# Report for the Power/Performance Estimations of GNN

## 1. Design Overview

In the RTL design, considering that deeper hidden layers have increased bitwidth, which results in more logic delay for multiplication and sum(), I added flip-flop stages to the DNN with a tendency to relieve the deeper levels' critical paths. Moreover, I implemented an RTL design of DNN with flexible bitwidth and flip-flop stage insertion, as shown in the MS1-MS5 submissions. GNN designs with different frequencies are implemented from the behavioral to the physical layout level. They are then evaluated and compared in this milestone.

## 2. Design Comparison by PT

From the *reports_730, reports_883, and reports_970*, the PT reports for the 2, 4, and full flip-flop stage designs are included. The required metrics are listed below:

| **Design** | **Area(mm^2)** | **Max Frequency(MHz)** | **Min Latency(ns)** | **Power(mW)** | **Energy(pJ)** | **EDAP(pJ ns-mm^2)** |
|---|---|---|---|---|---|---|
| full-stages | 0.00244 | 970 | 7.22 | 6.89 | 49.75 | 0.876 |
| 4-stages | 0.0022 | 883 | 4.53 | 5.79 | 26.23 | 0.261 |
| 2-stages | 0.00197 | 726 | 2.75 | 3.86 | 10.61 | 0.057 |

From the table, we can see that the high-frequency design performs poorly in **latency** but performs well in **throughput**, which is not covered in this project's specifications. Additionally, latency is quadratic to EDAP. Moreover, high-frequency design requires more area for flip-flops and routing effort to meet timing closure. Meanwhile, high-frequency mode also indicates larger power. So, considering the EDAP metric, the 2-stage design stands out. **This is important because further optimization is based on these findings from the initial designs and evaluation.**

## 3. Optimization and improved results

### 3.1. Outcome

According to the files under *reports_opt724* folder, the improved results are listed below:

| **Design** | **Area(mm^2)** | **Max Frequency(MHz)** | **Min Latency(ns)** | **Power(mW)** | **Energy(pJ)** | **EDAP(pJ ns-mm^2)** |
|---|---|---|---|---|---|---|

| Design | Area(mm^2) | Max Frequency(MHz) | Min Latency(ns) | Power(mW) | Energy(pJ) | EDAP(pJ ns-mm^2) |
|---|---|---|---|---|---|---|
| improved | 0.00193 | 724 | 1.38 | 3.65 | 5.037 | 0.0134 |

The updated simplified design architecuture is shown in Figure 1:

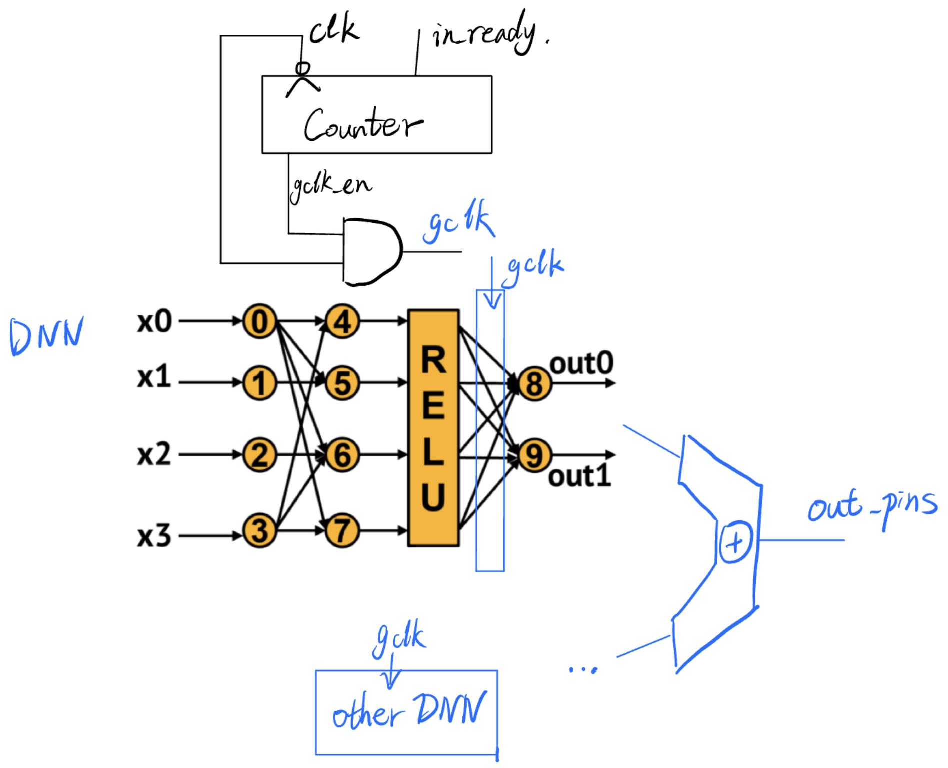


Figure 1. Simplified design architecture

## 3.2. Optimization

The optimization is based on the following two aspects:

### 3.2.1. Flip-flops stage

From Figure 1, we can see that only one stage of flip-flops is inserted into the design, located at a deeper level, so that it can effectively decrease the critical path and latency cycle. As a result, from the outcome table, we can see that latency is decreased from 2.74 to 1.38, while the frequency does not change significantly. However, the configuration for APR is different, so it may have some cost in the area. Nonetheless, fewer register requirements also decrease the area. The area is reduced from 0.00197 to 0.00193.

### 3.2.2. Gated Clock

From Figure 1, we can see that a gated clock is inserted into the design to decrease power. However, implementing a gated clock is not as simple as adding some logic. One challenging aspect is in the further Synthesis and APR:

1. In Synthesis and APR, we should add more timing constraints for analysis because it is now a multi-clock design. For example, I added the following constraints for the DC shell:

```
# Find the destination pin of the clk_gated signal
set clk_gated_net [get_nets gclk]
set clk_gated_pin [get_pins -of_objects $clk_gated_net]
# Define gated clock
create_generated_clock -name clk_gated -source [get_ports clk] -divide_by 1
$clk_gated_pin

set_clock_groups -asynchronous -group [get_clocks clk] -group [get_clocks clk_gated]
```

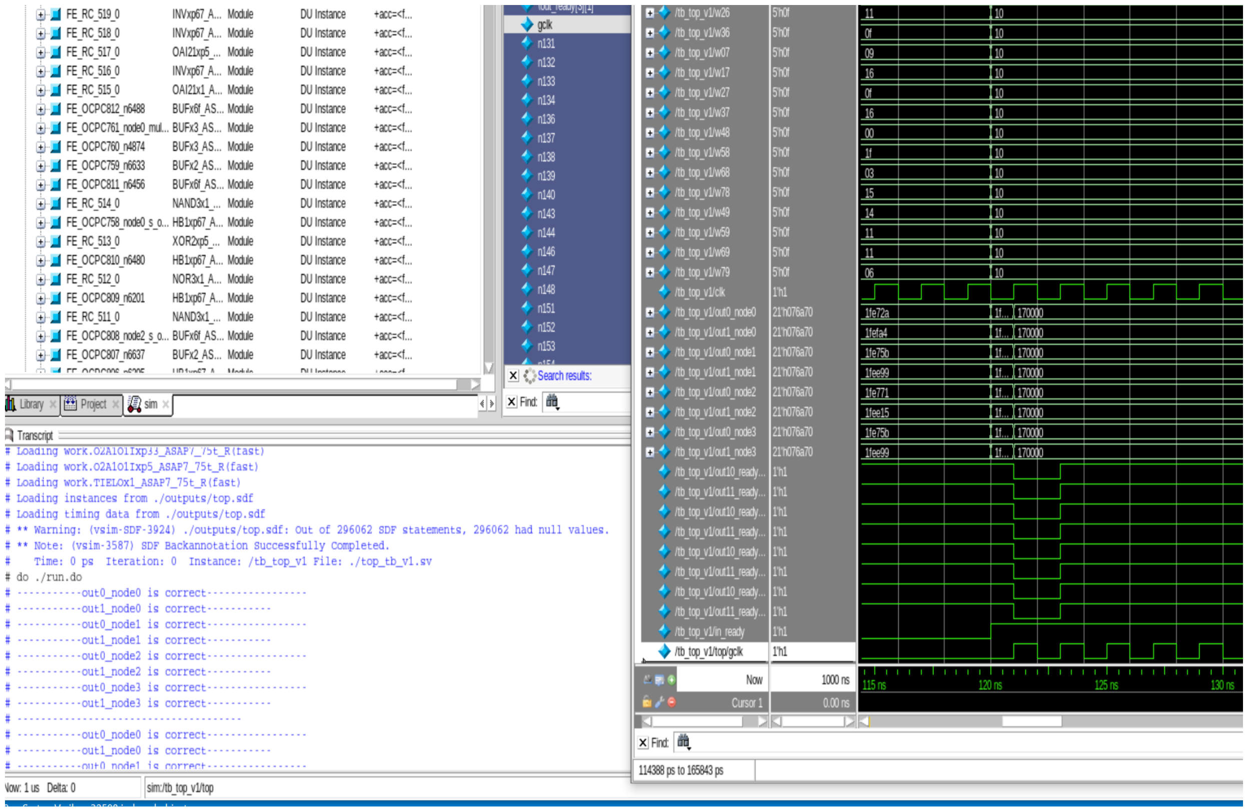
To clarify that, I also attached the sdc file.

2. To avoid contention, from Figure 1, we can see that I generated the gclk_en by clk_negedge. Otherwise, it would encounter hold time violation.

As a result, we can see a drop in power from 3.86 to 3.65, while the timing analysis still passes.

## 3.3. Validation

Since I have changed the design, I need to go through MS2-MS5 again to validate the design and generate outputs. In short, here I just show the post-simulation results similar to MS3. Other internal reports have been uploaded to the GitHub repo. If they are required, please check the *optimized_* folders.



```
FE_RC_519_0          INVxp67_A...  Module  DU Instance  +acc=<f...
FE_RC_518_0          INVxp67_A...  Module  DU Instance  +acc=<f...
FE_RC_517_0          OAI21xp5_...  Module  DU Instance  +acc=<f...
FE_RC_516_0          INVxp67_A...  Module  DU Instance  +acc=<f...
FE_RC_515_0          OAI21x1_A...  Module  DU Instance  +acc=<f...
FE_OCPC812_n6488     BUFx6f_AS...  Module  DU Instance  +acc=<f...
FE_OCPC761_node0_mul... BUFx3_AS...  Module  DU Instance  +acc=<f...
FE_OCPC760_n4874     BUFx3_AS...   Module  DU Instance  +acc=<f...
FE_OCPC759_n6633     BUFx2_AS...   Module  DU Instance  +acc=<f...
FE_OCPC811_n6456     BUFx6f_AS...  Module  DU Instance  +acc=<f...
FE_RC_514_0          NAND3x1_...   Module  DU Instance  +acc=<f...
FE_OCPC758_node0_s_o... HB1xp67_A...  Module  DU Instance  +acc=<f...
FE_RC_513_0          XOR2xp5_...   Module  DU Instance  +acc=<f...
FE_OCPC810_n6480     HB1xp67_A...  Module  DU Instance  +acc=<f...
FE_RC_512_0          NOR3x1_A...   Module  DU Instance  +acc=<f...
FE_OCPC809_n6201     HB1xp67_A...  Module  DU Instance  +acc=<f...
FE_RC_511_0          NAND3x1_...   Module  DU Instance  +acc=<f...
FE_OCPC808_node2_s_o... BUFx6f_AS...  Module  DU Instance  +acc=<f...
FE_OCPC807_n6637     BUFx2_AS...   Module  DU Instance  +acc=<f...
```

```
# Loading work.O2A1O1Ixp33_ASAP7_75t_R(fast)
# Loading work.O2A1O1Ixp5_ASAP7_75t_R(fast)
# Loading work.TIELOx1_ASAP7_75t_R(fast)
# Loading instances from ./outputs/top.sdf
# Loading timing data from ./outputs/top.sdf
# ** Warning: (vsim-SDF-3924) ./outputs/top.sdf: Out of 296062 SDF statements, 296062 had null values.
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.
#    Time: 0 ps  Iteration: 0  Instance: /tb_top_v1 File: ./top_tb_v1.sv
# do ./run.do
# -----------out0_node0 is correct----------------
# -----------out1_node0 is correct----------
# -----------out0_node1 is correct----------------
# -----------out1_node1 is correct----------
# -----------out0_node2 is correct----------------
# -----------out1_node2 is correct----------
# -----------out0_node3 is correct----------------
# -----------out1_node3 is correct----------
# -------------------------------------
# -----------out0_node0 is correct----------------
# -----------out1_node0 is correct----------
# -----------out0_node1 is correct----------------
```

Figure 2. Post-simulation waveform

As shown in the waveform, the latency is 1 clk, the clock is gated and all outputs are correct.

## 4. Further Discussion

1. Regarding the verification part, I have generated the .sdf by PrimeTime for ModelSim to perform back-annotation. However, it seems that ModelSim cannot identify these parasitic delays. Is it due to version incompatibility? Moreover, I cannot open VCS on the CAE machine, which obstructs me from further verification, as we can see in Figure 3.
2. Further optimization options: In this design, 4 DNNs are implemented for GNN. However, DNNs can be shared to save area. It is a common shrinking method implemented by adding more timing control and result latching logic, but it will worsen the latency.
3. Limitation: this design lacks a reset signal from the specifications. Therefore, it cannot maintain some more complicated logic safely.