IIIT-B Chip Design Studio Weekly Report
Week: 1
Team Name/Project: Team Friday - RISC V Single Cycle CPU

Team Members:

Namai Chandra 23f3000200@es.study.iitm.ac.in
Madhur Thareja 23f3000178@es.study.iitm.ac.in
Aditya Prakash 23f3000345@es.study.iitm.ac.in
Vivek Dhariwal 23f3000338@es.study.iitm.ac.in

1. Updates

Current Progress:

Successful hands-on with the provided tools Genus and Innovus. Our CPU base design is complete, and both the RTL simulation and synthesis processes have been finalized. To practice using the software, we initiated Genus and Innovus with example files, such as implementing a counter and following the netlist-to-GDS workflow. The same methodology was applied to the proof-of-concept for our CPU design. The current Verilog code was modified by replacing the 'initial' blocks with 'always' blocks, ensuring compatibility with the synthesis process.

Challenges Faced:

1. Initially, understanding the new tools was somewhat challenging, and implementing the synthesis procedure proved to be difficult. Custom scripts were created to define custom paths to the library files, enabling efficient elaboration of Genus commands.

2. The original Verilog code for the CPU included 'initial' blocks, which are not synthesizable, causing errors during the 'elaborate' process. This issue was resolved by replacing the 'initial' blocks with 'always' blocks, ensuring compatibility with the synthesis process.

3. The detailed aspects of Innovus, including floor planning parameters, required thorough study, as they were not initially intuitive.

4. While understanding the theoretical concepts was manageable, implementing the concepts like hardware acceleration, nano power management system and TinyML on the CPU is something we are still working on when it comes to the operations.

2. GitHub Link: Provide the GitHub repository link for the project, if any:

Repository: https://github.com/NamaiBest/iiitb_chip_design_studio

3. Project Idea

The implementation of this CPU design can be approached in various ways, each with its own set of advantages and limitations:

1. Block Diagram Approach
2. Module Instantiation Approach
3. Single File Approach

For our design, we opted for the **module instantiation approach.** This method involves creating separate Verilog modules for each functional block and integrating them into the CPU module. It combines the modularity of the block diagram approach with the added benefits of a structured and scalable design. This approach simplifies the CPU module code, making it easier to debug. However, it does require managing and maintaining multiple modules, which can add complexity.

Currently, we aim to introduce new instruction sets to our existing CPU model to enable **TinyML** implementation and enhance hardware acceleration using various algorithms. TinyML operates locally within a system, eliminating the need for cloud connectivity and uses edge computing. It works with a limited, optimized dataset, which reduces latency. To address storage constraints, we plan to integrate external storage, such as an SRAM or a similar storage device. While establishing a communication system between the CPU and the storage block may pose challenges, we intend to resolve this in future developments.

ML is a field which allows computers to adapt to certain situations and react to changes. This is made possible by the training, during which the computer actually learns how to behave in a specific task with variable parameters and input data. The known input data set is applied to the network many times and each time is called **epoch**.

In neural networks, a neuron's output is called activation, and the connections between neurons are termed weights. One specific type of NN that we plan to implement on our CPU is the CNN, **Convolutional Neural Networks** (CNNs) are widely used for their excellent results and efficient acceleration. Each filter in a CNN extracts specific features from the input, producing a feature map that highlights where the feature is detected. The number of output feature maps corresponds to the number of filters in a layer, with the weights tuned during training to optimize feature recognition.

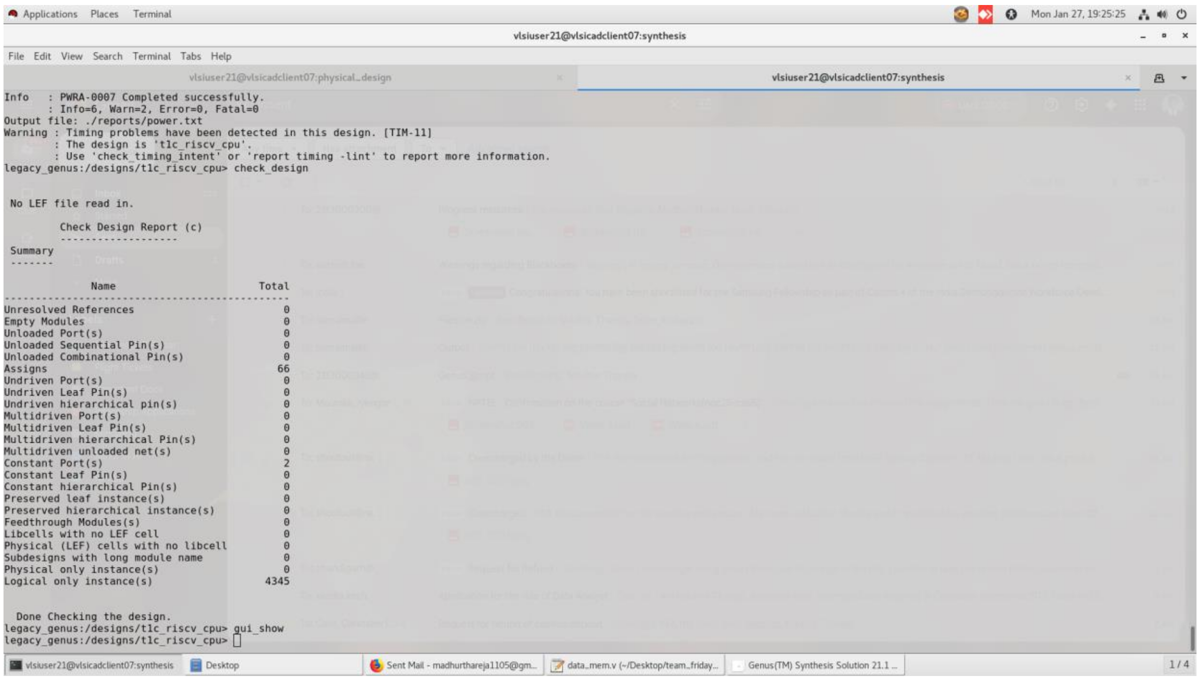
We also plan to incorporate a *low-power, self-sustaining RISC-V CPU architecture* inspired by ONIO.zero, which is powered entirely by ambient energy, showcasing sustainable technology through energy harvesting.

Incorporating all the above factors will not only add a novelty factor but also venture into a domain that few have explored.

4. Schematic/Simulations

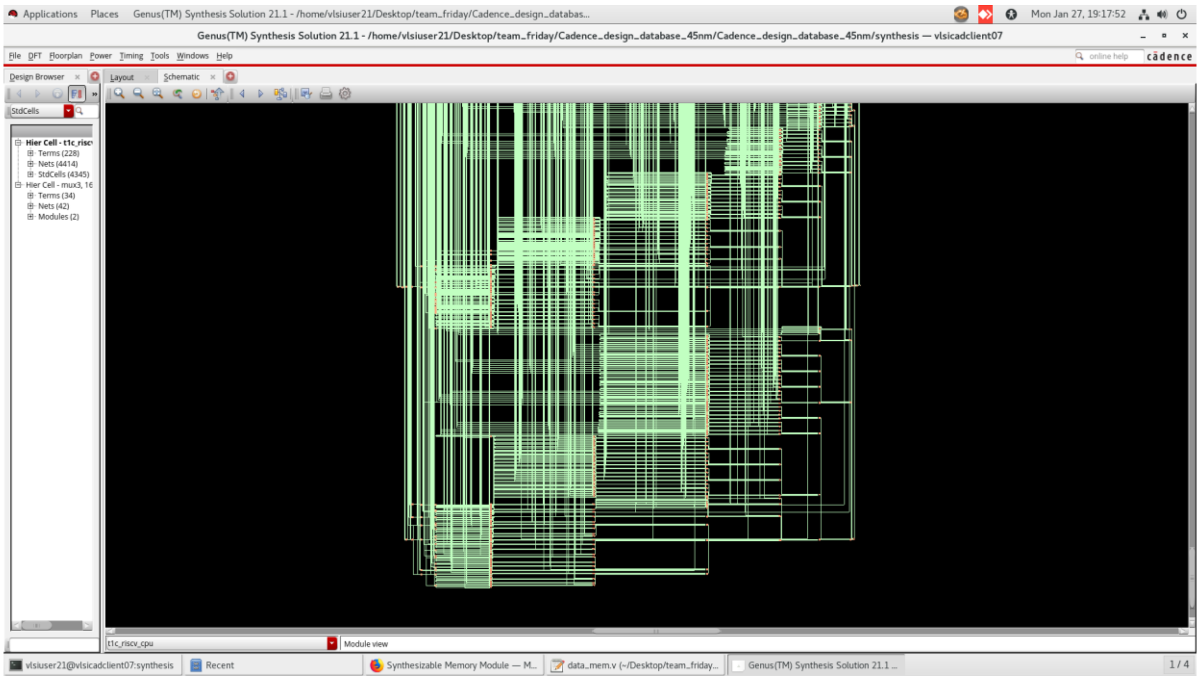
Include schematics, simulation results, or any relevant visuals:

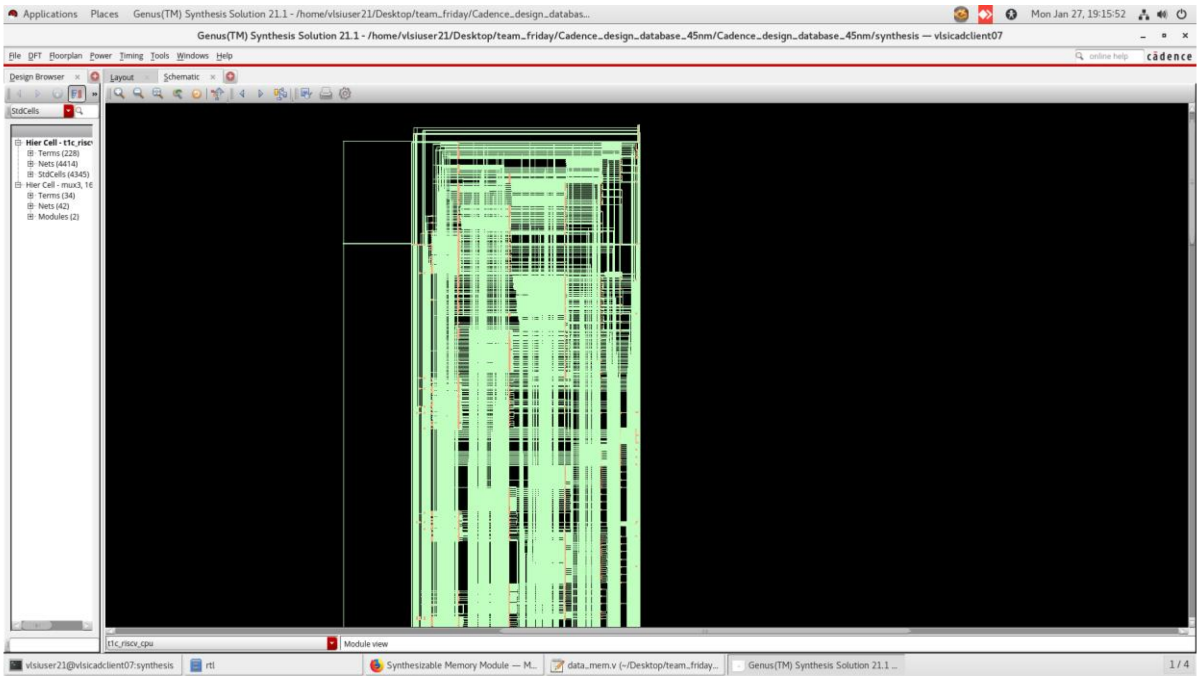
- **Schematics & Simulation Results:**

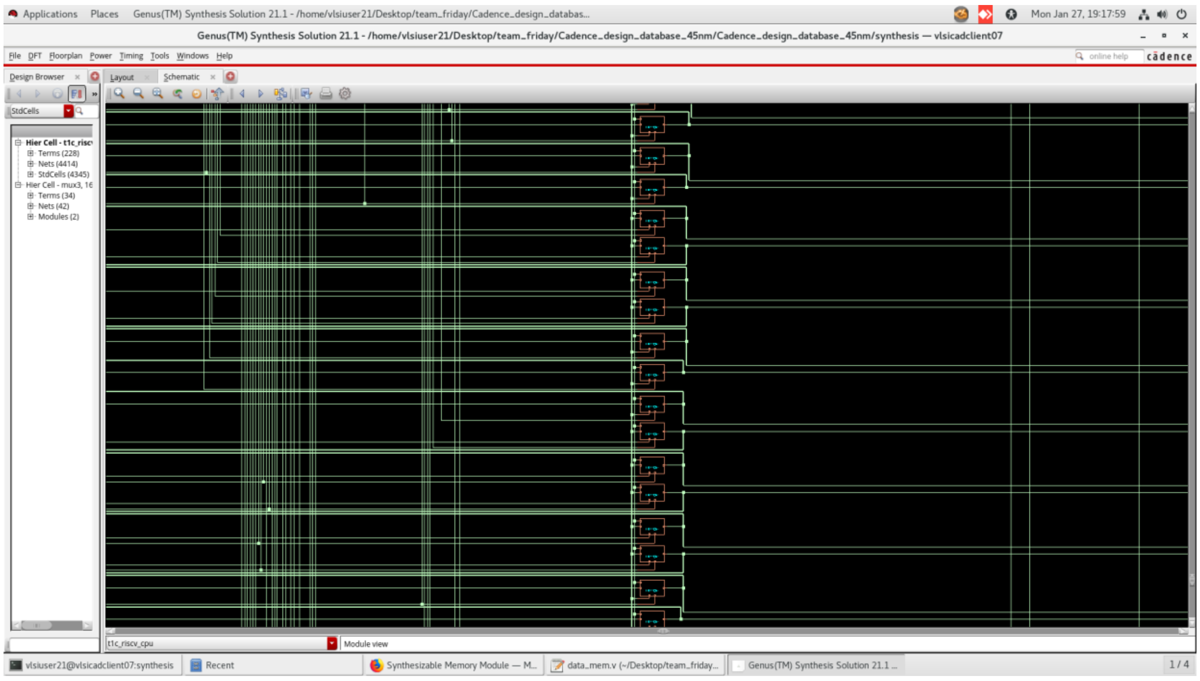


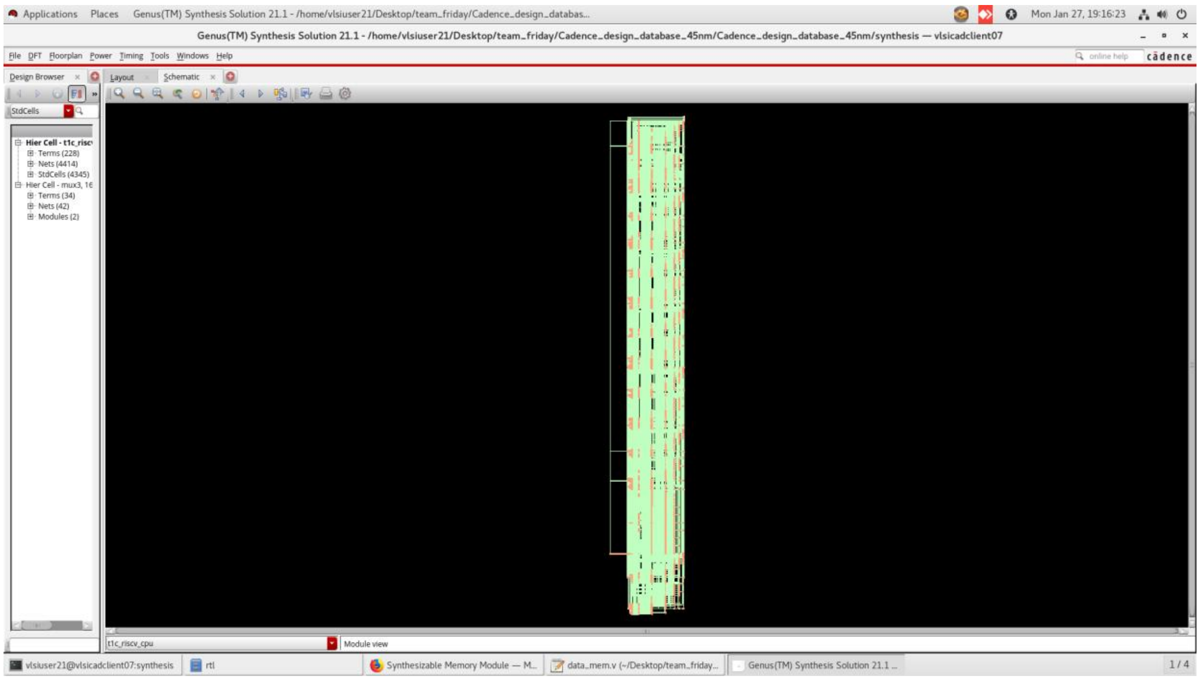
```
Info    : PWRA-0007 Completed successfully.
        : Info=6, Warn=2, Error=0, Fatal=0
Output file: ./reports/power.txt
Warning : Timing problems have been detected in this design. [TIM-11]
        : The design is 'tlc_riscv_cpu'.
        : Use 'check_timing_intent' or 'report timing -lint' to report more information.
legacy_genus:/designs/tlc_riscv_cpu> check_design


 No LEF file read in.

         Check Design Report (c)
         -------------------
 Summary
 -------

              Name                        Total
-----------------------------------------------
Unresolved References                         0
Empty Modules                                 0
Unloaded Port(s)                              0
Unloaded Sequential Pin(s)                    0
Unloaded Combinational Pin(s)                 0
Assigns                                      66
Undriven Port(s)                              0
Undriven Leaf Pin(s)                          0
Undriven hierarchical pin(s)                  0
Multidriven Port(s)                           0
Multidriven Leaf Pin(s)                       0
Multidriven hierarchical Pin(s)               0
Multidriven unloaded net(s)                   0
Constant Port(s)                              2
Constant Leaf Pin(s)                          0
Constant hierarchical Pin(s)                  0
Preserved leaf instance(s)                    0
Preserved hierarchical instance(s)            0
Feedthrough Modules(s)                        0
Libcells with no LEF cell                     0
Physical (LEF) cells with no libcell          0
Subdesigns with long module name              0
Physical only instance(s)                     0
Logical only instance(s)                   4345


  Done Checking the design.
legacy_genus:/designs/tlc_riscv_cpu> gui_show
legacy_genus:/designs/tlc_riscv_cpu> 
```

Successful Elaboration and Optimized Netlist Generation

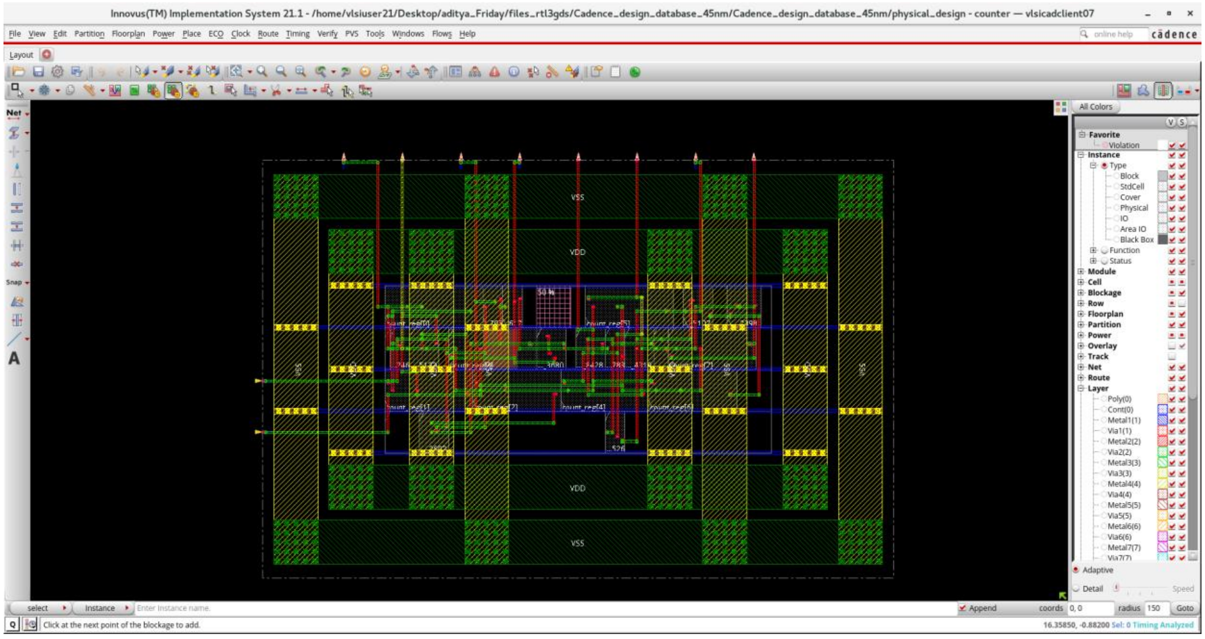

Genus(TM) Synthesis Solution 21.1 - /home/vlsiuser21/Desktop/team_friday/Cadence_design_database_45nm/Cadence_design_database_45nm/synthesis — vlsicadclient07
File DFT Floorplan Power Timing Tools Windows Help
Design Browser
Layout
Schematic
StdCells
Hier Cell - t1c_riscv
Terms (228)
Nets (4414)
StdCells (4345)
Hier Cell - mux3, 16
Terms (34)
Nets (42)
Modules (2)
t1c_riscv_cpu
Module view
cadence


Applications Places Genus(TM) Synthesis Solution 21.1 - /home/vlsiuser21/Desktop/team_friday/Cadence_design_databas...
Mon Jan 27, 19:15:52
Genus(TM) Synthesis Solution 21.1 - /home/vlsiuser21/Desktop/team_friday/Cadence_design_database_45nm/Cadence_design_database_45nm/synthesis — vlsicadclient07
File DFT Floorplan Power Timing Tools Windows Help
cadence
Design Browser
Layout
Schematic
StdCells
Hier Cell - t1c_riscv
Terms (228)
Nets (4414)
StdCells (4345)
Hier Cell - mux3, 16
Terms (34)
Nets (42)
Modules (2)
t1c_riscv_cpu
Module view
vlsiuser21@vlsicadclient07:synthesis
rtl
Synthesizable Memory Module — M...
data_mem.v (~/Desktop/team_friday...
Genus(TM) Synthesis Solution 21.1 ...
1 / 4


Applications Places Genus(TM) Synthesis Solution 21.1 - /home/vlsiuser21/Desktop/team_friday/Cadence_design_databas...
Mon Jan 27, 19:17:59
Genus(TM) Synthesis Solution 21.1 - /home/vlsiuser21/Desktop/team_friday/Cadence_design_database_45nm/Cadence_design_database_45nm/synthesis — vlsicadclient07
File DFT Floorplan Power Timing Tools Windows Help
cadence
Design Browser
Layout
Schematic
StdCells
Hier Cell - t1c_riscv
Terms (228)
Nets (4414)
StdCells (4345)
Hier Cell - mux3, 16
Terms (34)
Nets (42)
Modules (2)
t1c_riscv_cpu
Module view
vlsiuser21@vlsicadclient07:synthesis
Recent
Synthesizable Memory Module — M...
data_mem.v (~/Desktop/team_friday...
Genus(TM) Synthesis Solution 21.1 ...
1 / 4


Genus(TM) Synthesis Solution 21.1 - /home/vlsiuser21/Desktop/team_friday/Cadence_design_database_45nm/Cadence_design_database_45nm/synthesis — vlsicadclient07
File DFT Floorplan Power Timing Tools Windows Help
Design Browser
Layout
Schematic
Hier Cell - t1c_riscv
Terms (228)
Nets (4414)
StdCells (4345)
Hier Cell - mux3, 16
Terms (34)
Nets (42)
Modules (2)
t1c_riscv_cpu
Module view
Mon Jan 27, 19:16:23

Partial RTL to GDS -


Innovus(TM) Implementation System 21.1 - /home/vlsiuser21/Desktop/aditya_Friday/files_rtl3gds/Cadence_design_database_45nm/Cadence_design_database_45nm/physical_design - counter — vlsicadclient07
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
Layout
VSS
VDD
All Colors
Favorite
Violation
Instance
Type
Block
StdCell
Cover
Physical
IO
Area IO
Black Box
Function
Status
Module
Cell
Blockage
Row
Floorplan
Partition
Power
Overlay
Track
Net
Route
Layer
Poly(0)
Cont(0)
Metal1(1)
Via1(1)
Metal2(2)
Via2(2)
Metal3(3)
Via3(3)
Metal4(4)
Via4(4)
Metal5(5)
Via5(5)
Metal6(6)
Via6(6)
Metal7(7)
Via7(7)
Click at the next point of the blockage to add.
16.35850, -0.88200 Sel: 0 Timing Analyzed

5. Analysis

Cell Count:

- Library Cells Available: 324 combinational, 128 sequential cells (total 452 usable cells).

Used Cells:

- Initial Stage: 67 instances with 352 units of area.
- Post Optimization: 194 instances with 517 units of area.

Delay Parameters:

- Typical Gate Delay: 127.6 ps.
- Standard Slew Rate: 17.9 ps.
- Standard Load: 1.0 fF.

Runtime and Resource Utilization:

- Total runtime for synthesis stages: ~20 seconds (split across stages such as mapping and cleanup).
- Memory Utilization: 251.6 MB across all synthesis stages.
- Threads Utilized: 8 out of 12 available CPUs.

- **Key Findings**

Hardware acceleration can be implemented to optimize performance by offloading computational tasks to specialized hardware units, potentially enhancing processing speed. Key findings include identifying suitable algorithms for acceleration and exploring ways to optimize resource utilization. Some algorithms are -

1. Matrix Multiplication: Commonly used in machine learning and scientific computations, matrix multiplication can be accelerated using dedicated hardware units like vector processors or SIMD (Single Instruction, Multiple Data) units.
2. Neural Network Inference (TinyML): Hardware accelerators designed for neural network computations can dramatically speed up TinyML tasks, reducing inference time for edge devices.

On the research level we discovered that TinyML integration can be done with frameworks like TensorFlow that support a wide range of cores and devices and provide libraries that allow for an easy and fast development of new DL models in many different programming languages, based on what is needed. This is why a format called Open Neural Network eXchange (ONNX) has been adopted as standard accepted input format for the model, so that the design and training of a NN can be done with any framework of choice without any problem. We can use this to further optimize the limited memory in our CPU.

Insights or Learnings

The project utilized Cadence tools, with Genus specifically employed for achieving synthesis and Innovus for further RTL to GDS that we are working on optimizing even further. Key learnings included writing scripts in Bash for efficient workflow, synthesis workflows, debugging Verilog modules, and managing floor planning parameters for efficient chip design, such as adding power management rings, optimizing routing, and other critical design considerations.

We became familiar with working with applications at the terminal level, which was a different from the GUI typically provided by most applications.

- **Improvements or Modifications Needed**

As mentioned earlier, the hardware acceleration and TinyML modules have not yet been integrated and is still under development.

Quantization could also be a viable option to reduce memory requirements and lighten the computational cost. The exact hardware acceleration algorithm has not been finalized but will be presented in the week 2 report. In addition to hardware acceleration, memory acceleration may also need to be implemented on the CPU, along with additional ports to enable communication with an external storage device that holds the trained data for TinyML.

Similarly, the final machine learning algorithm for the CPU is expected to be presented by the end of week 3.