the custom IP to let it know when to start. The Done PIO is monitored by HPS to determine when the entire calculation is done. Our custom IP design on Qsys is given below in Figure 5.



sdram_controller	SDRAM Controller				
clk	Clock Input	Double-click to export	**sys_clk_sy...**		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0	0x3ff_ffff
wire	Conduit	**sdram_wire**			
sys_clk	System and SDRAM Clocks for DE-seri...				
ref_clk	Clock Input	**system_ref_clk**	***exported***		
ref_reset	Reset Input	**system_ref_reset**			
sys_clk	Clock Output	Double-click to export	sys_clk_sys...		
sdram_clk	Clock Output	**sdram_clk**	sys_clk_sdr...		
reset_source	Reset Output	Double-click to export			
sdram_master_0	sdram_master				
clock	Clock Input	Double-click to export	**sys_clk_sy...**		
reset	Reset Input	Double-click to export	[clock]		
conduit_end	Conduit	**sdramstartstop**	[clock]		
On_Chip_Master	Avalon Memory Mapped Master	Double-click to export	[clock]		
SDramMaster	Avalon Memory Mapped Master	Double-click to export	[clock]		
doneSignal	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	**sys_clk_sy...**		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0090	0x0000_009f
external_connection	Conduit	**donesig**			
startSignal	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	**sys_clk_sy...**		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0080	0x0000_008f
external_connection	Conduit	**startsig**			
onchip_memory2_0	On-Chip Memory (RAM or ROM)				
clk1	Clock Input	Double-click to export	**sys_clk_sy...**		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x400_0000	0x405_cc5f
reset1	Reset Input	Double-click to export	[clk1]		

Figure 5: Custom Master Module

Most of the operations done was predominantly reading in data values and calculating for a node. One method of improving speed of calculations was by pipelining the reading of data values with the calculations. Every clock cycle we would send out an address corresponding the where the data is and then as data is retrieved, we would put it through calculations. When enough data values are received and calculated, the master module will store it away and start calculating for another node. This is repeated until it reaches the output layer. After all predictions have been made, the master module writes these predictions to an area in SDRAM, where the HPS will compare the predictions against the expected values.

4. RESULTS

In our final design, it processed images every 0.004 seconds and the total accuracy ranged between 92% to 93%.

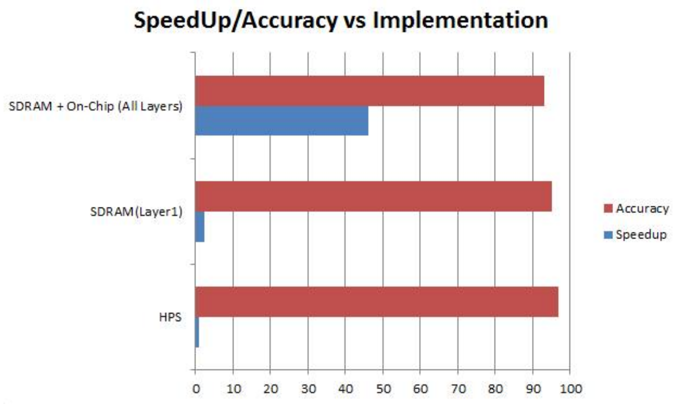
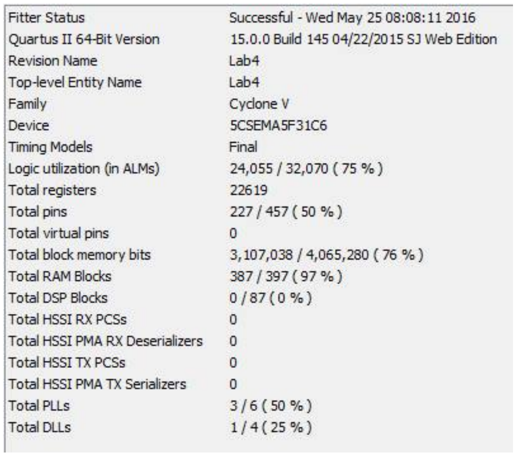


Figure 6: Speed-Up and Accuracy Benchmark For Each Design

In the table above, we compared our design to two other designs that we had originally. One of design was our milestone 1, which only accelerated the first hidden layer through use of the SDRAM. The design only sped up by 2.4. However, by using as much of the hardware allotted to us, we were able to get a much higher speed up.



Fitter Status	Successful - Wed May 25 08:08:11 2016
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Web Edition
Revision Name	Lab4
Top-level Entity Name	Lab4
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	24,055 / 32,070 (75 %)
Total registers	22619
Total pins	227 / 457 (50 %)
Total virtual pins	0
Total block memory bits	3,107,038 / 4,065,280 (76 %)
Total RAM Blocks	387 / 397 (97 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	3 / 6 (50 %)
Total DLLs	1 / 4 (25 %)

Figure 7: Fitter Resource Report

5. LESSONS LEARNED

A large portion of our time was consumed by debugging timing issues that our group did not think about or understand fully. This came in the form of poor coding practices in Verilog. We wasted nearly a week debugging the code and its integration into hardware. A large part of this could have been remedied by going to office hours more often or simply seeking the Mohammad's help throughout the quarter. Also, a good week and a half was wasted purely on trying to implement multiple modules together, when in fact we should have implemented and tested module by module. There was a lot of wasted time due to poor planning. Our group had a good ideas on how to perform parallel tasks in this project, but the poor planning and unnecessary debugging led to a lot of wasted time. We had hoped to implement more methods of improving speeds, but ran out of time.

Overall, the class was pretty interesting in terms of relativity to modern day technology. However, there are a lot of ways the class could be improved. Our group dealt with debugging by monitoring memory and outputting certain valuable data to the FPGA peripherals. This still did not remedy the fact that compile times are 20-30 minutes long (especially if we mess up one line of code). Going over how to use ModelSim or SignalTap to really understand what's wrong with our designs would have been extremely helpful and would remove our need to recompile so often.

6. CONTRIBUTION BREAKDOWN

Yiu Yung Wong - *Algorithm Design, Data Cleaning, Hardware Architecture Development*

Stanley Chen - *Hardware Architecture Design and Development, Testing*

Kevin Wang - *Hardware Architecture Development, Testing*

7. BROADER ASPECTS OF ENGINEERING DESIGN IN A GLOBAL, ECONOMIC, AND SOCIETAL SETTINGS

This project is very relevant in today's technological world. Altera, one of the leading FPGA companies, recently got acquired by Intel. One of the predicted reasons why is that Intel wishes to implement these chips alongside their servers, which would reduce the power consumption of complex algorithms through the FPGA's ability to implement parallelism. Every clock cycle that occurs means that process is able to do more work, thus requiring less total energy for the entire process.

Another reason is that computer architecture design is trying to move to exploiting more parallelism. Moore's law is reaching its end, hence we must find new ways to improve performance.

Overall, this project shows how a lower frequency unit can still overcome the typical high frequency cores.

8. CONCLUSION

Overall, our group felt we learned a lot throughout these last two quarters about digital design. We express our thanks to Professor Soheil Ghiasi and Mohammad Motamedi for providing this Senior Design course and helping us throughout the entire two quarters.