This phase of the project consisted of developing the synthesis model in Matlab, converting it to synthesizable VHDL code, then synthesizing the actual hardware to be instantiated on the FPGA. In order to develop the Matlab synthesis model, two additional Matlab files had to be written, a synthesis function and a synthesis function testbench. The synthesis function contains the forward inference code for the LeNet5 neural network. This code is an iterative model that computes the forward inference of the neural network using a set of for loops. The synthesis function testbench consists of the behavioral model of the neural network that is used to source the network coefficients, along with the test data, which then repeatedly calls the synthesis function and records whether the prediction was right or wrong.

The next step was to run this function through the HDL Workflow Advisor. The input types for the function were successfully defined and all weights, intermediate values, and input data was successfully quantized to an 8 bit word size. This quantization was arrived at via multiple simulations run using Matlab's Fixed-Point Conversion tool. Quantizing to 8 bits was sufficient to preserve the accuracy of the network, while reducing the data size from a 32 bit single precision data type.

Next, the code generation target was configured. This is where the first issue was encountered. The academic license that is available at RIT does not contain the board support package for the Nexys4 DDR. As a result, the required VHDL code was generated as a generic FPGA project, with the chip manually specified.

The final step before VHDL generation was to configure the optimizations used to create the VHDL model. The clocks were configured with an asynchronous, active high reset, with clock enables. In order to simplify the entity interface, the system was configured to map the constant model parameters into block RAM. The model was also pipelined, with 10 stages, to account for the 10 stages of the LeNet5 model. The VHDL code was then generated.

Once generated, the model was then synthesized. This is where the most significant issue was encountered. After waiting three hours for synthesis to complete, the total resource utilization was extremely high. The summary resource utilization is shown as Table 1.

Multipliers	2,597,058
Adders/Subtractors	5,357,644
Registers	62,834
RAMs	0
Multiplexers	11,152
I/O Bits	501,924

Table 1: Summary of synthesized resource utilization.

This level of resource utilization is physically impossible to implement on the provided FPGA hardware. Furthermore, it appears that even though Matlab is configured to place constant values in block RAM, this does not appear that this is happening. Based on an examination of the code, it appears that Matlab is attempting to synthesize the entire model as combinational logic, which leads to extremely high count for multipliers and adders. The

region that is responsible for this are the fully connected layers. Matlab appears to be using generate statements to build crude vector multipliers by arraying hundreds of thousands of multipliers and adders, leading to the massive resource consumption. Lastly, deeper examination of the HDL Resource Utilization Report shows that even though the quantization is configured to be 8 bits for all values, the number of 8 bit multipliers and 8 bit adders is only 840 and 0, respectively. However, the vast majority of the multipliers and adders created are 32 bit devices, indicating that Matlab may be ignoring many of its synthesis settings.

NOTE: For further details, the following files may be of interest in the submitted project archive.

- HDL Resource Utilization Report:
  nnproject/synthBuild/lenetSynthMatlab/hdlsrc/resource_report.html
- Matlab-Generated VHDL Model:
  nnproject/synthBuild/lenetSynthMatlab/hdlsrc/lenetSynthMatlab_fixpt.vhd