Lab 4: ASIC Acceleration for Graph Convolutional Neural Networks (GCNs)

Milestone 1: Due 04/14/24, Wednesday, 11:59pm
Milestone 2: Due 04/30/24, Wednesday, 11:59pm
It is an individual task for each student to complete the entire design.

There are 4 stages for this lab.

Stage 1: Write the Verilog code for the GCN module, and verify the functionality.

Stage 2: Synthesize your design using Design Compiler and verify the synthesized netlist.

Stage 3: Perform automatic place-and-route (APR) using Innovus.

Stage 4: Post-APR simulation and run Innovus for power measurement.
The target is to complete a low-power GCN module, with end-to-end latency < 100ns.

NOTE: At every stage, please make different folders such as RTL/, Synthesis/, Innovus/.

1. Introduction

Graphs are data structures that model a set of objects (nodes) and their relationships (edges). Recent advancements have been made in using machine learning to analyze graphs due to their expressive nature for various applications, such as social networks, natural science, bioinformatics, knowledge graphs, recommendation systems etc. [1-8]. Graph Convolutional Neural Network (GCN) is one of the popular methodologies to perform machine learning on graph data. It transforms high-dimensional graph data to low dimensions without altering the graph structure. GCNs have two main operations:

(1) Feature Aggregation: Each node takes the features associated with its neighboring nodes and aggregates it to give a new feature matrix (FA). Mathematically, feature aggregation is a matrix vector multiplication of Adjacency matrix (A) and Feature matrix (F). An Adjacency matrix is a physical representation of the graph (Fig. 1 shows how a graph can be represented in the form of an Adjacency matrix) and a feature matrix represents all the feature vectors of each node, which is like the pixel values of an image in a traditional convolutional neural network used for image processing.

(2) Feature Transformation: Here the dimensionality reduction is performed. It can be represented as the multiplication between FA and a weight matrix and gives the output (Z). Then a traditional activation function is used to perform classification on the output.

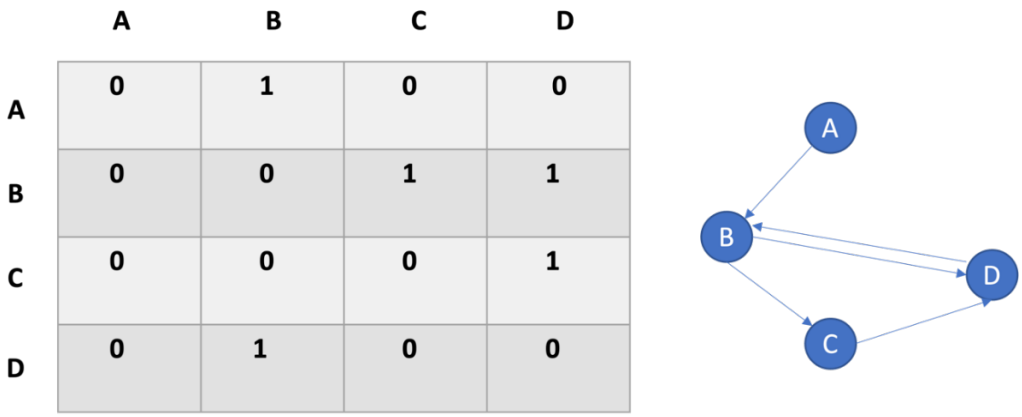

Figure 1. Representation of Adjacency matrix, for each edge between two nodes '1' in the matrix denoted the presence of a directed edge and '0' denotes the absence of an edge in the Graph.

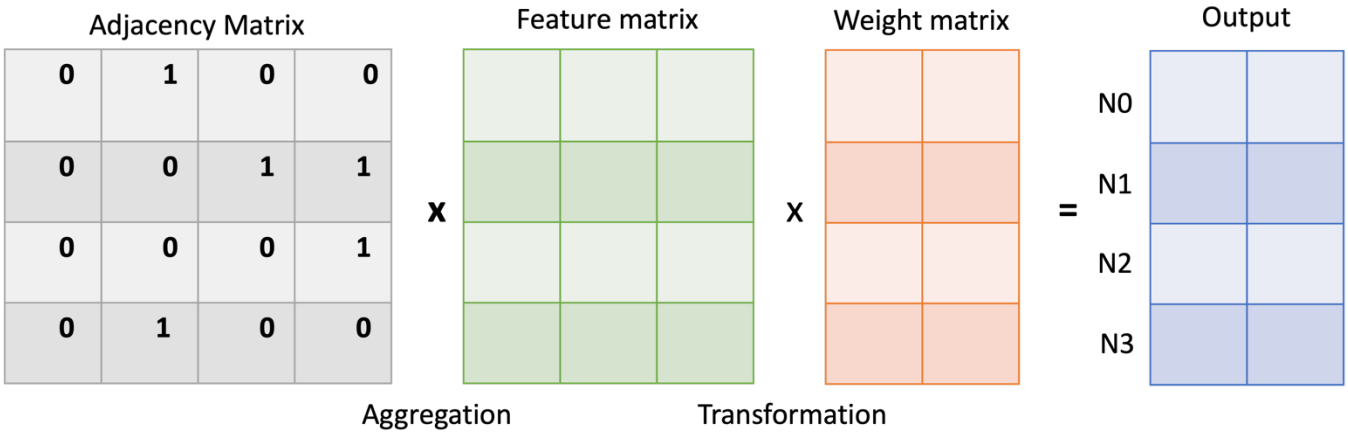

Figure 2. Overview of the GCN operation.

References:

[1] http://refhub.elsevier.com/S2666-6510(21)00001-2/optemUwNLRh7P

[2] http://refhub.elsevier.com/S2666-6510(21)00001-2/sref153

[3] http://refhub.elsevier.com/S2666-6510(21)00001-2/sref7

[4] http://refhub.elsevier.com/S2666-6510(21)00001-2/sref56

[5] Hamaguchi, T., Oiwa, H., Shimbo, M., Matsumoto, Y., 2017. Knowledge transfer for outof-knowledge-base entities : a graph neural network approach. In: Proceedings of IJCAI, pp. 1802–1808

[6] David Duvenaud et al. Convolutional networks on graphs for learning molecular fingerprints. arXiv preprint arXiv:1509.09292, 2015.

[7] Hanjun Dai et al. Learning steady-states of iterative algorithms over graphs. In International conference on machine learning. PMLR, 2018.

[8] Rex Ying et al. Hierarchical graph representation learning with differentiable pooling. arXiv preprint arXiv:1806.08804, 2018.

Functionality and Design

In this lab, you will design a graph convolutional network (GCN) module for a node classification application. The input graph is given in Fig. 3 with ground truth label shown next to each node. Each node in the graph represents a movie. By using GCN, we classify each node into one of three genres: Action (A), Humor (H) and Family (F). Due to the associative nature of Matrix multiplication, either Feature transformation or aggregation can be performed first. As for the activation function you will use an **argmax function** (Argument max) to perform the classification on the final output.

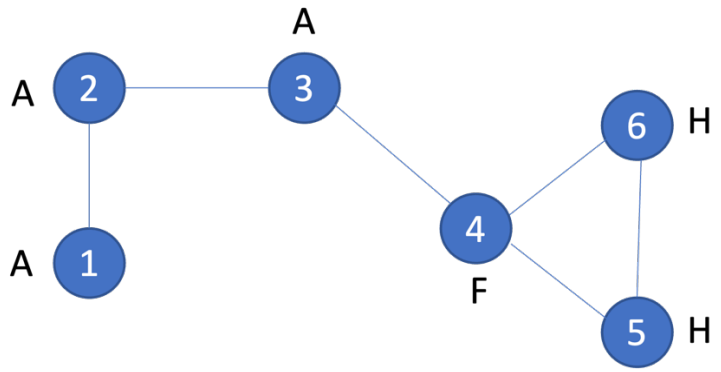

Figure 3. The graph for movie classification.

- Assume the input graph to be undirected. The adjacency matrix is completely binary.
- Since the adjacency matrix is fully binary and very sparse (much more 0's than 1's), it will be extremely inefficient to perform a matrix multiplication for aggregation. So, an alternative approach is to perform a vector addition of the transformed/untransformed features. You will be responsible for choosing and designing a hardware efficient sparse multiplier/adder to perform the aggregation/transformation. **Hint**: Think through ways to speed up the multiplication and addition, such as different multiplier and adder architectures for fast weighted sum computation.
- You will not be provided with the adjacency matrix. Instead, you will be provided with a Coordinate format (COO) of the adjacency matrix. In this version there are only two rows. This is a more efficient way of representation as it saves a lot of memory for large matrices.
- The **first row will have the source node** and the **second row will have the destination node**. The number of columns will represent the number of edges in the graph. You will be responsible for decoding the COO matrix to perform the desired computation. In the TB you'll have a multidimensional array denoting this.
- References for COO format:
    - https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html
    - https://scipy-lectures.org/advanced/scipy_sparse/coo_matrix.html

- https://en.wikipedia.org/wiki/Sparse_matrix

- The feature transformation is straightforward and will require a matrix multiplier.
- As for the final classification the argmax function is used. This function essentially compares all the values and returns the address/index of the max value. Visual illustration denoted in below figure.

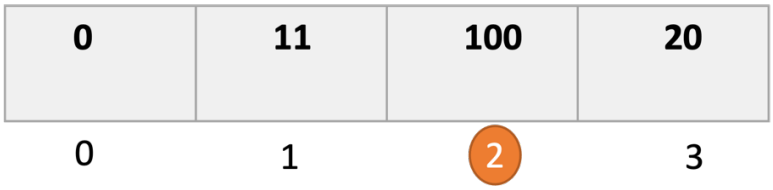

Figure 4. Argmax Function.

- Without any pipelining between the input and output flip-flops, your clock frequency could be relatively slow. However, with pipelining, your clock frequency would be relatively fast, and you could have a number of transformation + aggregation operations in flight, increasing your throughput. But with too much pipelining, the overhead of additional flip-flops would degrade both power and performance. You should try to find the sweet spot.
- As illustrated in the Fig. 5, all the inputs and outputs should be connected to input/output flip-flops at the boundary of the GCN module (no logic between the primary inputs/outputs and the boundary flip-flops). **Hint**: Make a parametrized DFF and use it for both inputs and outputs.
- The GCN module should assert the **enable_read,** with the **read_address**, to fetch the inputs from the memory instantiated in the testbench.
- On requesting the inputs for both the weight matrix and feature matrix, you will only be able to access one row of the feature or weight matrix at a time.
- Once the final classification is done by the GCN, the **done** signal should be asserted along with the max address of the final output matrix. The final output will be stored in **max_addi_answer**.
- It is then compared to the gold_address.txt file, which determines the correctness of the GCN module.
- You will be given 96 features so the resulting feature matrix will be 6x96 in size.
- The weight matrix is 96x3 in size. Each element in the feature and weight matrix is a 5-bit unsigned integer.
- The final output post argmax will be a 6x1 matrix each 3 bits wide denoting the index of each node.
- The **start signal** **will be given by the testbench** which is when your GCN module should start performing the computation. When every valid output is generated, it should generate **we signal** and the **corresponding address(s)** so that the memory in the testbench can capture it.

- One all the computation is done; the **done signal should be activated**. This done signal will be used to calculate the throughput.

The GCN module should have the following module definition and input/output ports.

```
module gcn
  #(parameter FEATURE_COLS = 96,
    parameter WEIGHT_ROWS = 96,
         parameter FEATURE_ROWS = 6,
    parameter WEIGHT_COLS = 3,
    parameter FEATURE_WIDTH = 5,
    parameter WEIGHT_WIDTH = 5,
    parameter DOT_PROD_WIDTH = 16,
    parameter ADDRESS_WIDTH = 13,
    parameter COUNTER_WEIGHT_WIDTH = $clog2(WEIGHT_COLS),
    parameter COUNTER_FEATURE_WIDTH = $clog2(FEATURE_ROWS),
    parameter MAX_ADDRESS_WIDTH = 2,
    parameter NUM_OF_NODES = 6,
    parameter COO_NUM_OF_COLS = 6,
    parameter COO_NUM_OF_ROWS = 2,
    parameter COO_BW = $clog2(COO_NUM_OF_COLS)
)
(
  input logic clk,
  input logic reset,
  input logic start,
  input logic [WEIGHT_WIDTH-1:0] data_in [0:WEIGHT_ROWS-1],
  input logic [COO_BW - 1:0] coo_in [0:1], //row 0 and row 1


  output logic [COO_BW - 1:0] coo_address,
  output logic [ADDRESS_WIDTH-1:0] read_address,
  output logic enable_read,
  output logic done,
  output logic [MAX_ADDRESS_WIDTH - 1:0] max_addi_answer [0:FEATURE_ROWS -
1]
);
```

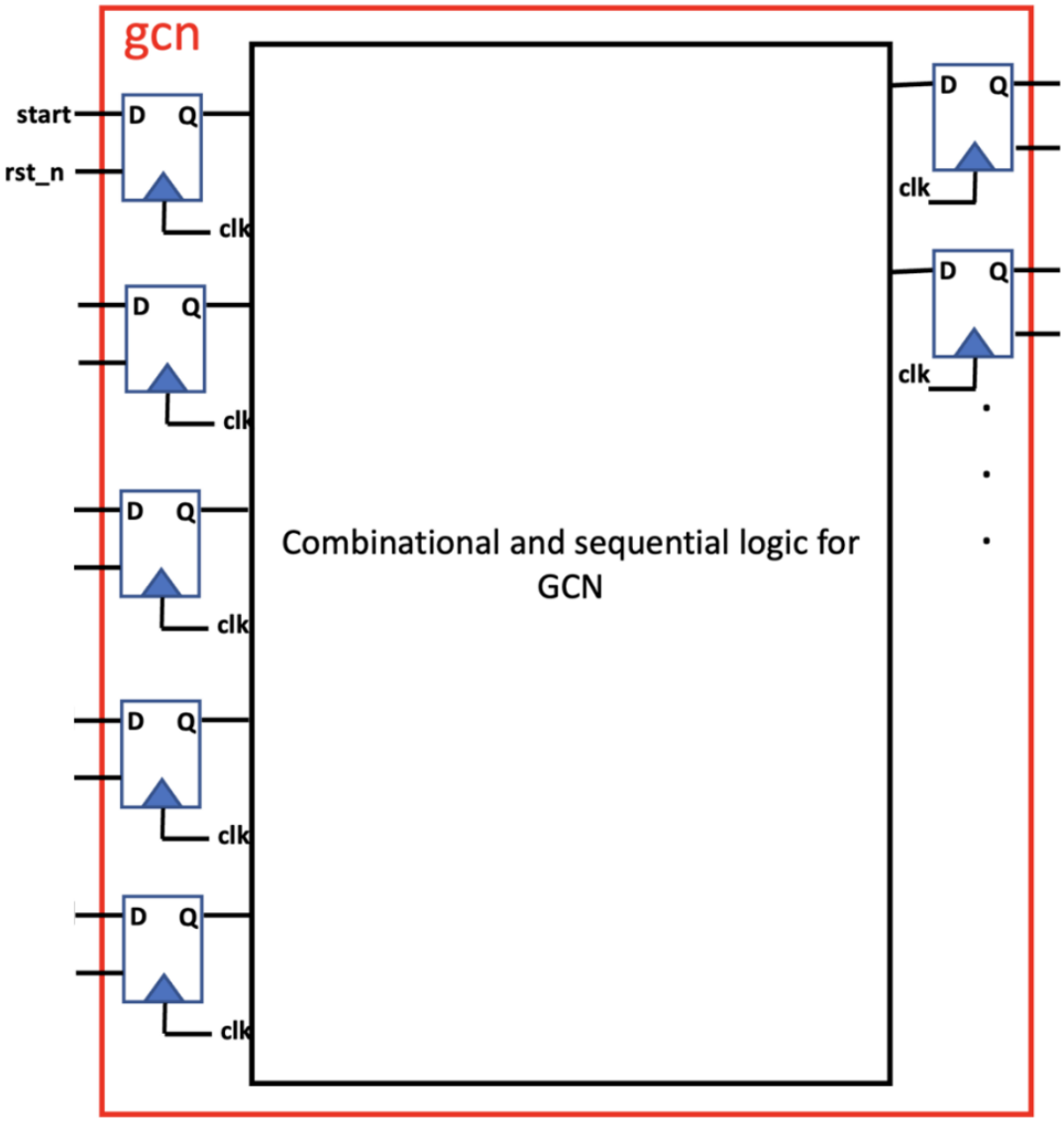

Figure 5. Illustration of the hardware module.

Testbench and Simulation

You should use the testbench Verilog file that instantiates the GCN module above and verifies the correct transformation and aggregation operations.

- There will be two testbenches, one for behavioral and one for Post-Synthesis and Post-APR. They are identical except for the fact that the Post-Synthesis and Post-APR testbench will transform packed arrays into unpacked arrays.
- The System Verilog testbench will have separate memories for the feature matrix, weight matrix and the sparse matrix.
- There is a golden output file provided in Lab 4, gold_address.txt, which has the expected **max_addi_answer** values for the given feature, weight, and adjacency matrix.

- As a preparation for power measurement, the following codes need to be included in your **testbench file**. Check and confirm it is in the testbench.

```
initial
  begin
    $dumpfile("current_output.vcd");
    $dumpvars(0,testbench);
  end
```

- To measure throughput, go through the following procedure.
    - As instructed, set your clock period from Innovus in your testbench file by adjusting the parameter **HALF_CLOCK_CYCLE** inside your testbench.
    - In the testbench simulation, simply record the total simulation, and this will be your **"Tot_latency"**. This time is essentially (clock period * number of cycles) -- the overall time it takes for your GCN module to go through transformation + aggregation operations. (You will also be asked to submit a screenshot that proves this.)

Synthesis and Automatic Place & Route (APR)

- Refer to class lectures and the instruction/tutorial document that is provided for synthesis using Synopsys Design Compiler (DC) and automatic place and route (APR) using Cadence Innovus. Those should be similar to what you did in Lab 3.

- A sample DC script is provided to you at /afs/asu.edu/class/e/e/e/eee525b/asap7_library/scripts/example.dc.tcl. Make necessary changes for your design and run the script to perform synthesis.

- Unlike Lab 3, this time you have more modules to be synthesized together, so you need to read them all into DC for analysis. (Hints: search how to **read a list of files** into DC)

- What you can change in the synthesis script includes: clock period, top level name, path to your Verilog files, and your Verilog file names. If necessary, you can modify the line which reads your files. **All other contents should not be modified.**

- File paths for technology and library files necessary for synthesis:

  /afs/asu.edu/class/e/e/e/eee525b/asap7libs/asap7libs_24/verilog
  /afs/asu.edu/class/e/e/e/eee525b/asap7libs/asap7libs_24/db

- A sample Innovus APR script is provided to you at /afs/asu.edu/class/e/e/e/eee525b/asap7_library/scripts/example_innovus.tcl. There are some useful commands in it, but not all. We suggest you to start with what you have done in Lab 3, and make necessary changes for your Lab 4 design to perform APR.

  If you prefer to use the Innovus automation script, ensure to understand each line and make necessary changes to suit your needs.

- You might need some commands to fix all kinds of violations which are not necessary in Lab 3, because Lab 4 is more complex. Try to search what commands you need and how to use them.
- Regarding the port location of the multiplier module, place all the input ports on the west and/or north and/or south side, and place all the output ports on the east side using Innovus.

Power Measurement

- Follow the Innovus instructions and refer to the user manual to measure the average power consumption that corresponds to your testbench operation using the saved *.vcd file.

Grading - 100 points total

There will be two milestones for this lab.

For Milestone 1, only the functionality and post synthesis simulation will be graded.

For Milestone 2, you will run AP&R and optimize the design to reduce the overall power consumption, under the latency constraint of 100ns for the GCN module. You will be graded on post AP&R simulation of the testbench, power analysis, and latency values. The design DRC and LVS correct in Innovus.

For Milestone 1 Submission (40 points)

- **15 points for behavioral Verilog**: Your design must satisfy the functionality. The behavioral Verilog should produce an output that matches exactly with gold_address.txt, and must go through synthesis successfully.
    - The grader could simulate your behavioral Verilog netlist with a feature matrix or a different weight matrix, so make sure that your module satisfies functionality for general inputs.
- **20 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. Submission requirements are:
    - Note: The grader could simulate your post-synthesis Verilog netlist with a feature or a different weight matrix.
- **5 points for concise report:** Please give a concise report on your GCN design commenting on the employed architecture, the initial design decisions you made for the GCN design.

For Milestone 2 Submission (60 points)

- **10 points for initial Innovus APR**: Design must have gone through clock-tree synthesis, place & route procedures in Innovus correctly. Submit the corresponding files depicted below. 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.
- **10 pts for APR, DRC/LVS of updated design:** Optimize your entire design for low power (reported by Innovus), and meet < 100ns performance (**"Tot_latency"**). Your design should still meet all of the submission criteria for Milestone 1 (Synthesis, APR), and should pass DRC and LVS without any error.
- **10 points for post-layout Innovus analysis:** Determine the power of your multiplier(s) using Innovus with the *.vcd file. The .vcd file should be created by running the testbench code with commands such as $dumpfile and $dumpvars. Then provide *.vcd with *.sdf/*.spef to Innovus and run proper commands to obtain the power consumption for your multiplier module.

- **10 pts for low-power optimization quality:** All the submissions will be evaluated and sorted based on the power consumption. Points for optimization will be given proportional to the rank of the submission, i.e., the project with rank *i* (=0, 1, 2, …) will get the following points for optimization.

$$Points(i) = \frac{Number\ of\ submissions - i}{Number\ of\ submissions} \times 10$$

- **10 pts for full report (with analysis of each part)**: A good report would present an optimization description on different parts of the GCN module (including any new techniques that you used to optimize the design), and the final result of the optimization. The report should conclude with a discussion of where the bottleneck is in terms of performance and power.
- **10 pts for demo:** Please note that we will consider only the files that are submitted on canvas on or before submission deadline. You will lose points for the demo if any of your files are corrupted or missing. If your sub-circuit instances point to other local libraries, you will lose points. Please double-check and triple-check these items. The following items will be examined during your demo:
    - Innovus layout
    - DRC / connectivity violations for your final Innovus layout
    - Functional correctness, items from your report.
    - You should be able to explain your design aspects during the demo.

Submission

Put all files in one Unix directory with the following organization:

< Milestone 1 >

Lab4_<studentID>_MS1

1) Report document

Architecture and high-level block diagram what is expected here.?

Concise report w/ initial design decisions (# of parallel multipliers, etc.)

2) Behavioral Verilog Netlist file And all supporting files. - yes what is expected here.? - gpt give code

3) Synthesized Verilog Netlist file Done

4) Synthesis report with top 5 critical paths What is synthesis report...??? Confirm critical paths

5) Testbench Verilog file GCN_TB and TB_Post_Syn...??

6) tb_gcn.syn.log: testbench simulation log file with synthesized Verilog netlist model sim command

< Milestone 2 >

Lab4_<studentID>_MS2

1) Report (follow template)

Architecture and high-level block diagram done

Concise report w/ final results

- Design decisions done
- Total latency (= total testbench simulation time from ModelSim)
  - Unit must be in “ms” (milli-seconds) done
  - Include screenshot from ModelSim done
- Power (from Innovus)
  - Unit must be in “mW” (milli-Watts) done
- Area (exclude power/ground rings, only the entire standard cells + filler cells area – show x and y dimension from screenshot)
  - Unit must be in “mm^2” done
- Innovus density (before filler cell insertion)
  - A value between 0 and 1 done
- # of gates: execute Innovus command “reportGateCount” and copy & paste the content in your report done

2) Behavioral Verilog Netlist done

3) Synthesized Verilog Netlist done

4) Innovus output Verilog netlist done

5) Testbench Verilog file done

6) tb_gcn.apr.log: testbench simulation log file with post-Innovus Verilog netlist done

Screenshot of ModelSim testbench simulation upon completion (when “Are you sure you want to finish” window pops up) – the final line in the screenshot should contain the total simulation time, and this should be the same value as the “Total latency” above in the report. done

7) gcn.ctsrpt: Innovus clock tree synthesis report file done

- This is generated after running clock tree synthesis (ckSynthesis command of example.apr.tcl file), or generate by running Innovus command reportClockTree -clk clk -report *.ctsrpt

8) gcn.drc.rpt: Innovus DRC violation report file done

- This is generated after running ‘verify_drc’

9) gcn.conn.rpt: Innovus connectivity violation report file done

- This is generated after running ‘verifyConnectivity’

10) gcn_postRoute.summary (from Innovus/timingReports directory) done

- Should only have positive slack (WNS) ??????

11) gcn_postRoute_hold.summary (from Innovus/timingReports directory) done

- Should only positive slack (WNS) ????????/

12) Post-layout power report from Innovus done

13) Top 3 worst timing paths from Innovus (setup and hold) done

14) Virtuoso Layout Screenshot done

15) Virtuoso DRC Screenshot done

16) Virtuoso LVS clean Screenshot done

Make sure none of the items above are omitted.

Then compress them to one tar file by using the following command.

```
tar -cjf Lab4_<studentID>_MS*.bz2 Lab4_<studentID>
```

Submit the *.bz2 file to canvas.