Report

	V0	V1
Times(s)	28.8741	0.0729225
GFLOPS	0.0569426	22.5468
LUT	8947	25716
LUTAsMem	695	728
REG	9378	39736
BRAM	79	79
URAM	407	467
DSP	7	322
Freq(MHz)	200	200
WNS(ns)	0.035	0.057

Performance Analysis

From cnn.h we get,

$$kNum = 64$$

$$kKernel = 4$$

$$kInImSize = 116$$

$$kOutImSize = 112$$

Total Flops can be calculated by,

$$\text{FLOPS} = kOutImSize \times kOutImSize \times kNum \times kNum \times (kKernel \times kKernel)$$

Therefore,

$$FLOPS = 882,083,584$$

If II=1, it means that, under ideal conditions, each operation in the innermost loop can be initiated every clock cycle. When running on 200MHz,

$$EstimatedCycle = (882,083,584)/(200,000,000) = 4.11s$$

V0

With no optimization at all, from v++.log, we know II=7. Therefore, the estimated running time is 7*4.11 = 28.77s.

V1

After changing the loop order of convolution into this,

```
    for (int h = 0; h < kOutImSize; ++h) {
        for(int j = 0; j < kNum; ++j){
            for (int p = 0; p < kKernel; ++p) {
                for (int q = 0; q < kKernel; ++q){
                    for (int w = 0; w < kOutImSize; ++w) {
#pragma HLS PIPELINE II=1
                        for (int i = 0; i < kNum; ++i) {
                            local_output[h][w][i] += local_input[h+p][w+q][j] *
local_weight[p][q][i][j];
                        }
                    }
                }
            }
        }
    }
```

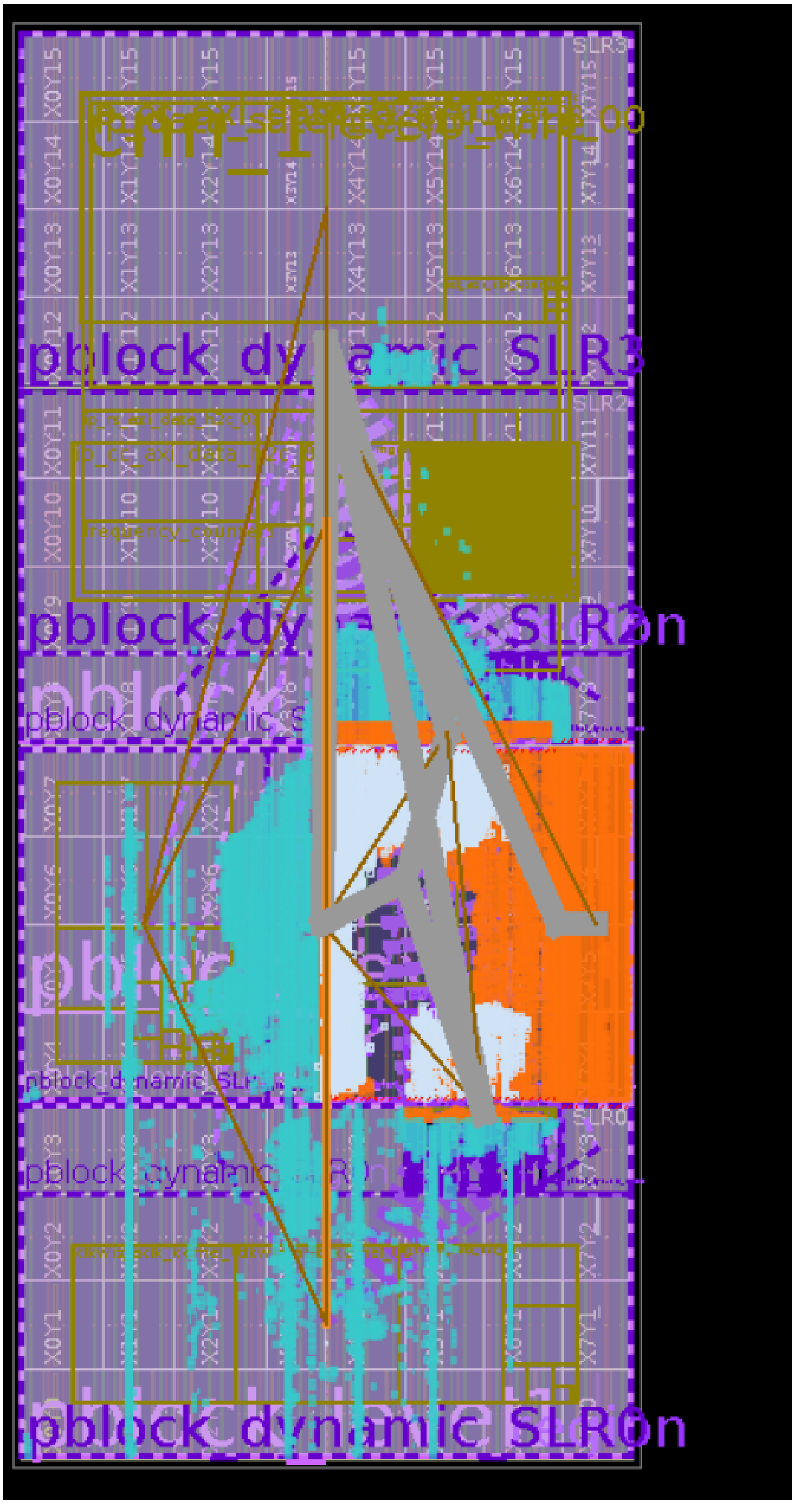
The estimated speed-up is 64x. Therefore, the estimated running time is 4.11/64 = 0.064s.

To make sure the II=1, the following optimization methods are used.

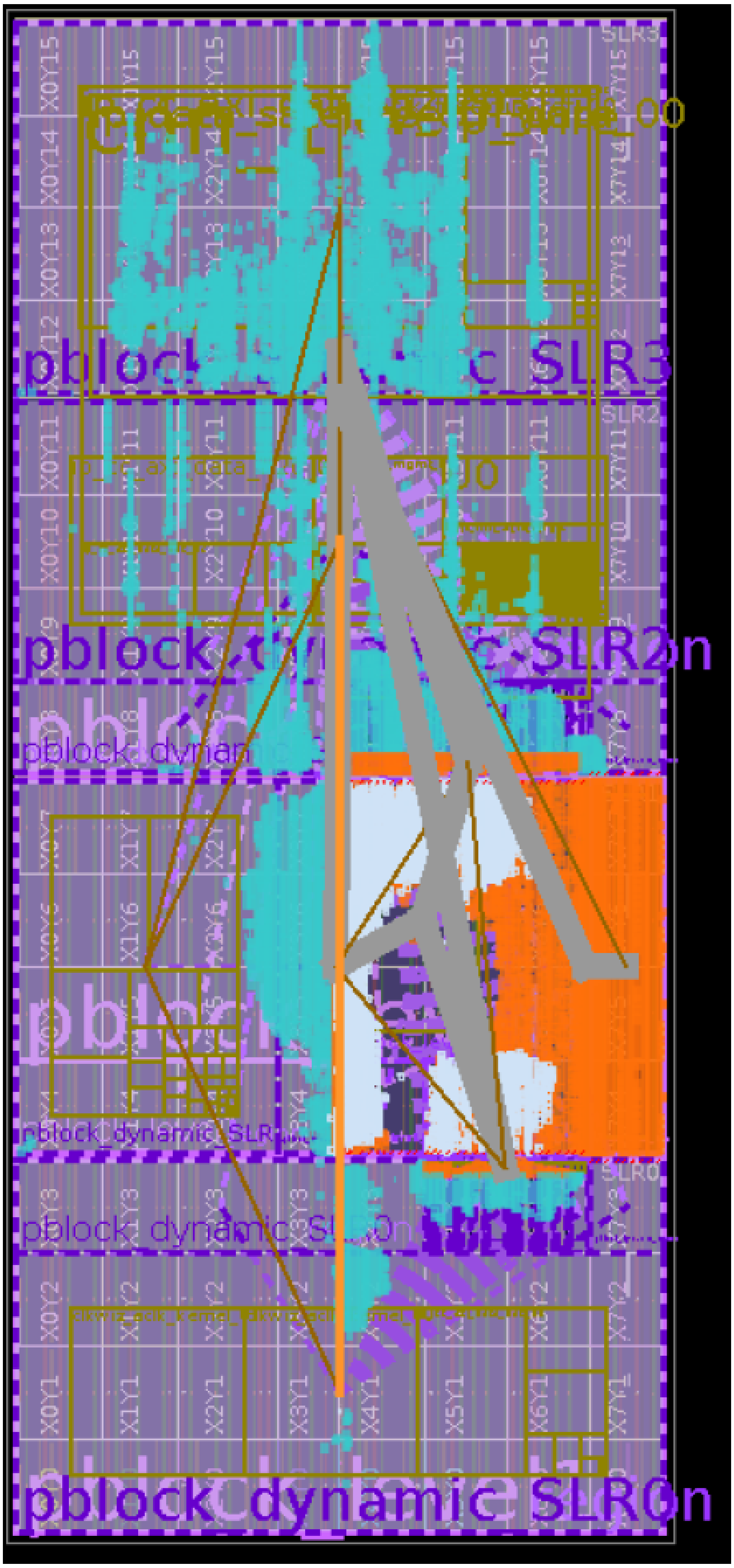
- Change `local_output` from `RAM_1P_URAM` to `RAM_2P_URAM`, i.e., dual-port URAM. This change is crucial because, in the innermost loop, there are read-modify-write operations on the same data unit. With dual-port memory, one port is used for reading and the other for writing, greatly reducing read-write conflicts and scheduling difficulty, thus increasing the likelihood of achieving **II=1**.
- Perform **complete partitioning** on the `i` dimension (channel dimension) of `local_output` (`#pragma HLS ARRAY_PARTITION complete dim=3`). This means that for every spatial position (h,w)(h, w), all elements in the `i` dimension of `local_output[h][w][i]` are split into independent storage units, enabling simultaneous access to different channels in the same cycle.
- Perform **complete partitioning** on the `i` dimension of `local_weight` (`#pragma HLS ARRAY_PARTITION variable=local_weight complete dim=3`). By completely partitioning the weights of the kernel across the channel dimension, multiple channel weight data can be accessed simultaneously in the innermost loop. This removes the limitation of a single memory port, accelerating access and reducing bandwidth constraints.
- Use the `#pragma HLS PIPELINE II=1` directive in the innermost loop to instruct the tool to initiate a new iteration every clock cycle, thereby achieving **instruction-level parallelism (ILP)**. By combining array partitioning and dual-port memory, the tool has the opportunity to complete multiple data accesses and computations within a single clock cycle, striving to achieve the performance target of **II=1**.

Chip Layout

v0


SLR3
pblock_dynamic_SLR3
SLR2
pblock_dynamic_SLR2n
frequency_counter
SLR0
pblock_dynamic_SLR0n
X0Y15
X7Y15
X0Y1
X7Y1

v1


SLR3
pblock_dynamic_SLR3
SLR2
pblock_dynamic_SLR2n
pblock_dynamic
pblock_dynamic_SLR
SLR0
pblock_dynamic_SLR0
pblock_dynamic_SLR0n

ENGN2911x: Reconfigurable Computing
Project CNN Accelerator Using U250 FPGA, DUE: 12/13/2024 8AM

Instructor: Peipei Zhou **Email: peipei_zhou@brown.edu**
TA: Jinming Zhuang **Email: jinming_zhuang@brown.edu**

1) Project Description:

In the project, please use the *project_src.zip* source code to build and optimize your convolutional neural network (CNN) accelerator.

1. One host code *host.cpp* and one HLS kernel *cnn.cpp* are provided.
2. Students are required to modify *cnn.cpp* in V1*.*
3. Each time you can construct your project following the directory structure below:

```
./cnn_v0/
├── Makefile
├── src
│   ├── cnn.cpp
│   ├── cnn.h
│   └── host.cpp
└── utils.mk
```

Every time after successful compilation and test please make sure following files are correctly saved for each version. Not all the files need to be submitted, some of them are used to complete your Lab report:

- Modified source code**: *host.cpp*, *cnn_v0.cpp***, and ***cnn_v1.cpp***.
- Host and Device executable files: ***hello_world*** and ***cnn.xclbin***
- Kernel resource report: ***_x.hw.xilinx_u250_gen3x16_xdma_4_1_202210_1/reports/link/imp/impl_1_kernel_util_routed.rpt***
- Timing report: ***_x.hw.xilinx_u250_gen3x16_xdma_4_1_202210_1/reports/link/imp/impl_1_hw_bb_locked_timing_summary_routed.rpt***

2) Assignment Requirement:

1. Version 0: Baseline convolutional neural network. Need to understand the program especially the data layout for the IFM, Weights and OFM, and the dataflow of CNN computation. (No modifications needed in *cnn.cpp*).
2. Version 1: Need to do code optimization and apply proper unroll and pipeline pragmas. (Mainly focus on computation optimization. Wider port and streaming DDR are not mandatory).
3. A summary report with details listed in the next section.

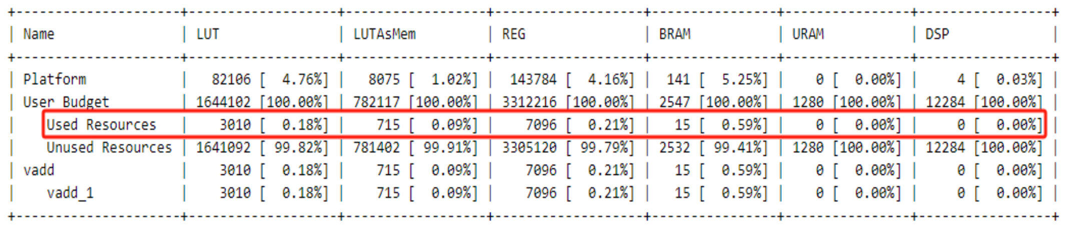
3) Submission Guide (total 100pts)

In your homework, please submit a project_**YourBrownID**.zip file with the following contents:
Please make sure each design runs with correct results on 200MHz FPGA design. (2x10=20pts)
v1 should run at least 300x faster than v0 (15 pts)

1. A “*report.pdf*” including a similar Table with the following information for all the four versions. **(20 pts)**

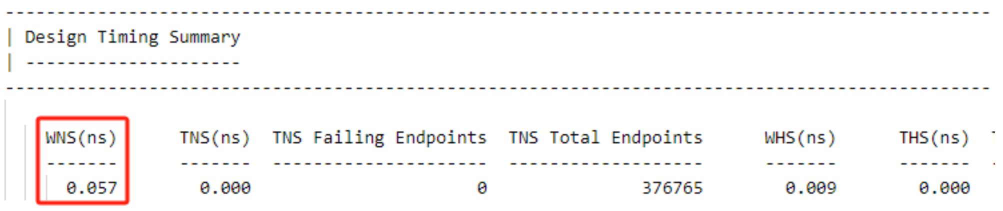
	V0	V1
Time(s)		
GFLOPS		
LUT		
LUTAsMem		
REG		
BRAM		
URAM		
DSP		
Freq (MHz)	200	200
WNS(ns)		

- Time: Kernel execution time in second measured by user timer APIs.
- Throughput in GFLOPS (GFLOPS = total giga floating point operations / total execution time). (2 x 2.5 = 5 pts)
- HLS kernel resources from following report: (2 x 5 = 10 pts) (_x.hw.xilinx_u250_gen3x16_xdma_4_1_202210_1/reports/link/imp/impl_1_kernel_util_routed.rpt):



```
+----------------------+------------------+------------------+-------------------+----------------+----------------+-----------------+
| Name                 | LUT              | LUTAsMem         | REG               | BRAM           | URAM           | DSP             |
+----------------------+------------------+------------------+-------------------+----------------+----------------+-----------------+
| Platform             |  82106 [  4.76%] |  8075 [  1.02%]  |  143784 [  4.16%] |  141 [  5.25%] |    0 [  0.00%] |     4 [  0.03%] |
| User Budget          | 1644102 [100.00%] | 782117 [100.00%] | 3312216 [100.00%] | 2547 [100.00%] | 1280 [100.00%] | 12284 [100.00%] |
|   Used Resources     |   3010 [  0.18%] |   715 [  0.09%]  |    7096 [  0.21%] |   15 [  0.59%] |    0 [  0.00%] |     0 [  0.00%] |
|   Unused Resources   | 1641092 [ 99.82%] | 781402 [ 99.91%] | 3305120 [ 99.79%] | 2532 [ 99.41%] | 1280 [100.00%] | 12284 [100.00%] |
| vadd                 |   3010 [  0.18%] |   715 [  0.09%]  |    7096 [  0.21%] |   15 [  0.59%] |    0 [  0.00%] |     0 [  0.00%] |
|   vadd_1             |   3010 [  0.18%] |   715 [  0.09%]  |    7096 [  0.21%] |   15 [  0.59%] |    0 [  0.00%] |     0 [  0.00%] |
+----------------------+------------------+------------------+-------------------+----------------+----------------+-----------------+
```

- Timing information frequency and worst negative slack (WNS): (2 x 2.5 = 5 pts) (_x.hw.xilinx_u250_gen3x16_xdma_4_1_202210_1/reports/link/imp/impl_1_hw_bb_locked_timing_summary_routed.rpt)
  - WNS in ns (around line 140):

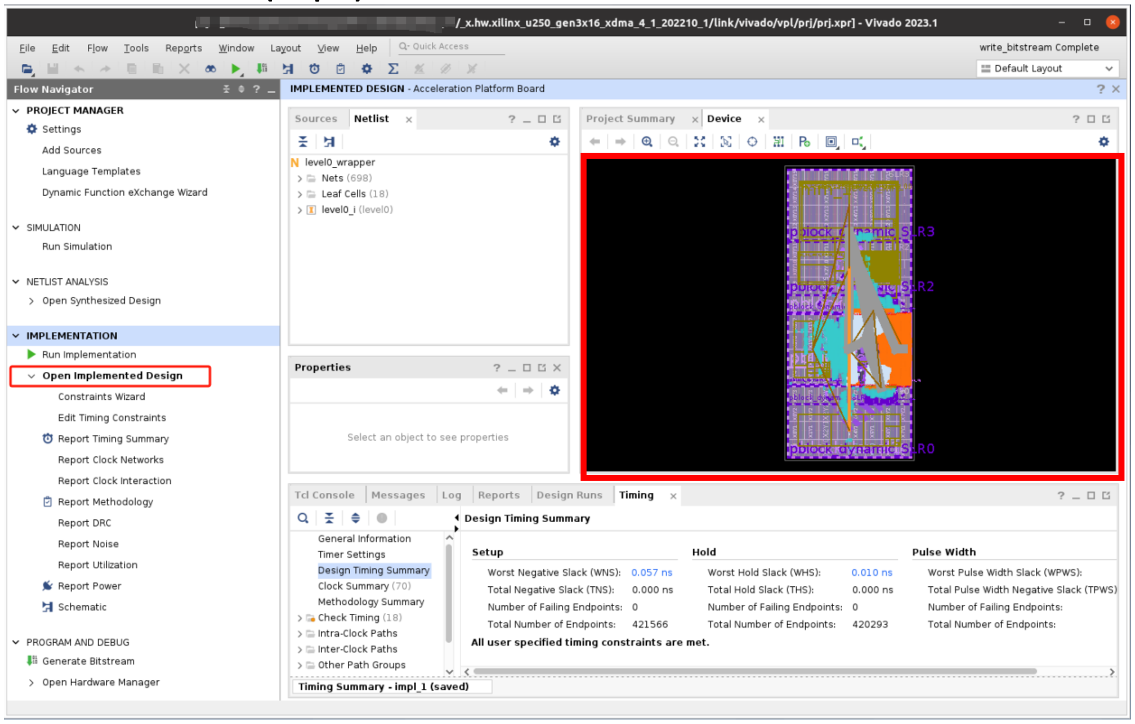


```
------------------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)
    -------      -------  ---------------------  -------------------      -------      -------
      0.057        0.000                      0               376765        0.009        0.000
```

2. **Report in report.pdf about the performance analytical model for your v0 and v1. Report all the optimizations you have applied. (25pts)**
   - Should include the estimated cycles for computation and communication.
   - Should be as close as the on-board execution time.
   - Should provide detailed analysis or reasons to explain the estimated result.

3. Screenshot on the chip layouts for designs v0-v1 and report them in the *report.pdf* following the guidelines in *Vivado_GUI.pdf*. Adjust the window size of the layout and make a screenshot. **(10 pts)**



Design Timing Summary

```
Setup                                        Hold                                         Pulse Width
  Worst Negative Slack (WNS):    0.057 ns     Worst Hold Slack (WHS):         0.010 ns     Worst Pulse Width Slack (WPWS):
  Total Negative Slack (TNS):    0.000 ns     Total Hold Slack (THS):         0.000 ns     Total Pulse Width Negative Slack (TPWS)
  Number of Failing Endpoints:   0            Number of Failing Endpoints:    0            Number of Failing Endpoints:
  Total Number of Endpoints:     421566       Total Number of Endpoints:      420293       Total Number of Endpoints:
All user specified timing constraints are met.
```

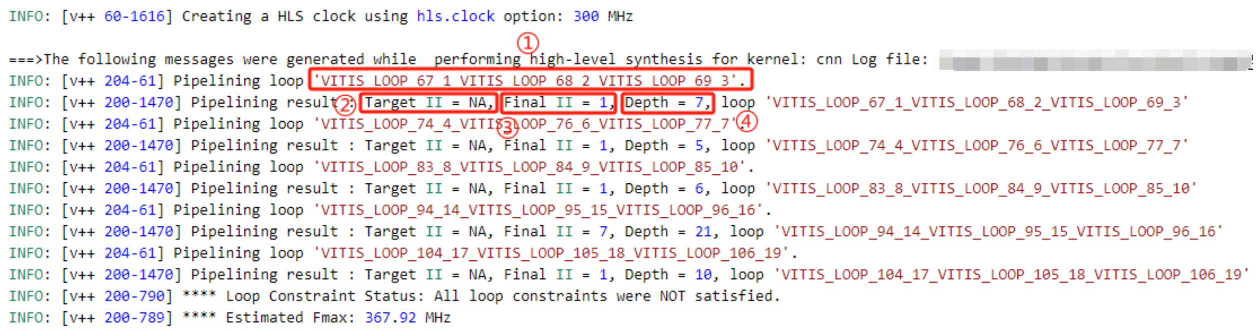
4. **Modified source code, device binary and report files** with the following directory structure. In the top level, there should be a "*report.pdf*" file and three folders (src, v0, and v1). In each folder, there should be the source code, binary files and generated reports shown below. **Please include these contents in a folder called project_YourBrownID and then create a zip file called project_YourBrownID.zip. The word "YourBrownID" should be replaced by students Brown ID starting with "140".** (**10 pts**: folder name & organization 5pts, all the files 5pts)

```
├── report.pdf
├── src
│   ├── cnn_v0.cpp
│   └── cnn_v1.cpp
├── v0
│   ├── cnn.xclbin
│   ├── impl_1_hw_bb_locked_timing_summary_routed.rpt
│   └── impl_1_kernel_util_routed.rpt
└── v1
    ├── cnn.xclbin
    ├── impl_1_hw_bb_locked_timing_summary_routed.rpt
    └── impl_1_kernel_util_routed.rpt
```

Q & A

1. How to estimate the execution time of the program?

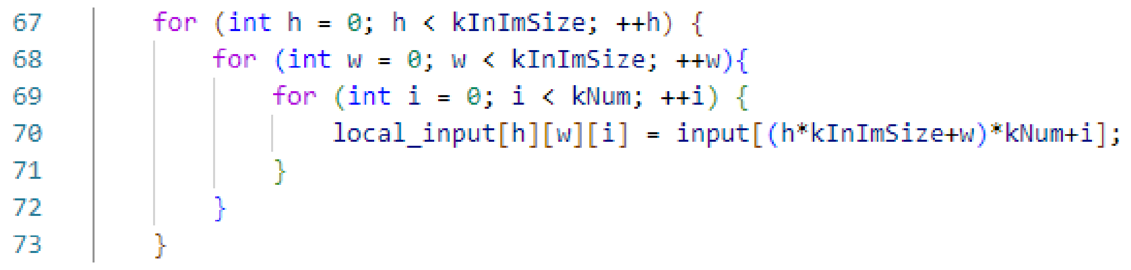
This is the first step to understand the bottleneck of the design and then optimization can be applied. For each code snippet with perfect nested loop, there will be an estimated scheduling report after C → RTL compilation. Usually the report will be generated within 10 mins and will be saved in “ *_x.hw.xilinx_u250_gen3x16_xdma_4_1_202210_1/logs/cnn/v++.log”. The report for Version 0 is shown below:*



```
INFO: [v++ 60-1616] Creating a HLS clock using hls.clock option: 300 MHz

===>The following messages were generated while  performing high-level synthesis for kernel: cnn Log file: [redacted]
INFO: [v++ 204-61] Pipelining loop 'VITIS_LOOP_67_1_VITIS_LOOP_68_2_VITIS_LOOP_69_3'.
INFO: [v++ 200-1470] Pipelining result : Target II = NA, Final II = 1, Depth = 7, loop 'VITIS_LOOP_67_1_VITIS_LOOP_68_2_VITIS_LOOP_69_3'
INFO: [v++ 204-61] Pipelining loop 'VITIS_LOOP_74_4_VITIS_LOOP_76_6_VITIS_LOOP_77_7'.
INFO: [v++ 200-1470] Pipelining result : Target II = NA, Final II = 1, Depth = 5, loop 'VITIS_LOOP_74_4_VITIS_LOOP_76_6_VITIS_LOOP_77_7'
INFO: [v++ 204-61] Pipelining loop 'VITIS_LOOP_83_8_VITIS_LOOP_84_9_VITIS_LOOP_85_10'.
INFO: [v++ 200-1470] Pipelining result : Target II = NA, Final II = 1, Depth = 6, loop 'VITIS_LOOP_83_8_VITIS_LOOP_84_9_VITIS_LOOP_85_10'
INFO: [v++ 204-61] Pipelining loop 'VITIS_LOOP_94_14_VITIS_LOOP_95_15_VITIS_LOOP_96_16'.
INFO: [v++ 200-1470] Pipelining result : Target II = NA, Final II = 7, Depth = 21, loop 'VITIS_LOOP_94_14_VITIS_LOOP_95_15_VITIS_LOOP_96_16'
INFO: [v++ 204-61] Pipelining loop 'VITIS_LOOP_104_17_VITIS_LOOP_105_18_VITIS_LOOP_106_19'.
INFO: [v++ 200-1470] Pipelining result : Target II = NA, Final II = 1, Depth = 10, loop 'VITIS_LOOP_104_17_VITIS_LOOP_105_18_VITIS_LOOP_106_19'
INFO: [v++ 200-790] **** Loop Constraint Status: All loop constraints were NOT satisfied.
INFO: [v++ 200-789] **** Estimated Fmax: 367.92 MHz
```

- The location of loops that are scheduled is shown in ① which are loops 67-69.



```
67  for (int h = 0; h < kInImSize; ++h) {
68      for (int w = 0; w < kInImSize; ++w){
69          for (int i = 0; i < kNum; ++i) {
70              local_input[h][w][i] = input[(h*kInImSize+w)*kNum+i];
71          }
72      }
73  }
```

- Without explicitly specifying the *pipeline II pragma (*in ② the *TARGET II = NA*), Vitis HLS will try to set the II of the innermost loop (Line 69) to 1.

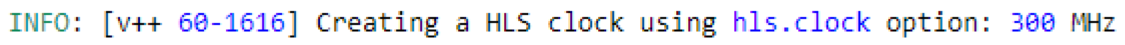
- ③ and ④ are the key factors to estimate the execution cycle of Line 67-73. Setting Line 69 with *pipeline II = 1* means, the programmer wants the compiler to create a schedule that makes the number of cycles to process one iteration of this loop to 1 cycle. The final II is the achieved number of cycles after a latency (Depth). **So, the total estimated cycle for this code snippet can be calculated by: (total number of iterations) * (Finall_II) + (Depth - 1).**

- This cycle execution method can also be applied for other code snippets in Version 0. After aggregating them together and convert it to second (each cycle is 5ns under 200MHz), you should get similar execution time compared with the real on-board one. And you could find the bottleneck of the design.

2. Why is Final II of loop 96 equal to 7?

This is because it is a reduction loop that the data indexed by this loop will be added together. More specifically, if we want to make *pipeline II* of this loop to be 1, it means the result of last loop should be calculated in one cycle to allow the next iteration to add it back. However, this is not achievable in one cycle due to the complexity of MAC operation. Please provide solutions to avoid this. If Final II could be 1, 7x speedup is achieved.

3. Why does the compilation stall in the stage below:



```
INFO: [v++ 60-1616] Creating a HLS clock using hls.clock option: 300 MHz
```

For our final project, the compilation time in this stage should be less than 10 minutes. If this is far more than 10 minutes. It’s most likely that the partitioning strategy applied in the design is too complicated and won’t be achievable.

4. Unroll pragma cannot be applied to the statement that involves off-chip access.

5. How to make sure the frequency of the design is 200 MHz?

There are two frequency settings in the *Makefile*. The final implementation frequency is determined by Line 48 of *Makefile*. The frequency of HLS in Line 94, is to guide the C -> RTL compilation. To provide enough budget for final implementation, the HLS frequency is set higher than 200MHz. **In conclusion, keeping *Makefile* as what it is will guarantee that the final frequency is 200MHz.**