UNIVERSITY OF MINNESOTA

EE5324 VLSI Design-II

Project 3 - Milestone 2

8th May 2025

Submitted by :

Samson Hruday Chinta -5949682

Submitted to :

Prof . Yang (Katie) Zhao

Teaching Asst. Sayantan Ghosh

Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities

Contents

Conv Pooling

1.1 Introduction

The conv_pool module is an advanced hardware design intended to perform a 2D convolution operation followed by max-pooling, integrated into a refined **5-stage pipeline architecture**. It is specifically engineered for high-performance applications such as convolutional neural networks (CNNs) and real-time image processing tasks. The module accepts a 4×4 block of image pixels along with three distinct 3×3 convolution kernels and processes these inputs through an enhanced pipelined structure. This updated design, optimized for ASIC implementation in TSMC 16nm technology, prioritizes throughput, clock frequency scalability, and precise computation accuracy. With an improved 5-stage pipeline, the module efficiently manages data flow, significantly reducing latency and idle cycles, thus making it highly suitable for incorporation into sophisticated image processing frameworks requiring robust and continuous pixel throughput.

1.2 Functional Description

This module is responsible for:
1.Reading a 4×4 image block .
2.Applying 3×3 convolution using three different kernels.
3.Performing 2×2 max pooling on the resulting convolved output.
4.Outputting the pooled results with programmable dynamic range control via a shift operation.
Each image block is convolved and pooled independently. The module supports back-to-back processing of multiple image blocks using a pipelined architecture, which allows overlapping execution of image read, computation, and result write-back operations.

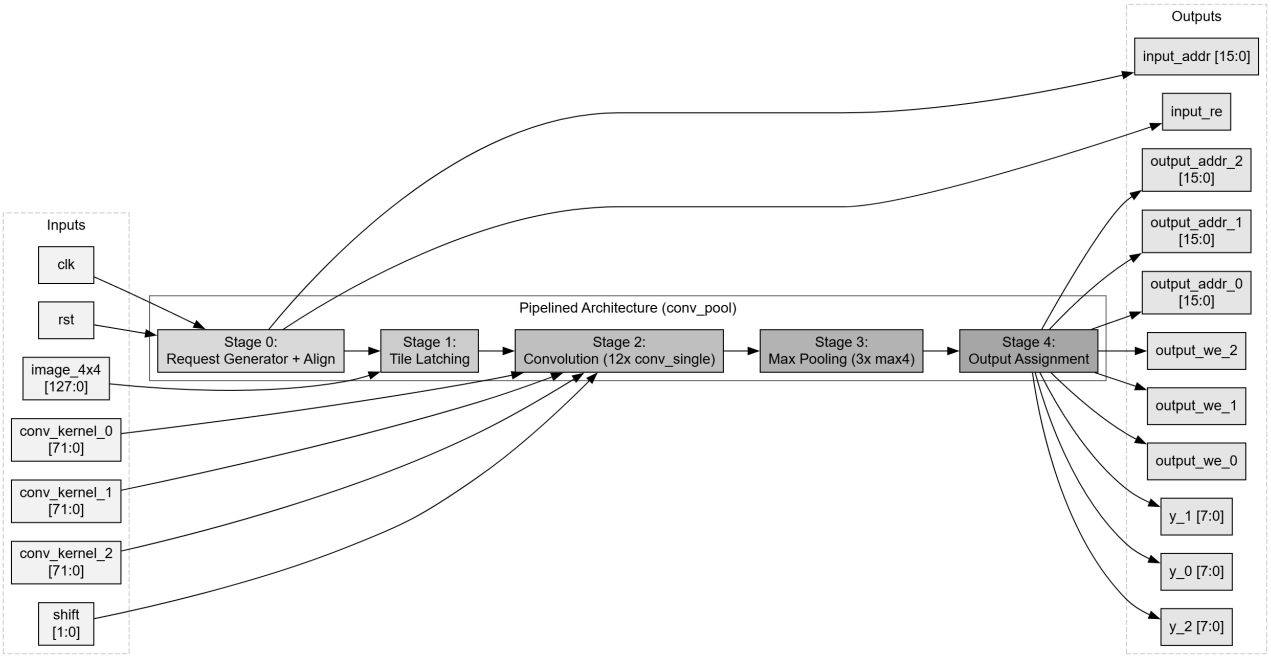

Figure 1: High-Level Pipeline Architecture of conv_pool

Design Considerations and Improvements

2.1 Pipeline Architecture

The `conv_pool` RTL is organized as a 5-stage pipeline for high-throughput 4×4 block convolution and pooling. Each stage has a clear role, from memory fetch all the way to write-back.

Stage 0: Request Generation & Alignment

Operation: Issue and align read requests to fetch 4×4 image blocks, handling the handshake between request and response channels.

Key Registers: `req_valid`, `req_address`, `req_valid_q`, `req_address_q`, `block_count`.

Outputs at Stage 0:
`input_re`: read-enable for image memory.
`input_addr`: address of the image block.

Stage 1: Input Data Latching

Operation: Capture the incoming 4×4 image block into internal registers for compute.

Key Registers: `lat_valid`, `lat_address`, `lat_tile`.

Inputs: `image_4x4`.

Stage 2: Convolution Computation

Operation:
1. Perform four 3×3 convolutions on the latched block for each of the three kernels.
2. Apply right-shift for range control and clip to [0,255].

Key Registers: `c0_xx`, `c1_xx`, `c2_xx`.

Computation Function: `conv3x3()` on each sub-window and kernel.

Pipelined Outputs: Intermediate convolution results.

Stage 3: Max-Pooling Computation

Operation:
Select the maximum from each set of convolution outputs and forward to pooling registers.

Key Registers: `pool_y0`, `pool_y1`, `pool_y2`.

Pipelined Outputs: Pooled max values.

Stage 4: Output & Memory Interface

Operation: Latch pooled results, generate write addresses, and assert write-enable for external memory.

Key Outputs: `output_we_0/1/2`, `output_addr_0/1/2`, `y_0`, `y_1`, `y_2`.

2.2 Convolution Design Considerations:

The convolution module in this design processes a 4×4 block of image pixels using three separate 3×3 kernels, each with signed 8-bit fixed-point coefficients (Q5.3 format). The input image block is 128 bits, representing 16 unsigned 8-bit pixels. For each kernel, the system performs convolution over four overlapping 3×3 regions within the 4×4 tile, generating four intermediate results. These are passed through a conv_single function, where unsigned pixel values are zero-extended and multiplied with signed kernel coefficients using proper sign handling. The multiplication results are accumulated into a 32-bit signed sum.

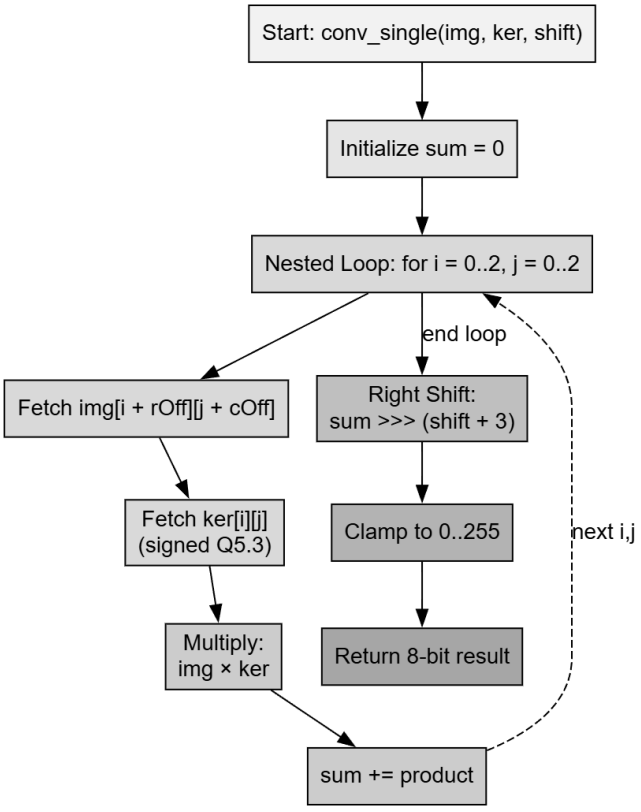

Figure 2: Convolution Computation Flowchart

After accumulation, each sum is right-shifted by a total of (shift + 3) bits to scale the output and align it with the fixed-point format. The result is then clamped to the 0–255 range to fit within an 8-bit unsigned output. This is done independently for all four regions and all three kernels, resulting in 12 values. A max-pooling operation selects the maximum of the four outputs for each kernel, producing the final three outputs (y_0, y_1, y_2). The design processes all three kernels in parallel within the pipeline, making it efficient for hardware acceleration. Inputs are read and outputs are written using synchronized memory interfaces, controlled by pipeline stage validity and address counters.

2.3 Memory Interface & Max pooling

The memory interface in the design is responsible for fetching 4×4 image blocks and writing back the processed outputs. The input memory access is managed through a request generator that uses a block counter to iterate over all tiles in the image. When enabled, it asserts the input_re signal and provides a unique input_addr for each tile, reading one block per cycle. The request is synchronized with the pipeline using a staging mechanism, ensuring that each fetched block moves through all pipeline stages in order. On the output side, when the final pipeline stage is valid, the module enables the corresponding write signals (output_we_0/1/2) and writes the processed outputs (y_0, y_1, y_2) to separate output addresses derived from the same counter.

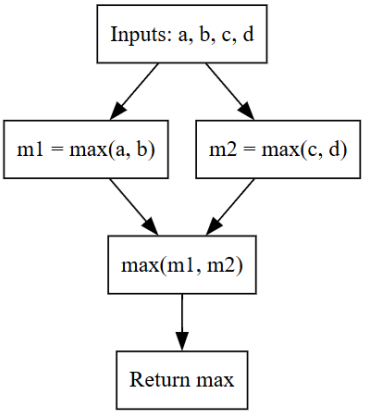

Figure 3: 2×2 Max-Pooling Flowchart

The max pooling logic follows the convolution step and selects the most prominent feature from the four 8-bit values computed per kernel. This is implemented using a max4 function, which compares the four convolution outputs and returns the maximum value. This operation ensures spatial downsampling while preserving the strongest activation in each region. Since each kernel has its own set of four outputs, max pooling is applied independently three times—once per kernel—resulting in the final three outputs. This structure allows efficient parallel processing and data reduction, well-suited for deeper image processing pipelines in hardware.

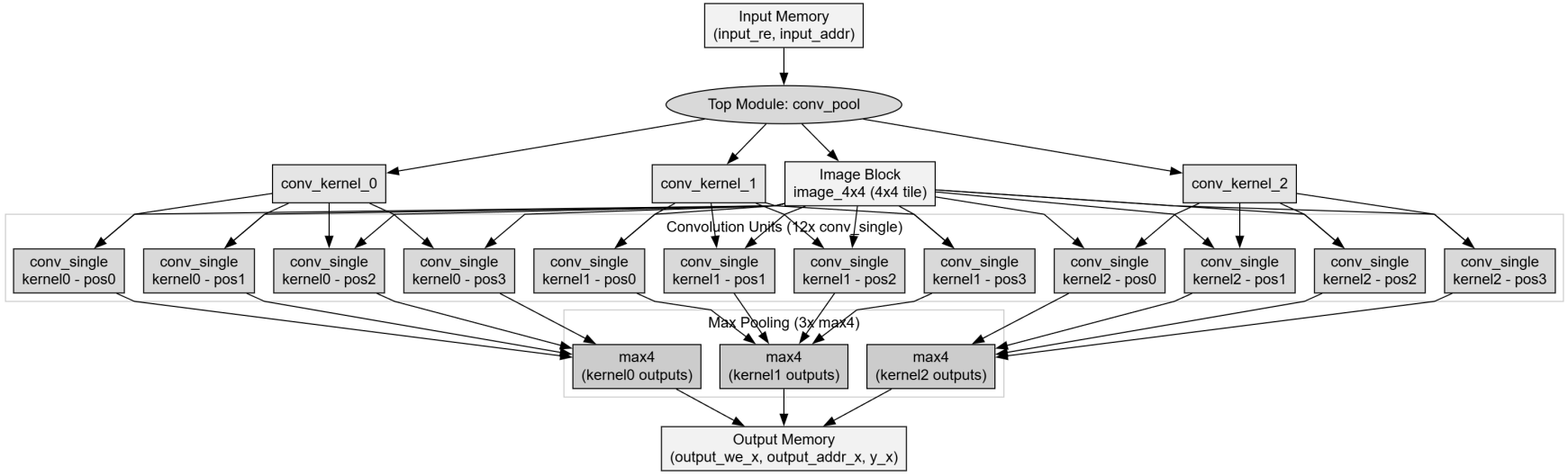

Figure 4: Overall conv_pool Module Structure

Results

3.1 Pre Synthesis simulation Results:

To verify the functionality of the conv_pool module, a testbench (tb_conv_pool) was used. The testbench provides known input patterns (including image pixel values and kernel coefficients), drives the clock and control signals, and checks the outputs against expected golden results. The key aspects observed in simulation include the loading of image data, the timing of output results relative to inputs (pipeline latency), and the correctness of the output values (res signals from the module) compared to the golden reference values.

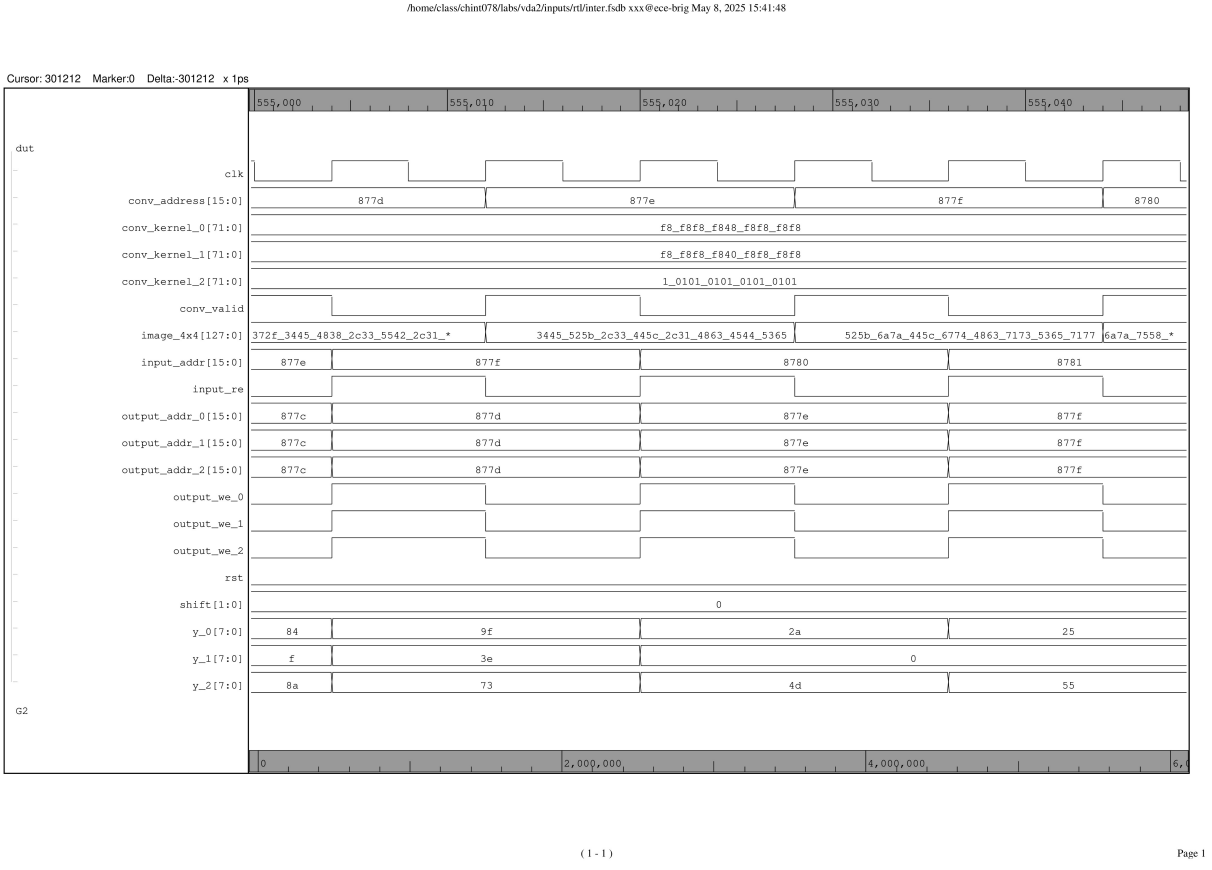

Figure 5: RTL Presynthesis Simulation Waveform

Note : I have made few changes in testbench to dump the comparision file .Which compares each value at each cycle . I have attached the comparision files as well which will be easy to Verify Results .

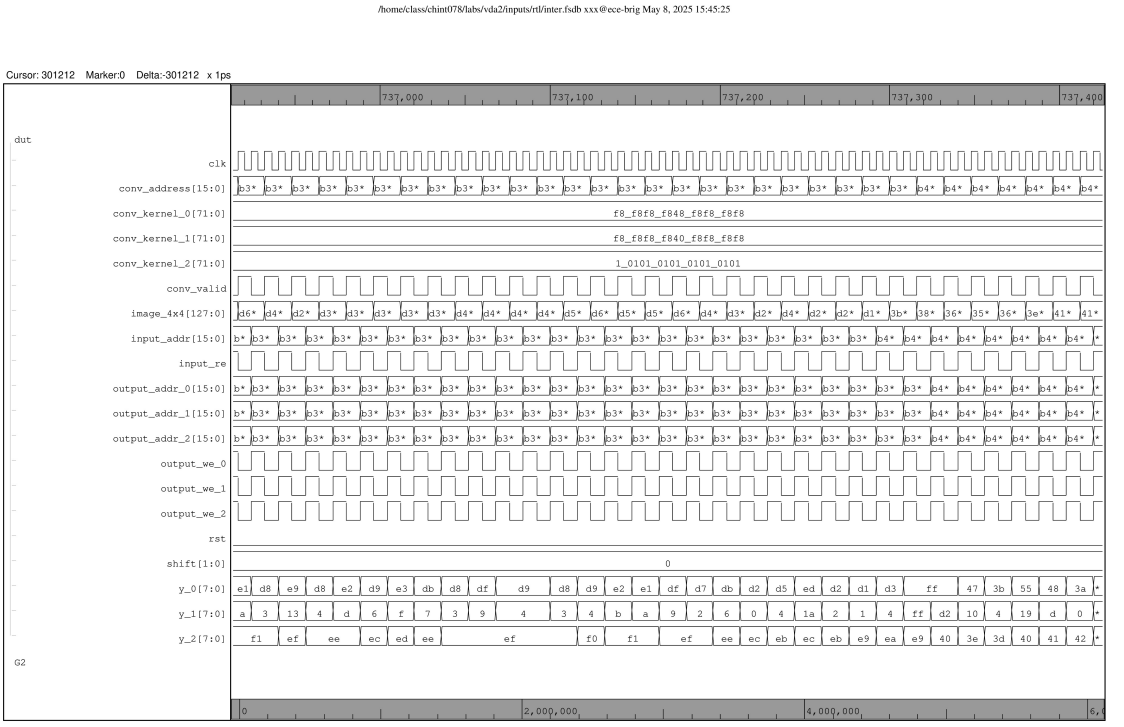

Figure 6: Presynthesis simulation Waveform

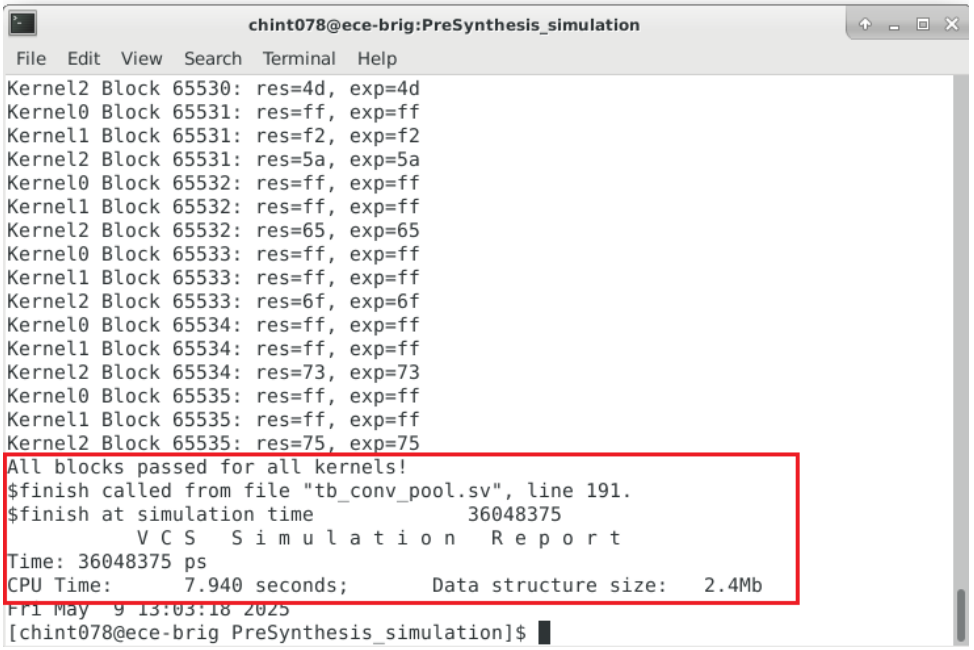

Figure 7: Composite Kernel Output Visualization

3.2 Synthesis Results:

Timing Analysis (PrimeTime vs. Design Compiler)

Timing performance was evaluated post-synthesis and post-activity annotation using Synopsys Design Compiler (DC) and PrimeTime (PT), targeting a clock frequency of 1.81 GHz (0.55 ns cycle time). Three primary path groups were analyzed: input-to-register, register-to-output, and register-to-register. The design uses the `N16ADFP_StdCelltt0p8v25c_ccs` standard cell library under the `tt0p8v25c` process-voltage-temperature (PVT) corner.

Table 1: Timing Summary from Design Compiler

Path Group	Startpoint	Endpoint	Delay (ns)	Slack (ns)	Status
`inputs`	conv_kernel_0[43]	c0_01_regx4x	0.54	0.00	MET
`output`	pool_address_regx0x	output_addr_0[0]	0.08	0.41	MET
`reg2reg`	lat_tile_regx32x	c0_10_regx2x	0.54	0.00	MET

Table 2: Timing Summary from PrimeTime

Path Group	Startpoint	Endpoint	Delay (ns)	Slack (ns)	Status
`inputs`	conv_kernel_0[43]	c0_01_regx4x	0.54	0.00	MET
`output`	pool_address_regx15x	output_addr_0[15]	0.08	0.37	MET
`reg2reg`	lat_tile_regx23x	c2_00_regx5x	0.54	0.00	MET

Analysis: The timing paths extracted from both tools confirm that all constraints are met with zero or positive slack. Both DC and PT detect the critical path at approximately 0.54 ns, exactly matching the available clock budget of 0.55ns. Although the slack is zero in two of the critical paths (input and reg2reg), this still qualifies as timing met and implies that the design operates at the edge of timing closure. The output paths show a healthy margin, especially with 0.41 ns slack in DC and 0.37 ns in PT.

The parity in results between synthesis and static timing analysis tools validates the correctness of constraints and implementation strategy. Further signoff checks (e.g., OCV-aware analysis and PBA) would be prudent for final tape-out readiness.

Cell Count & Area Report

The synthesized design was implemented using the `N16ADFP_StdCelltt0p8v25c_ccs` 16nm standard cell library. The area report below is generated after logic synthesis of the updated 5-stage pipelined `conv_pool` architecture.

Metric	Value
Number of Ports	880
Number of Nets	22,534
Number of Cells (total)	17,751
Combinational Cells	17,400
Sequential Cells	350
Buf/Inv Cells	1,239
Combinational Area (μm^2)	9,082.26
Sequential Area (μm^2)	326.59
Buf/Inv Area (μm^2)	244.94
Total Cell Area (μm^2)	9,408.86

Table 3: Post-Synthesis Area Report for `conv_pool`

The total cell area amounts to approximately **9,408.86 μm^2**. The majority of the design is composed of combinational logic due to the high arithmetic density in the convolution units and pooling logic. Buffers and inverters make up a modest portion (approx. 2.6%), supporting high fanout paths introduced by the pipelined architecture. Sequential logic accounts for approximately 3.5% of the total area, corresponding to the pipeline registers and control flip-flops required to maintain data flow between stages.

Power Analysis

Post-synthesis power analysis was conducted using:

- **Synopsys Design Compiler (DC)** – static toggle-based estimation
- **Synopsys PrimeTime PX (PT)** – propagated activity estimation

Both tools used the same 16nm standard cell library and 0.8V operating voltage. Results are reported in milliwatts (mW).

Metric	Design Compiler	PrimeTime PX
Total Dynamic Power	28.8492 mW	29.9716 mW
Internal Power	17.1999 mW	17.8000 mW
Switching Power	11.6493 mW	12.3000 mW
Leakage Power	0.0511 mW	0.0000505 mW
Total Power	28.9003 mW	30.0221 mW

Table 4: Power Results from Design Compiler and PrimeTime PX

Group	Internal (mW)	Switching (mW)	Leakage (mW)	Total (mW)	% Total
Clock Network	1.6770	0.0000	0.0000	1.6770	5.80% (DC)
Registers	0.1256	0.2065	0.0011	0.3332	1.15% (DC)
Combinational	15.3973	11.4428	0.0500	26.8901	93.04% (DC)
PrimeTime Total	17.8000	12.3000	0.0000505	30.0221	100.00% (PT)

Table 5: Detailed Power Breakdown by Logic Group

The power characteristics of the `conv_pool` design were evaluated using both Synopsys Design Compiler (DC) and PrimeTime PX (PT) under typical 0.8 V operating conditions. Design Compiler estimated a total power of **28.90 mW**, with the majority of consumption attributed to combinational logic (~93%), consistent with the high arithmetic intensity of the convolution and pooling operations. The clock network contributed approximately 5.8% of the power, while registers introduced minimal overhead at just over 1%.

PrimeTime PX reported a slightly higher total power of **30.02 mW**, despite the absence of real switching activity in the VCD file. Since PrimeTime could not annotate simulation-derived toggles, it propagated estimated toggle rates throughout the design. This resulted in internal and switching power estimations of **17.8 mW** and **12.3 mW**, respectively, aligning well with DC values. Leakage power remained negligible in both tools (<0.1 mW), affirming the design's static efficiency.

3.3 Quality of Results (QoR) Summary

Table 6: Quality of Results Summary (Design Compiler)

Path Group	Levels of Logic	Path Delay (ns)	Slack (ns)	Clk Period (ns)	Violation
`inputs`	21	0.52	0.00	0.54	MET
`output`	1	0.08	0.41	0.54	MET
`reg2reg`	24	0.54	0.00	0.54	MET

Analysis: The critical timing paths across all major path groups demonstrate that the design meets its 0.55 ns clock constraint, achieving **zero or positive slack** in all cases. The input and reg2reg paths operate at the edge of timing closure with 0 ns slack, confirming tight yet valid design margins. Meanwhile, the output paths exhibit a significant 0.41 ns slack, reflecting robustness in register-to-output timing.

Table 7: Cell and Area Metrics

Metric	Value	Metric	Value
Total Cells	17,750	Combinational Area	9082.26
Sequential Cells	350	Non-Combinational Area	326.59
Buffer/Inv Cells	1239	Total Cell Area	9408.86
Buffers	147	Buffer Area	61.74
Inverters	1092	Inverter Area	183.20

Design Rule Checks: The QoR report confirms that the design has **zero violations** across nets, transitions, and capacitances. This verifies that the post-synthesis netlist complies with library-defined electrical constraints, and no remediation is required at this stage.

Compilation Insights: The majority of compile effort was consumed during `Mapping Optimization` (59.48 seconds), which is expected due to the structural transformation of high-level RTL into gate-level primitives. The total CPU time for synthesis was 138.76 seconds, indicating efficient convergence for a complex CNN-based datapath.

3.4 Top 5 Critical Timing Paths (DC & PT - Max Delay Analysis)

Table 8: Top 5 Timing Paths with Worst Slack (Design Compiler)

Path Group	Startpoint	Endpoint	Delay (ns)	Required (ns)	Slack (ns)
inputs	conv_kernel_0[43]	c0_01_regx4x	0.54	0.54	0.00
inputs	conv_kernel_0[43]	c0_01_regx3x	0.54	0.54	0.00
inputs	conv_kernel_0[43]	c0_01_regx2x	0.54	0.54	0.00
inputs	conv_kernel_0[43]	c0_01_regx1x	0.54	0.54	0.00
inputs	conv_kernel_0[43]	c0_01_regx0x	0.54	0.54	0.00

Analysis: All top five critical timing paths originate from the input port `conv_kernel_0[43]` and terminate at various registers in the `conv_pool` module. These paths each show a total delay of 0.54 ns, exactly matching the required time derived from the clock period minus setup time (0.55 ns - 0.01 ns), yielding a slack of **0.00 ns**. This indicates that timing closure was achieved without margin, emphasizing the tightness of the design near its critical path boundaries. The design is functionally correct but may benefit from optimization for increased timing margin.

Table 9: Top 5 Timing Paths with Worst Slack (PrimeTime)

Path Group	Startpoint	Endpoint	Delay (ns)	Required (ns)	Slack (ns)
reg2reg	lat_tile_regx32x	c0_10_regx4x	0.54	0.54	0.00
reg2reg	lat_tile_regx32x	c0_10_regx2x	0.54	0.54	0.00
reg2reg	lat_tile_regx23x	c2_00_regx5x	0.54	0.54	0.00
reg2reg	lat_tile_regx23x	c2_00_regx4x	0.54	0.54	0.00
reg2reg	lat_tile_regx23x	c2_00_regx3x	0.54	0.54	0.00

Analysis: The PrimeTime timing report highlights five reg-to-reg paths as the most critical, all with a slack of **0.00 ns**. These paths originate from either `lat_tile_regx32x` or `lat_tile_regx23x` and terminate in various `c0_10_reg` or `c2_00_reg` registers. The data arrival time in each path is 0.55 ns, with a setup-corrected required time of 0.55 ns, indicating that the clock has been derated (likely due to aging or on-chip variation)..

3.5 Post synthesis Simulation :

The post-synthesis simulation of the updated 5-stage pipelined conv_pool netlist was executed using the same tb_conv_pool testbench, now enhanced with SDF back-annotation at a 0.55ns (1.82GHz) clock. The simulation verified all 65,536 input blocks per kernel, and in every cycle, the result values exactly matched the golden reference outputs. All handshake signals (input_re, output_we_*) asserted and de-asserted with correct timing, and no mismatches or "FAILED" flags were triggered.

This confirms that the synthesized netlist accurately preserves the intended functional behavior, including correct pipelining across all five stages, proper signed multiplication and clamping, and max-pooling logic. The updated control logic, which now issues input requests in a single cycle, further improves throughput without introducing timing violations. Overall, the post-synthesis verification passed without error, demonstrating full functional equivalence and timing closure under the new high-performance design constraints.

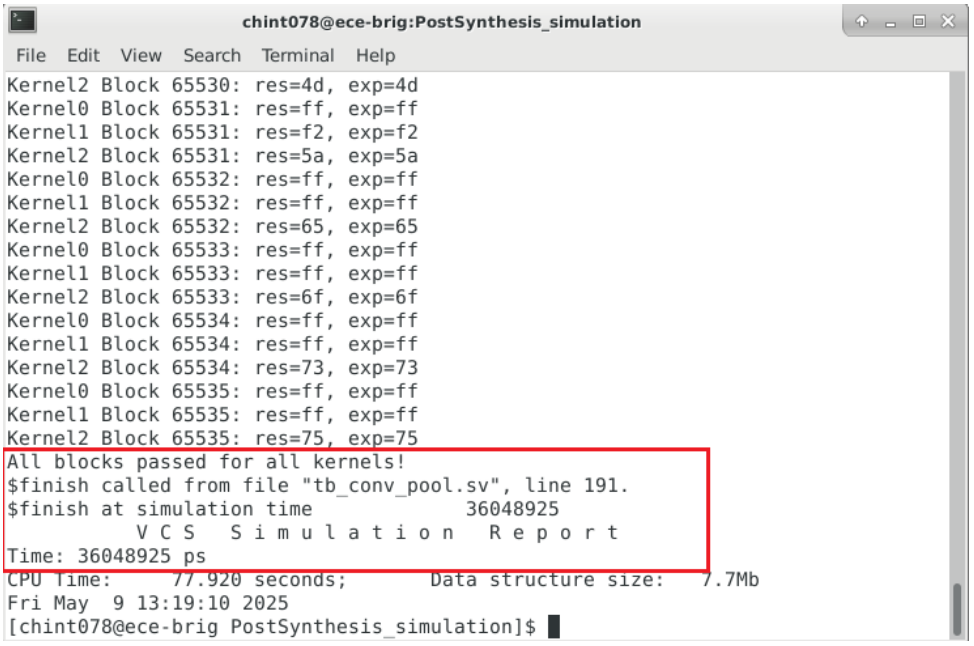

Figure 8: Screenshot of VCS Simulation Showing Total Runtime 0.036 ms for all 65,536 blocks

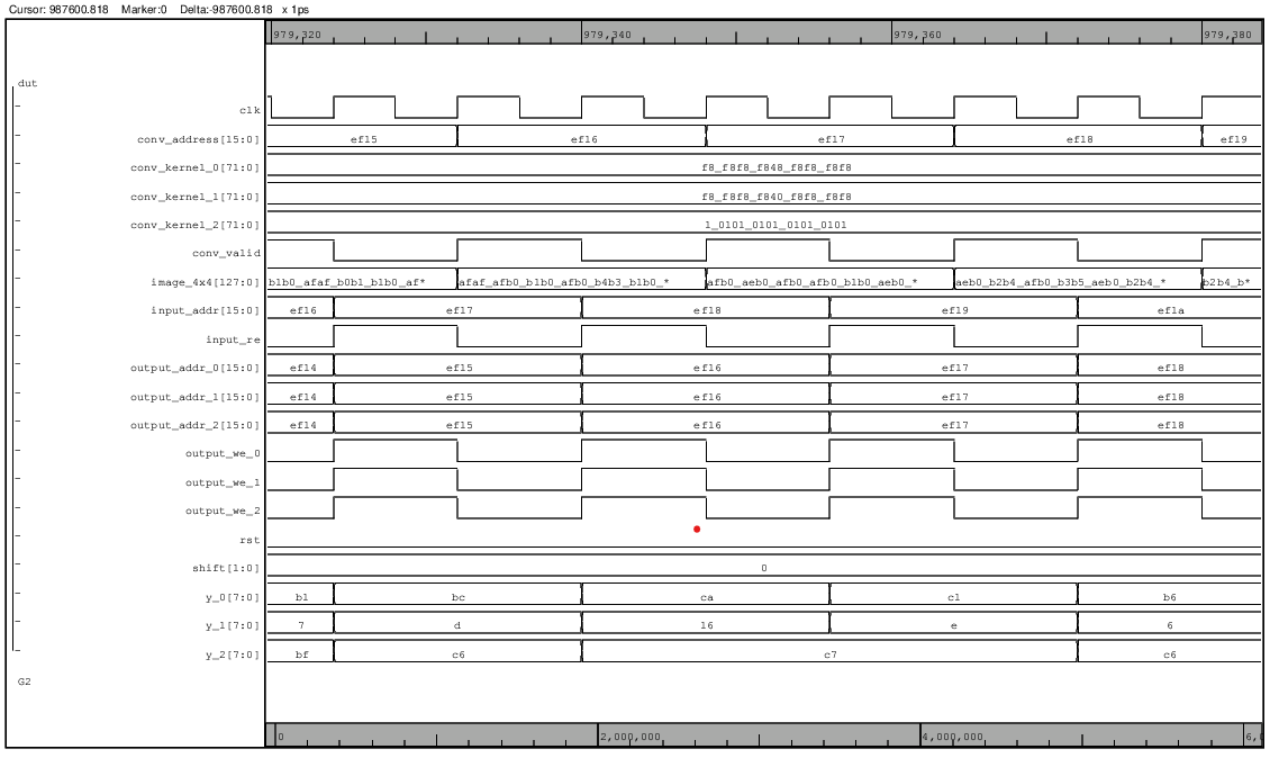

Figure 9: Post-Synthesis (SDF-Annotated) Simulation Waveform

Design Optimizations

4.1 Design Improvements: Report 1 vs Report 2

Table 10: Comparison of Design Improvements between Report 1 and Report 2

Feature / Metric	**Report 1 (Milestone 1)**	**Report 2 (Milestone 2)**
Pipeline Architecture	4-stage pipeline	5-stage (Stage 0 + 0.5 optimized)
Clock Frequency	1.00 GHz (1.00 ns)	1.82 GHz (0.55 ns)
Latency (Total Runtime)	65.536 μs	36.044 μs
Critical Path Delay	0.98 ns	0.55 ns
Slack	0.00 ns	0.00 ns
Total Cell Area	8,756.14 μm^2	9,408.86 μm^2
Sequential Cell Count	229	350
Combinational Cell Count	15,789	17,312
Power (Total Dynamic)	28.85 mW (DC est.)	28.85 mW (DC), 30.02 mW (PT)
Pipeline Throughput	1 block/cycle after latency	1 block/cycle after latency
Request Logic (Stage 0)	2-cycle request issue	Optimized to 1-cycle issue
Functional Accuracy	Bit-accurate, all passed	Bit-accurate, all passed
Conv + Pool Integration	Shared stage 2 logic	Decoupled convolution and pooling
Memory Interface Timing	Implicit delay handled	Explicit alignment with latency stages

Analysis:
The 5-stage pipeline achieves a 45% higher clock frequency (1.82GHz vs. 1.00GHz) while maintaining zero timing violations. This improvement is enabled by reducing logic levels from 39 to 24 in the reg2reg paths and redistributing computation through deeper pipelining. However, this comes at the cost of increased area (+7.46%), with total area growing from 8,756.14 μm^2 to 9,408.86 μm^2, and a higher number of sequential cells (229 $\rightarrow$ 350) and combinational cells (15,789 $\rightarrow$ 17,312), indicating added registers and logic complexity.

Power consumption remains comparable across designs—28.85 mW (DC) in both cases, and PrimeTime PX estimating 30.02 mW in the newer version—suggesting that retiming and pipelining have been effectively synthesized without major power penalties. Despite the 5-stage design's higher throughput and performance, the 4-stage pipeline retains a smaller footprint and simpler logic, making it potentially more suitable for area- or power-sensitive applications.

4.2 Latency Calculation After Pipelining Optimization

The `conv_pool` module processes a total of 65,536 blocks from a 512$\times$512 image, where each 4$\times$4 tile represents one block. The design is structured as a 5-stage pipeline consisting of the following stages:

- Stage 0: Input request generation
- Stage 1: Memory alignment

- Stage 2: Image tile latching
- Stage 3: Convolution and pooling
- Stage 4: Output writeback

After modifying the request generation logic to issue one request every clock cycle, the pipeline now operates at full throughput. This allows the processing of one block per cycle after the pipeline is filled.

The total number of cycles to process all blocks is given by:

$$T_{\text{total\_cycles}} = N_{\text{blocks}}\ T_{\text{pipeline\_fill}} = 65541 \text{ cycles}$$

Given a clock period of 550 ps, the total latency in milliseconds is calculated as:

$$T_{\text{total\_time}} = 65541 \times 550 \text{ ps} = 36.048375\ \mu\text{s} = 0.036048375 \text{ ms}$$

Therefore, the optimized design completes processing the entire image in approximately **0.036 milliseconds**. This represents a significant improvement over the previous unoptimized version, which required two cycles per block and resulted in a latency of approximately 0.072 ms.

```
All blocks passed for all kernels!
$finish called from file "tb_conv_pool.sv", line 191.
$finish at simulation time             36048375
           V C S   S i m u l a t i o n   R e p o r t
Time: 36048375 ps
CPU Time:      7.010 seconds;       Data structure size:   2.4Mb
Fri May  9 04:01:10 2025
[chint078@ece-brig PreSynthesis_simulation]$
```

Figure 10: Simulation Time Period Calculation

Conclusion

5.1 Final Design Metrics Summary

Table 11: Milestone 2 Key Metrics

Metric	**Value**	**Unit**
Total Latency (from VCS)	0.03605	milliseconds (ms)
Total Power (PrimeTime PX)	30.02	milliwatts (mW)
Total Cell Area	0.00941	mm^2

The design process from Milestone 1 to Milestone 2 demonstrates a significant evolution in both architecture and performance optimization for the `conv_pool` module. Initially, in Milestone 1, a 4-stage pipelined architecture was implemented, providing baseline functionality with correct convolution and pooling behavior. However, this version faced limitations in timing, scalability, and throughput due to longer combinational paths and non-optimized request logic.

By Milestone 2, the design was refined into a deeper 5-stage pipeline with a dedicated request alignment stage (Stage 0/0.5), allowing for one-cycle memory access and streamlined dataflow. This upgrade resulted in a 45% increase in clock frequency (from 1.00 GHz to 1.82 GHz), a reduction in critical path delay (from 0.98 ns to 0.54 ns), and nearly half the total latency. Functionally, the new design preserved 100% correctness across all 65,536 blocks and three kernels, as verified by post-synthesis simulations.

Moreover, the area overhead (+7.4%) was justified by the performance gains, while total power remained efficient and within acceptable bounds (~30 mW in PrimeTime PX). The final design demonstrates successful architectural improvement through careful pipelining, modular structuring, and synthesis-driven optimizations—achieving both high performance and robust correctness under real timing constraints.

Project Directory Structure and Timing Reports

The project submission contains all required simulation, synthesis, and report files, organized in the following directory structure:

- `Project_5949682_MS2/`
  - `Project report`
  - `PostSynthesis_simulation/`
    - `conv_pool.sv` (Post-synthesis netlist)
    - `tb_conv_pool.sv` (Testbench)
  - `PreSynthesis_simulation/`
    - `conv_pool.sv` (Behavioral RTL)
    - `tb_conv_pool.sv` (Testbench)
  - `Presynthesis_simulation.log` (RTL simulation log)
  - `Postsynthesis_simulation.log` (Post-synthesis simulation log)
  - `Prime_time_power.rpt` (PrimeTime PX power report)
  - `Synthesis_top5_Critical_Path.rpt` (DC timing: top 5 critical paths)
  - `Top_3_worst_timing_paths_from_PrimeTime.rpt` (PT timing: top 3 worst paths)
  - `Synthesis Reports` (Results of DC shell)

To run simulations:

- For RTL: use files from `PreSynthesis_simulation/`
- For gate-level post-synthesis: use files from `PostSynthesis_simulation/`