Project: Designing GNN Accelerator (SP23)

Milestone 1: Due 3/2/22 (Thursday) 11:59pm
Milestone 2: Due 3/9/22 (Thursday) 11:59pm
Milestone 3: Due 3/30/22 (Thursday) 11:59pm
Milestone 4: Due 4/13/22 (Thursday) 11:59pm
Milestone 5: Due 4/27/22 (Thursday) 11:59pm
Milestone 6: Due 5/4/22 (Thursday) 11:59pm

Read the entire document carefully before starting the lab!
Do the lab individually. Your designs will be checked for plagiarism.

There are 6 stages for this project.

Milestone 1: Write the Verilog code for a deep neural network (DNN) that will be embedded into a graph.

Milestone 2: Write the Verilog code for a GNN that embeds the DNN from MS-1.

Milestone 3: Synthesize your design using Design Compiler and verify the synthesized netlist.

Milestone 4: Perform automatic place-and-route (APR).

Milestone 5: Post-APR-export GDS, import the GDS into Virtuoso layout, and perform

DRC/LVS on the final layout of the design.

Milestone 6: Make power/performance estimations.

NOTE: At every stage, make different folders such as RTL/, Synthesis/, APR/, Performance/.

Introduction

Graph Neural Network (GNN) is widely used for classification problems. In simple words, GNN consists of nodes. Each node implements a neural network. The nodes are connected with directed edges. The graph represents a wireless application comprised of 4 tasks in this project. Figure 1 shows the diagram.

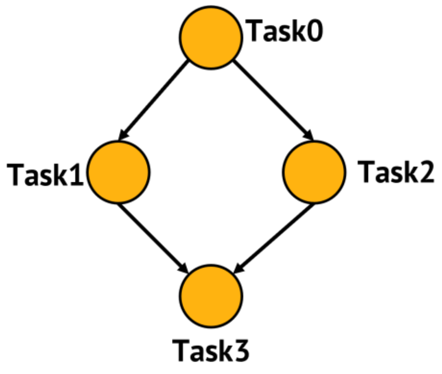

Figure 1: Structure of the Graph

Assume that each task can execute on Arm A53 or Arm A57 core. The GNN is trained to schedule the tasks of a given application to these cores. Its goal is to determine which core (A53 or A57) should perform a given task to minimize the execution time of the entire application.

The neural network embedded in each node in Figure 1 has 4 inputs, 4 neurons in the hidden layer, and 2 outputs (each corresponding to a core), as shown in Figure 2. The neural network consists of multiply and accumulate operations as well as ReLu operations. For example, the output of neuron 4 (y_4) is computed as $y_4 = \sum_{i=0}^{3} x_i \times w_{i4}$, where w_{ij} denotes the weight for the link between neuron i and neuron j. The outputs of layer-1 operation go through ReLu operation, where $ReLu(x) = \max(0, x)$.

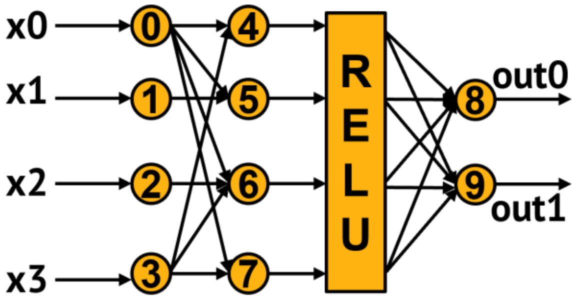

Figure 2: Structure of the Neural Network with fully connected layers. The figure does not show all the connections for brevity.

1. Milestone 1 (Due 3/2/22, Thursday)

The objective of MS-1 is to implement the inference of the neural network embedded in each node (Figure 2) using HDL (Verilog/System Verilog). Assume that all inputs are 5 bits and output values are 17 bits. Also, assume all inputs and the weights are signed. The most significant bit represents the sign bit. The top-level module should have the following module definition and input/output ports.

```
module top (x0, x1, x2, x3, w04, w05, w06, w07, w14, w15, w16, w17, w24, w25, w26, w27, w34, w35, w36,
w37, w48, w58, w49, w59, w68, w69, w78, w79, out0, out1, in_ready, out0_ready, out1_ready, clk);

input [4:0] x0, x1, x2, x3, w04, w05, w06, w07, w14, w15, w16, w17, w24, w25, w26, w27, w34, w35, w36, w37,
w48, w58, w49, w59, w68, w69, w78, w79;
input in_ready;

input clk;

output [16:0] out0, out1;
output out0_ready, out1_ready;

// Implementation of the neural network
endmodule
```

- x0, x1, x2, x3 are the inputs to the neural network (are provided in the testbench).
- wij are the weights for the link between neuron i and neuron j (are provided in the testbench).
- out0, out1 are the outputs as shown in Figure 2.
- The in_ready signal denotes when all the inputs are ready and the out0_ready, out1_ready signal indicates when the output is ready.
- **Each layer of the neural network as well as ReLu is executed at each clock cycles**. You **do not** need to consider a pipelined design.
- **Do not change the output value once you obtain the outputs**.

<u>Testbench and Simulation for MS1:</u>

- The testbench containing weights and inputs is uploaded in the canvas. Inputs in the testbench are in 2's compliment form.
- An Excel sheet with the output calculation is also uploaded in the canvas. The testbench and the sample screenshot correspond to 'Few Negative' tab in the Excel sheet.

- You can use ModelSim/QuestaSim in CAE machine to simulate your design. A sample script to invoke QuestaSim (top_compile_run.tcsh) is provided in Canvas (Files/Tutorials and Project) along with a ReadMe file.

<u>Submission guidelines for MS1:</u>

- Prepare a single submission for the entire group.
- Your design must satisfy the functionality. All parts of the design should be synthesizable (will be useful for MS3). The behavioral Verilog should generate results that matches exactly with the golden results. Your Verilog code will be tested using the provided testbench. **<u>Note: Your behavioral Verilog may be simulated with a different set of inputs or different outputs, so ensure that your module satisfies functionality for general cases.</u>**
- Submit the code and the screenshot of the waveform in .zip format through the submission link. One sample screenshot of the waveform is uploaded in the canvas. You can also submit a text file stating whether you incorporated any design choice that considers any optimization. The naming format of the zip file should be **<FullNameofOnePerson>_<FullNameofAnotherPerson>_DNN.zip**
- Additional guideline: Each layer of the DNN should be executed at each clock cycle. All neurons in a single layer should be performed parallelly (in the same clock cycle).

2. Milestone 2: Constructing a GNN in Verilog (deadline: 3/9/22)

Operation of a GNN

Graph Neural Network consists of aggregation and combination operations. The operations take place for each node of the GNN. First, for each node, the features of that particular node and the neighboring nodes are aggregated. Let us assume f_i^k denotes the features for Node-k (Task-k in Figure 1). In our case, $i = 0,1,2,3$ (four features) and $k = 0,1,2,3$ (four nodes/tasks). Node-0 is connected with Node-1 and Node-2. Therefore, the aggregated feature for Node-0 is $F_i^0 = f_i^0 + f_i^1 + f_i^2$. F_i^k denotes the aggregated feature at Node-k. The aggregated features are the inputs to the DNN of the corresponding node. After the layer-1 operation happens, the outputs of the layer-1 for each node are aggregated in same fashion as the input features. More details regarding the working principle of GNN can be found in https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications#:~:text=Graph%20Neural%20Networks%20(GNNs)%20are,and%20graph%2Dlevel%20prediction%20tasks.

Your goal is to implement the inference of a GNN in Verilog. The top-level module of the GNN should have the following module definition and input/output ports.

```
module top ( x0_node0, x1_node0, x2_node0, x3_node0,
        x0_node1, x1_node1, x2_node1, x3_node1,
        x0_node2, x1_node2, x2_node2, x3_node2,
        x0_node3, x1_node3, x2_node3, x3_node3,
        w04, w14, w24, w34,
        w05, w15, w25, w35,
        w06, w16, w26, w36,
        w07, w17, w27, w37,
        w48, w58, w68, w78,
        w49, w59, w69, w79,
        out0_node0, out1_node0,
        out0_node1, out1_node1,
        out0_node2, out1_node2,
        out0_node3, out1_node3,
        in_ready,
        out10_ready_node0, out11_ready_node0,
        out10_ready_node1, out11_ready_node1,
        out10_ready_node2, out11_ready_node2,
        out10_ready_node3, out11_ready_node3,
        clk);

input [4:0] x0_node0, x1_node0, x2_node0, x3_node0;
input [4:0] x0_node1, x1_node1, x2_node1, x3_node1;
input [4:0] x0_node2, x1_node2, x2_node2, x3_node2;
input [4:0] x0_node3, x1_node3, x2_node3, x3_node3;
input [4:0] w04, w14, w24, w34;
input [4:0] w05, w15, w25, w35;
input [4:0] w06, w16, w26, w36;
input [4:0] w07, w17, w27, w37;
input [4:0] w48, w58, w68, w78;
input [4:0] w49, w59, w69, w79;

input clk;

output [20:0] out0_node0, out1_node0;
output [20:0] out0_node1, out1_node1;
output [20:0] out0_node2, out1_node2;
output [20:0] out0_node3, out1_node3;
output out10_ready_node0, out11_ready_node0;
output out10_ready_node1, out11_ready_node1;
output out10_ready_node2, out11_ready_node2;
output out10_ready_node3, out11_ready_node3;

//Implementation of GNN

endmodule
```

- xi_nodej are the i^{th}inputs to the Node-j of the GNN (are provided in the testbench).
- wij are the weights for the link between neuron i and neuron j (are provided in the testbench; same as the DNN).
- outi_nodej are the i^{th}outputs of Node-j.
- The in_ready signal denotes when all the inputs are ready and the outxy_ready_nodej signal indicates when the output is ready at Node-j.
- **Each layer of the neural network is executed at each clock cycles**. **Aggregation operation for a particular layer happens at each clock cycles.**

Testbench and Simulation for MS2:

- The testbench containing weights and inputs is uploaded in the canvas. Inputs in the testbench are in 2's compliment form. Assume that all inputs are 5 bits and output values are 21 bits.
- An Excel sheet with the output calculation is also uploaded in the canvas. The testbench and the sample screenshot correspond to 'Few Negative' tab in the Excel sheet.
- You can use ModelSim in CAE machine to simulate your design.

Submission guidelines for MS2:

- Prepare a single submission for the entire group.
- Your design must satisfy the functionality. All parts of the design should be synthesizable (will be useful for MS3). The behavioral Verilog should generate results

that matches exactly with the golden results. Your Verilog code will be tested using the provided testbench. **Note: Your behavioral Verilog may be simulated with a different set of inputs or different outputs, so ensure that your module satisfies functionality for general cases.**

- Submit the code and the screenshot of the waveform in .zip format through the submission link. One sample screenshot of the waveform is uploaded in the canvas. You can also submit a text file stating whether you incorporated any design choice that considers any optimization. The naming format of the zip file should be **<FullNameofOnePerson>_<FullNameofAnotherPerson>_GNN.zip**

3. Milestone 3: Synthesis

- We will be using 7nm technology for synthesis.
- You are required to synthesize the GNN design submitted in MS2. You are welcome to modify the design for any optimization.
- The detailed guide will be uploaded later.

4. Milestone 4: Automatic Place & Route (APR)

- Refer to class lectures and the instruction/tutorial document that is provided for synthesis using Synopsys IC compiler and automatic place and route (APR) using Cadence Innovus Compiler.
- You are required to perform APR on the synthesized GNN design submitted in MS3. You are welcome to modify the design for any optimization.
- The detailed guide will be uploaded later.

5. Milestone 5: Post APR

- Post-APR-export GDS, import the GDS into Virtuoso layout, and perform DRC/LVS on the final layout of the design.
- The detailed guide will be uploaded later.

6. Milestone 6: Power/Performance Estimations

- Follow the Primetime instructions to measure the average power consumption.
- The detailed guide will be uploaded later.

7. Grading - 100 points total

For Milestone 1 Submission (20 points):

- **15 points for behavioral Verilog and simulation**: Your design must satisfy the functionality. The behavioral Verilog should generate results in the output that matches exactly with the golden results provided in the screenshot.
- **5 points for generality**: The grader will simulate your behavioral Verilog with a different set of weights or a different set of input features, so ensure that your module satisfies functionality for general inputs.

For Milestone 2 Submission (20 points):

- **15 points for behavioral Verilog and simulation**: Your design must satisfy the functionality. The behavioral Verilog should generate results in the output that matches exactly with the golden results provided in the screenshot.
- **5 points for generality**: The grader will simulate your behavioral Verilog with a different set of weights or a different set of input features, so ensure that your module satisfies functionality for general inputs.

For Milestone 3 Submission (20 points):

- **15 points for synthesis and post-synthesis Verilog simulation:** Your design must go through synthesis successfully. With your synthesized Verilog netlist (*.v), the same testbench should pass the functionality successfully.
- **5 points for generality**: The grader will simulate your netlist with a different set of weights or a different set of input features, so ensure that your module satisfies functionality for general inputs.

For Milestone 4 Submission (20 points):

- **15 points for initial APR:** Design must have gone through clock-tree synthesis, place & route procedures in Innovus correctly. Submit the corresponding files. Note that the exact values in the files are not subject to grading, but the outputs of these files will be checked whether Innovus went through properly.

- **5 points for generality**: The grader will simulate your post route netlist with a different set of weights or a different set of input features, so ensure that your module satisfies functionality for general inputs.

<u>For Milestone 5 Submission (10 points):</u>

- **10 points for post-APR:** Error-free reports.

<u>For Milestone 6 Submission (10 points):</u>

- **5 points for correctness and concise report:** Please give a concise report on your design commenting on the employed architecture, the initial design decisions you made for the design.
- **5 points for optimization quality:** Optimize your entire design for performance ("min_latency"), area (core layout area excluding power/ground rings), and power (reported by PrimeTime). All the submissions will be evaluated and sorted using the following formula:

$$\boldsymbol{EDAP = energy \times min\_latency \times area}$$

You would want to minimize the quality metric above. Points for optimization will be given based on the optimization.