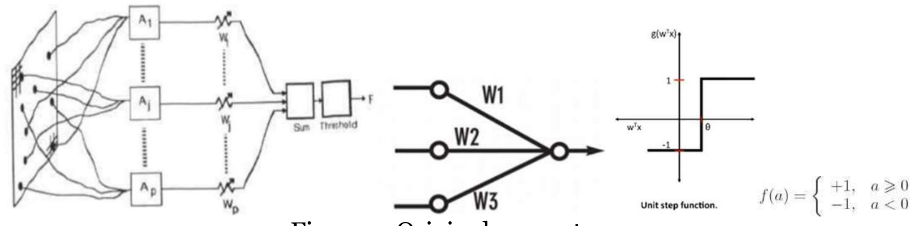
Autumn HT 2020

HT2020: IL2230 Hardware Architectures for Deep Learning

Lab 1

Hardware Design, Implementation and Evaluation of Artificial Neuron

Background



A1
Aj
Ap
W1
Wj
Wp
Sum
Threshold
F
W1
W2
W3
g(wᵀx)
1
-1
wᵀx
0
Unit step function.
$f(a) = \begin{cases} +1, & a \geqslant 0 \\ -1, & a < 0 \end{cases}$

Figure 1. Original perceptron

An artificial neuron is a conceptual model of a biological neuron. Figure 1 shows the original neuron model called perceptron, which uses the step function to achieve non-linear functionality.
The basic neuron model is mathematically a weighted sum of inputs applied to a nonlinear transformation, as shown in Figure 2. Here the nonlinear function f can be in different forms such as the step function, sigmoid, hyperbolic tangent, ReLU etc. Neuron is the building block of neural networks. It is the "brick" of the deep

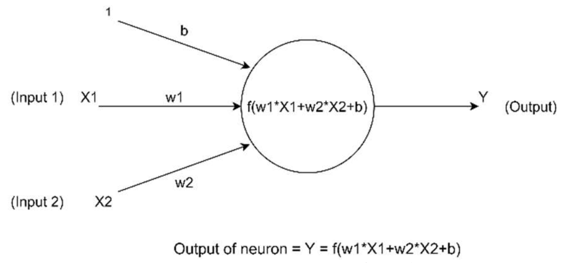


Figure 2. A two-input artificial neuron

learning "house". Therefore, it is essential to understand the hardware design organizations and their tradeoffs of neuron.

Task

The project intends to design, implement, and evaluate an N-input neuron in hardware. Therefore, a hardware model of the artificial neuron is to be built in VHDL or Verilog and synthesized to obtain its area, frequency, and power profiles. Looking into the functions of the neuron, we can find that the core computation is a repetitive application of Multiplier–ACcumulator (MAC) operation, shown in Figure 3. Therefore, one can explore the digital hardware organization principle with respect to serialization and parallelization. In fact, the project shall be carried out with three designs:

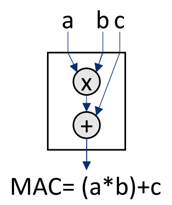


Figure 3. MAC unit

1. Generic N-input neuron modeling with one MAC unit (N is generic). This means that the N- input neuron shall be modeled with a single MAC unit, and the control path takes care of the repetitive MAC operations for N times. This is one extreme case of fully serial architecture.

2. Generic N-input neuron modeling with N-MAC units (N is generic). This is the other extreme that uses a fully parallel architecture (as many MAC units as needed).

3. Generic N-input neuron modeling with K-MAC units (N and K is generic, and K<N). This is a semi-parallel solution that the control path takes care of the repetitive MAC operations over N/K times and sums the intermediate results in the end. In this lab, we set K=2.

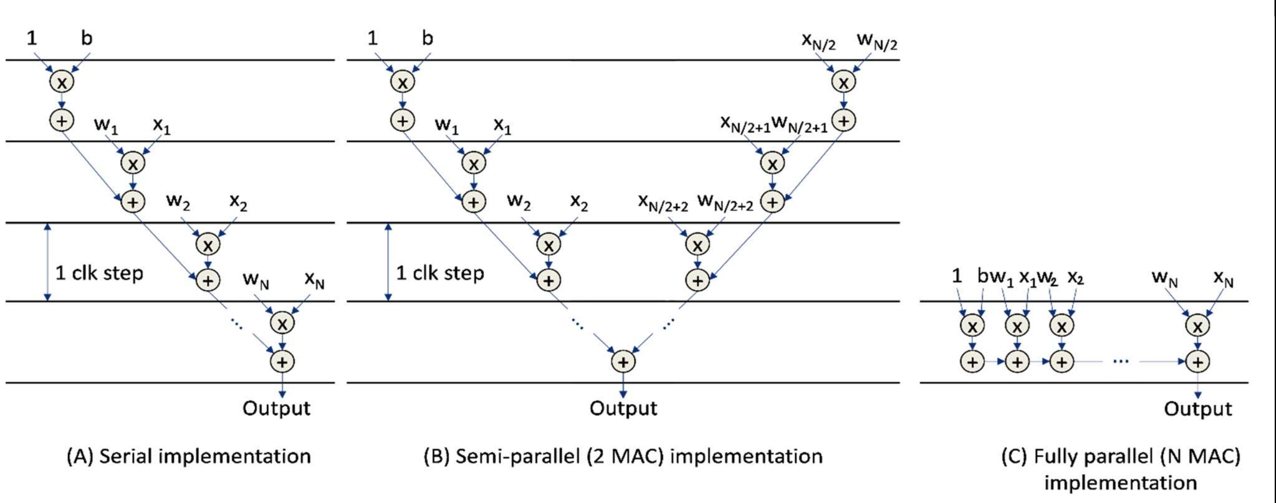


Figure 4. Schedule of the different implementations.

The schedule of each of these solutions is illustrated in Figure 4.
Additional constraints or considerations for three designs (models):

- Besides the digital organization, the project shall also investigate the impact of data precision (weight, input, output) on hardware complexity. For hardware efficiency, the input data, weight, and output data use 32-bit, 16-bit and 8-bit fixed point numbers.
  For 32-bit/16-bit/8-bit fixed point number, use 12/6/3 bits for the integer part and 20/10/5 bits for the fractional part, respectively.
  Note that the intermediate partial sum may use as many bits as needed.

- The nonlinear function to be considered is the step function, the sigmoid function and the ReLU function. For the sigmoid function, one shall use an approximation method, for example, piece-wise linear approximation or look-up table.
  Implement at least one (any one) of the three non-linear functions, preferably the ReLU function.

The project shall make a comparative evaluation of area, frequency and power for the following:

- The one-MAC design, K-MAC design, and N-MAC design for an N-input neuron under the same data precision (32-bit, 16-bit or 8-bit data precision) with the same nonlinear function.
- Consider N varying from 2 to 128 for the N-input neuron. Set N = 2, 4, 8, 16, 32, 64, 128.

Write separate RTL modules for the MAC unit and for the nonlinear function. Instantiate the MAC and nonlinear units as many times as needed to realize your neuron.

Deliverables

1. Source code and test benches verifying the correct functionality of your designs
2. Hardware model simulation results and synthesis results
3. Technical Report

The project shall result in a technical report with the design, implementation, and evaluation details. More importantly, proper conclusions shall be drawn from the comparative evaluations under different data precisions, and discussions for the design scalability (e.g. when N varies from 2 to 128) be given. In addition, you can write down the lessons learnt from the project.

Team size: 4 – 5 students

For IL2225 Embedded Hardware Design in ASIC and FPGA

LAB 1: Logic Synthesis and Design Space Exploration

For this part of the lab you should split your team in two sub teams, the same as the ones that you have in the IL2225 course.
Each student has to submit a report answering the following questions, highlighted with red. The report has to be submitted one day (24 hours) before the scheduled Lab-check.

Laboration Directory Structure

For Lab-1 and Lab-2 you should create the following directory structure. These directories will have the following purposes.

- **IL2225_LAB:** This will be the main directory which will have three more directories for LAB1 and LAB2 and global scripts. The global scripts directory will be used to store the scripts to instantiate different technology libraries.
- **LAB1/LAB2:** These directories will store the files for the respective laborations.
- **12_MAC, 1_MAC, 2_MAC:** These directories should be inside LAB1 and will be used for each of the designs
- **synopsys_dc.setup:** This file will contain the setup information for the design vision and will be discussed in the next section.
- **WORK:** This directory will contain the working files for design analyzer.
- **SOURCE:** Contains the design files for synthesis
- **SCRIPTS:** You can save your scripts in this directory
- **MAPPED:** The gate level netlist generated by the synthesis will be stored in this directory.
- **REPORTS:** All the generated reports will be saved in this directory.

Instructions and Questions

For this part of the lab we will work with three different designs:

1. One-MAC design with 12 inputs under 8-bit precision.
2. Two-MAC design with 12 inputs under 8-bit precision.
3. Fully parallel 12-MAC design with 12 inputs under 8-bit precision.

Draw: Draw a block diagram of each design.

Question 1: How many adders and multipliers are there in the critical path of each design?

Question 2: Compare the three architectures in terms of their critical path, throughput, and latency.

Definitions

Throughput is the rate of outputs per unit of time, i.e. how many outputs per unit of time are produced by your design.

Latency is the time required to perform one action in time units. For example the number of cycles that is required to produce a single output.

Design Vision Initialization

When using the Synopsys synthesis tool, there are startup files that are read by Synopsys. The name of this start up file is *".synopsys_dc.setup"*. A *".synopsys_dc.setup"* file is a script file that is executed automatically when design_vision or dc_shell is invoked. The setup file is useful for setting up variables and commands that you want to execute every time you invoke Design Vision. In this start up file four important parameters must be set before you can perform any synthesis.

- **search_path:** The parameter search_path is used to specify to the synthesis tool all the paths that it should search in when looking for a synthesis technology library to reference during synthesis. **target_library:** The file pointed to by the parameter target_library is the library that contains all the logic cells for mapping during synthesis.
- **symbol_library:** The parameter symbol_library points to the library that contains visual information on the logic cells in the synthesis technology library. All logic cells have a symbolic representation and

information about the symbols is stored in this library.

- **link_library:** The parameter link_library points to the library that contains information on the logic gates. Design Vision reads the setup file in the root of the home directory. Here the information that is general for all designs should be stored. A typical setup file located in the root of the home directory is:

```
set designer "Tom Cruise"

set company "KTH"

[getenv "SYN"]

set search_path "${SynopsysHome}/libraries/syn"
```

- Then the setup file in the current directory is read. Here design specific information should be stored, e.g. the file that specifies the target technology.

```
set target_library "tcbn90gtc.db"

set symbol_library "tcbn90g.sdb"

set link_path "$search_path"
```

The Design Compiler is invoked by typing dc_shell in the UNIX shell. The Design Vision is the graphical front-end version of DC and is launched by typing **design_vision.**

Design Vision Start-Up

1. Move to directory LAB1

```
Prompt>cd IL2225_LAB/LAB1
```

2. Invoke the design vision

```
/IL2225_LAB/LAB1/12_MAC > design_vision
```

3. Setup the path of the libraries by running the synopsys_dc.setup script.

```
Design_vision> source synopsys_dc.setup
```

4. Whenever you need help, you can access online documentation by clicking **Help/online help**.
5. The path of the library is:

/afs/it.kth.se/pkg/synopsys/extra_libraries/standard_cell/TSMC/tcbn90g_110a/Front_End/timing_power/tcbn90g_110a/

At this path you can find all the required libraries. So, whenever you need to find a specific library, look into this path.

6. If any of the commands given in this lab is not working, try to press tab while entering the command in the command window in design vision. You will see a quick help telling you the exact syntax of the command.
7. **You must know the reasons for the results you will be entering in the tables in rest of the lab. These reasons will help us judge if you have enough understanding to pass this lab.**

Logic Synthesis Steps

We have already mentioned the process of synthesis which is described as translation plus optimization plus mapping. In terms of the Synopsys tools, translation is performed by the *analyze* plus *elaborate* command sequence. Optimization and mapping are performed by the compile command. To run each of the synthesis steps, both GUI and scripts can be used. To automate the synthesis process, you can create a script containing all the required commands.

Design Entry

Before synthesis the design must be entered into DC in a standard format (VHDL, Verilog, EDIF, db etc). DC provides the following two methods of design entry: (a) *read* command (b) *analyze+elaborate* commands.

Read or Analyze

- The *read* command reads in files that are in another format than HDL, such as *dcc* and *pla*. Although it supports HDL format, it does not do the checking accomplished by analyze and elaborate.
- The *analyze* command reads a VHDL or Verilog file, checks for proper syntax and synthesizable logic, and stores the design in an intermediate format. Remember that you should read/analyze the files in order: 1- Packages, 2- All VHDL modules except top module, 3- Top level module.

```
analyze -format vhdl -lib WORK {"./SOURCE/<filename.vhd>"}
```

Elaborate

The elaborate command creates a design from the intermediate format produced by analyze. The elaborate command replaces the HDL operators in the design with synthetic operators and determines the correct bus sizes. The design is now expressed by generic technology independent components (GTECHlibrary).

- elaborate <TOP_DESIGN> ☐architecture <NAME> ☐library DEFAULT

Setting constraints

In this section we will learn to set constraints on our design. The constraints can be of following kinds:

Selecting Wire Load Models

The wireload model contains information that DC utilizes to estimate the interconnect delays during the pre-layout phase of the design. Several models appropriate to the different size of the logic are included in the technology library. These models define the capacitance, resistance, area factors. You can check all the available models by checking the tcbn90gtc.lib.

Question 7: Open this file in an editor and find the available wire_load models. List four of them in the following table. What is the difference between the wireload models that you have listed?

Table: Wireload model

Wire-load-model	Resistance	Capacitance	Slope

Select wireload mode by:

```
set_wire_load_mode (top/enclosed/segmented)
```

Select one of the wireload models using this command:

```
set_wire_load_model -name “name of wireload model"
```

Question 8: What do you understand by wireload mode?

Setting Operating Conditions

Set of operating conditions defined in the library specify the process, temperature, voltage and RC tree values.

These values are used during synthesis and timing analysis of the circuit.

Question 9: What is the default operating condition in tcbn90gtc.lib?

Explore the target directory for tcbn90gtc.lib and open another lib file. Read the operating conditions in that file and fill in the following table:
Table: Operating conditions

Library	Condition		Process	Temperature	Voltage	tree_type
tcbn90gtc.lib	NCCOM		1	25 C	1.0 V	balanced_tree

Set the operating condition of the design using the following command.

- `set_operating_conditions ” OPERATINGCONDITION”`

Question 10: Why do we need to specify the operating condition?

Modeling Clock

The design can be constrained by a clock. A clock can be created by the following command.

- `create_clock -name "clk" -period 5 -waveform { 0 2.5 } { clk }`

False Path

The reset pin doesn’t contribute to the signal path and can produce false timing analysis results. It’s important to ignore the timing path originating from reset pins. For that reason we need to use the following command.

- `set_false_path -from [get_port rst_n]`

Synthesis and Analysis

Mapping

The technology independent design is mapped to technology dependent library cells using the command:

- `compile -map_effort medium`

After the compile is completed check the output in the command window of design vision. There you can see how the synthesis process works and many memory elements are used and their type (flops or latches). Make sure that your design is latch free, i.e. there are no latches, and all memory elements are implemented as flip-flops.

Save Design

For saving the design you can use this command:

```
write -hierarchy -format ddc -output $HOME/IL2225_LABS/LAB1/12_MAC.ddc
```

After saving a design, you can load it using *Read* command.
For saving the design in the Verilog netlist format use this command:

```
write -hierarchy -format verilog -output $HOME/IL2225_LABS/LAB1/12_MAC.v
```

Generate Reports

Now that the design has been mapped to standard cells, it is the time to generate reports. You need to generate the following reports. The ">>" operator appends all the reports in one file.

1. Constraints

   - `report_constraints > ./REPORTS/constraint_12_MAC.rep`

2. Cells

   - `report_cell > ./REPORTS/cell_12_MAC.rep`

3. Area

   - `report_area > ./REPORTS/ area_12_MAC.rep`

4. Timing

   - `report_timing > ./REPORTS/ timing_12_MAC.rep`

5. Power

   - `report_power -analysis_effort low > ./REPORTS/ power_12_MAC.rep`

Experiments:

In this part, you have to synthesize all three discussed neuron architectures, i.e., 1-MAC, 2-MAC, and 12-MAC. You can create a synthesis script for each design by writing all commands in a file *myScript.tcl* to automate the synthesis process and execute it using:

```
File -> Execute Scripts
```

Fully Parallel Neuron:

Do the following experiments for all 3 designs located in:

1. /IL2225_LABS/LAB1/12_MAC
2. /1_MAC
3. /2_MAC

Start-up the *Design Vision* form each of the 1- 3 design folders.
Analyze all of the files located in the SOURCE folder and *Elaborate* the top level module.

Question 11: How many memory elements (latches and flip-flops) are there in the design?

Architecture	12_MAC	2_MAC	1_MAC
Memory elements			

Hint! You can see the number of flops and latches used in the command line during the synthesis process.
In the logical hierarchal window of design vision GUI, right click on the Top-level design, then click schematic view to see the schematic generated by the design vision.
Set the design constraints as explained in the previous section and compile the design with the clock period of 5ns. Then save the design and generates the reports.

To display critical paths in the schematic view:

1. Choose, *Schematic* > *New Design Schematic View*. Make sure the schematic view is the active view.
2. Select the critical paths by choosing, *Select* > *Paths From/Through/To*, and then clicking OK in the Select Paths dialog box. The default options are set to select the path or paths with the worst negative slack in the design.
3. Highlight the selected paths by choosing, *Highlight* > *Selected.*
4. If you can not see the critical path, try to ungroup the blocks of the design hierarchy until you do.
5. Use the magnification tools and the pointer to identify the startpoints and endpoints of the critical path.
6. Compare the start/endpoint with the one in the timing report.

Question 12: Does the critical path cross the boundaries of the modules? Write the start point and end point of the critical path in the table.

Architecture	12_MAC	2_MAC	1_MAC
Start point			
End point			

Question 13: Does the start and the end points of the critical path match with what is reported in the timing report?

Click on the ArithUnit on the schematic to see inside of the module. Highlight the critical path of this module.

Question 14: How many MACs are there in the critical path of the ArithUnit? Justify the observation and fill in the following table

Architecture	12_MAC	2_MAC	1_MAC
Number of MACs in the critical path			

Question 15: Which designs have the shortest and the longest critical paths? Does the critical path match with the one that you calculated in questions 2, 3 and 5?

Clock period = 5ns

Architecture	12_MAC	2_MAC	1_MAC
Area			
Timing Slack			

Question 16: Is the performance requirement met in all designs? Which design is the smallest and which one is the largest (area)? Why?

Change the clock period and recompile the designs.

Clock period = 2.5 ns

Architecture	12_MAC	2_MAC	1_MAC
Area			
Timing Slack			

Question 17: Is the timing requirement met or violated for each of the designs? Why?

Gate Level Simulation

In this section of the lab, your task is to do gate-level simulation using modelsim for your **partially parallel neuron (2-MAC)**. The purpose of the gate-level simulation is to verify the functionality of the gate-level netlist. You should show your working gate-level simulations to the lab assistant. For gate level simulation:
Create a folder containing:

1. The Verilog netlist of your design ($home/IL2225_LAB/2_MAC/db/main.v)
2. packages
3. Testbench
4. tcbn90g.v library file

You can find this library file in the directory:

```
/afs/it.kth.se/pkg/synopsys/extra_libraries/standard_cell/TSMC/tcbn90g_110a/Fron
t_End/verilog/ tcbn90g_110a$
```

For gate-level simulation you need to consider that:
After/during synthesis, check for the name of your top level design. Synthesis tool changes the name according to generics used. For instance, the name of top level in our design may be changed. You have to use this name in your test bench for proper simulation.

Check if the output is correct. You must show your gate level simulation to the assistant to pass this lab.

Remember that when you do gate level simulations, you should disable the optimizations in your simulator. You can do that in vsim by ticking the *disable optimization* box. In newer versions of vsim this option has been discontinued and will prompt an error. In this case you can disable the optimizations for inside the optimization options, by setting optimization level to -O0 and enabling *full visibility to all modules* in the visibility tab.

Question 17: Does your gate level simulation match the RTL simulation? Be prepare to run the simulation during the Lab check.

Optional: Write a synthesis script that you can use to synthesize the design similar to the one in the appendix for all the different architectures.

Lab 2: Logic synthesis and design optimizations

1. Objective

The purpose of this lab is to introduce following concepts:

1. Power Estimation using Value Change Dump File (VCD File)
2. Bottom Up Hierarchal Synthesis
3. Following Power Saving Methodologies during Synthesis:
    a. Clock Gating
    b. Operand Isolation

Each student in the group should write an individual report answering the questions highlighted with red. The report should be submitted in CANVAS in day (24 hours) before the scheduled lab-check.

2. Preparation Tasks

Synthesize the previously developed 2_MAC code in $IL2225_LAB/LAB2. You should also generate the following files during synthesis.

Report the Following:

Area:_______________Slack:_______________Power (Total):_______________
Operating Condition:_______________WireLoadModel:____________________
WireLoadModel Mode :____________________

1. Constraints.sdc file, it will be used in the next lab. You can generate it by using the following command at the end of the lab

```
write_sdc $ home/IL2225_LAB/LAB2/ MAPPED/constraints.sdc
```

2. Delays.sdf file: This file will be used to back annotate the delays of the circuit in our gate-level simulation. You can generate it by using the following command

```
write_sdf $ home/IL2225_LAB/LAB2 /MAPPED/delays.sdf
```

3. Power Estimation

In lab1, we use report_power to get the power estimation for our circuit. This estimation is based on statistical switching activities of all the wires in our circuit. By default this switching activity is 0.2. In reality the power consumption of an electronic circuit highly depend on the input data. In this section we will do gate level simulation of the neuron. ***Use the netlist which is synthesized in the preparation task***.

Compile the sdf file in vsim while doing the gate level simulation. You can also use GUI to add the sdf file into your simulation in modelsim. For that, click simulate->start simulation-> select sdf tab and select the location of the sdf file as well as its scope which will be /<name of the testbench entity>/<name of the DUT module instantiation>. This will back annotate all the delays in the circuit during our gate level simulation. During this gate-level simulation, we will record the activities inside the circuit in a Value Change Dump file. This VCD file will then be used to accurately estimate the power in our neuron.

1. Create a testbench for the neuron with sufficient amount of samples. A good testbench can have a random number generator to generate random samples at every sample clock. No. of samples in the testbench should be selected or specified by a constant. You can change number of samples by simply changing this constant. Use the following process to edit your test bench for random sample generation.

Fig1: *Code for Random Number Generation*

```
random_generator:process
    VARIABLE seed1:positive:=1;
    VARIABLE seed2:positive:=15;
    VARIABLE a_real: real;
    VARIABLE count :INTEGER:=0;
begin

 for j in 1 to number_of_samples loop
   wait until sample_clk'event and sample_clk = '1';
   uniform(seed1,seed2,a_real);
   a_real := a_real * 16.0;
   if(count < number_of_samples) then
   sample_wire  <=  conv_std_logic_vector(INTEGER(a_real),  4)  ;
   count := count+1;
   end     if;
 end loop;
end process random_generator;
```

2. Check the slack of the synthesized netlist by looking into the timing report. This will tell you the frequency at which the neuron will work. This will be the clock frequency which will be used in the test bench.
3. Now use the following commands to generate the VCD file
    a. Run the simulation just before the first sample is generated by the test bench by using the command
    b. At this point, create a VCD file by using the command
    c. Add the signals which are to be recorded in the VCD file. '-r' switch add everything in the design
    d. Run the simulation.
    e. Quit the simulation.

Note: Repeat steps for each of the cases given in Table-III.

```
a. run xxx ns
b. vcd file PATH/filename.vcd
c. vcd add -r <name of testbench entity>/*
d. run xxx ns
e. quit –f
```

Table I: Script to create a VCD file in ModelSim

4. Synopsys do not support VCD file format and instead supports Switching Activity Information Format. To use this technique in Synopsys, either we need to generate saif file or we can convert a VCD file into a saif format file using the following command. Use the following command on unix/linux prompt to generate saif file.

   vcd2saif -i myvcdfile.vcd -o mysaiffile

5. Now you have to use this saif file to do the power estimation in Synopsys.

```
read_file -format verilog {$ home/ IL2225_LAB/LAB2/MAPPED/<name>.v}
elaborate 2_MAC -lib WORK set_wire_load_mode segmented
set_wire_load_model - name TSMC8K_Lowk_Conservative
read_saif -input $ home/ IL2225_LAB/LAB2/mysaiffile -instance_name <tb_name>
report_power
```

Table II: Commands to estimate power in Synopsys using Saif File

Question 1: Fill in the following table for the neuron with 8-bit width.

No. of Sample	Simulation Cycles/Time	Power
5		
50		

Table-IIIA: Power Vs No. of Samples

6. Change the width of neuron to 16-bits by changing the generics in the RTL Code. Synthesize the design in same condition as above. (Note: Keep a backup of the source code. The original source code with 8-bits will be used further in the lab)

Question 2: Fill in the following table for the neuron with 16-bit width.

No. of Sample	Simulation Cycles/Time	Power
5		
50000		

Table-IIIB: Power Vs No. of Samples

Question 3: Explain the difference in the average power for different no. of sample in Table-IIIA and IIIB. Why is the power estimation different?

!!!ATTENTION!!! Make sure that you do not re-synthesize the design before and after the simulation. When you do the power estimation in design vision the .saif file must match to the loaded design. If you compile -> generate netlist -> simulate -> generate the .saif file and then re-compile before loading the saif file, the design in design vision will not match the original netlist. You can load the .dcc file that you saved together with the netlist to load the exact same design in design vision.

7. Use the saif file, used for table above, for 500 samples to fill the Table-IIIC.

Question 4: Explain the difference in power results.

wire_load_mode	Wire_load_model	Samples	Power
Top	TSMC8K_Lowk_Conservative	500	
enclosed	TSMC8K_Lowk_Conservative	500	
segmented	TSMC8K_Lowk_Conservative	500	

Table-IIIC: Power Vs No. of Samples

4. Power Saving During Synthesis

(Note: Use the original 8-bit code for this exercise)

There are two basic ways to implement power saving techniques during synthesis. These are:

1. Clock Gating
2. Operand Isolation

The theory behind these techniques will be discussed during the lectures. In this lab we will use the Synopsys synthesis tool to implement these in our design and see their effect. Before implementing these on our design please answer the following questions.

Question 5: What do you understand by clock gating?
Question 6: What do you understand by Operand Isolation?

(Hint: Please refer to the lecture notes if you do not know about Clock gating and Operand Isolation.)

Clock Gating:

Enabling clock gating in Synopsys design vision requires slight modification to the synthesis script that we used in the LAB1. These are:

- Using set_clock_gating_style to set the parameters to determine when to use clock gating and the type of the clock gating used. The clock gating can be latch based or latch less based. Read the online documentation to find the difference between these two styles.
- In this lab task, we will use latch based clock gating style. Use the following command before analyzing the source files: set_clock_gating_style -sequential_cell latch

- While compiling the design, you have to add the parameter -gate_clock to implement the clock gating function.

Question 7: What is the difference between the two clock gating styles? What are their benefits and disadvantages?

Report the Following:

Area:________________Slack:________________Power (Total):________________

Operating Condition:________________WireLoadModel:______________________

WireLoadModel Mode :_____________________

Operand Isolation:

In order to enable operand isolation, You need to set the following variable in your synthesis script.

- set do_operand_isolation "true"

Question 8: Re-synthesise your design using clock Gating and Operand Isolation. Fill the following tales. Explain the difference in Area, Timing and Power. Do they increase, decrease, or stay the same? Why?

Design Name	Power Saving Technique	Area	Timing	Power
	None			
	Clockgating and Operand Isolation			

Table-IV: Effect of Clock Gating and operand isolation (Report power with Saif)

Design Name	Power Saving Technique	Area	Timing	Power
	None			
	Clockgating and Operand Isolation			

Table-V: Effect of Clock Gating and operand isolation (Report power without Saif)

5. Hierarchal Bottom Up Synthesis

In this section we will look into a bottom up synthesis technique. Bottom up synthesis has the following advantage over the top down synthesis;

1. Large designs are compiled with divide and conquer approach.
2. Bottom up synthesis is not limited by the size of available memory. In top down synthesis, available memory is a major limiting factor for the size of the design.Disadvantages include;

1. It is cumbersome to do bottom up synthesis since it may involve much iteration to get it done.
2. Careful revision control is required to keep track of changes in the

design. The bottom-up compile strategy requires these steps:

1. Develop both a default constraint file and subdesign-specific constraint files. The default constraint file includes global constraints, such as the clock information and the drive and load estimates. The subdesign-specific constraint files reflect the time budget allocated to the subblocks.
2. Compile the subdesigns independently.
3. Read in the top-level design and any compiled subdesigns not already in memory.
4. Set the current design to the top-level design, link the design, and apply the top-level constraints. If the design meets its constraints, you are finished. Otherwise, continue with the following steps.
5. Apply the characterize command to the cell instance with the worst violations.
6. Use write_script to save the characterized information for the cell. You use this script to re-create the new attribute values when you are recompiling the cell's referenced subdesign.
7. Use remove_design -all to remove all designs from memory.
8. Read in the RTL design of the previously characterized cell. Recompiling the RTL design instead of the cell's mapped design usually leads to better optimization.
9. Set current_design to the characterized cell's subdesign and recompile, using the saved script of characterization data.
10. Read in all other compiled subdesigns.
11. Link the current subdesign.
12. Choose another subdesign, and repeat steps 3 through 9 until you have recompiled all subdesigns, using their actual environments.

A **pseudo code** of the compile script is given in Appendix A.

Question 9: A basic bottom up synthesis script, named bottomup_outline.scr is given in the LAB2 directory under SCRIPT directory. This script is not automatic, so for every pass you need to execute it using File->Execute Script in Design Vision. Your task is to complete the script and execute it to perform bottom up synthesis of your code. Set the clock period to 2ns.
Question 10: Explain what the characterization command is doing and why is needed.

Report the Following:

Area:_______________Slack:_______________Power (Total):_______________
Operating Condition:_______________WireLoadModel:____________________
WireLoadModel Mode :____________________

Good luck.

Lab 3: Physical Design, and characterization

1 Requirement

For this lab, you need the following files from Synopsys Design Vision

1. A gatelevel netlist file, synthesised by using tcbn90gtc.lib which is the .v file.
2. Synopsys Delay Constraints file, which can be generated using the command

```
write_sdc $ home/IL2225_LAB/LAB2/ MAPPED/constraints.sdc
```

To make this Lab more interesting, we will use a gatelevel netlist file which is synthesised while maintaining its hierarchy. For that purpose, you shouldn't use the ungroup –all command in the script which you have created in LAB1 and used in LAB2.

Each student in the group should write an individual report answering the questions in red. The report should be submitted in CANVAS at least 1 day (24 hours) before the scheduled Lab-check.

2 Cadence Innovus Implementation System

This task is the core of the lab work. Here, you will import the design into Cadence Innovus Implementation System for placement, routing... In Innovus, as in Design Vision, everything can be done using a script. However, this lab will take you through the graphical user interface and command- line which is more convenient at the beginning. Write "innovus" in the command line to start the tool.

Design import into Innovus

First, we need to import the design into Innovus. This requires the .v file that was generated in LAB2, the technology timing library files and the library exchange format (LEF) files. Notice in the lower right corner that there is no design in the memory.

Go to Design > Import Design...In the Verilog Netlist field, selects the .v file that you have saved in the previous lab. Select 'BY User' for the Top Cell and type the name of the top module in the corresponding field (An easy way to find out the name of the top cell in case you have trouble is to check the .sdf file that you have to generate using Design Vision).

In the LEF Files field, select the three dots, then press the folder icon and type the following. In the Filter box:

/afs/it.kth.se/pkg/synopsys/extra_libraries/standard_cell/TSMCHOME/digital/Back_End/
lef/tcbn90g_110a/lef

Double click on the file tcbn90g_9lm.lef and press **Close**.

In the Power Nets field, type **VDD** and in the Ground Nets field type **VSS**.

In the Analysis Configuration field, select the "mmmc. Tcl" file which is given on the webpage. This .tcl script identifies the path to the timing libraries and the timing constraint file "constraints.sdc". Check the path to the constraint file and change it if needed. Press the save button to save the configuration, and then press OK. It will take some time for Innovus to load the technology libraries. You can watch what is happening in the terminal where you started Innovus. As you can see in the lower right corner of the main window, the design now is In-Memory.

In the command window in which you run Innovus run the following command:

```
setDesignMode -process 90
```

This command will specify that you are implementing a design in the 90 nm technology node. Take a look at the command window.

Floorplan

Now, we will floorplan the chip. Go to Floorplan > Specify Floorplan...

In the Core Utilization box type 0.5. In the Core Margins by field, specify a Core to IO Boundary of 9 on all sides. Press OK.

Go to Power > Connect Global Nets... Select the Pin, type VDD in the Pin name box and VDD in the To Global Net box. Press Add to List. Type VSS in the 'Pin name' box and VSS in the 'To Global Net' box. Press Add to List again. Press Apply then Close.

Question 1: What is the utilization?

Question 2: Why do we keep a utilization of 0.5? Why not increase the Utilization close to 1?

Power and ground routing

We will add to our design rings and stripes for power and ground. To this end, go to Power

>Power Planning > Add Rings... Make sure that the Net(s) field contains both VDD and VSS. Press OK. Note how the rings are routed in metal 1 and metal 2.

We will now add power and ground stripes to connect the cells to the rings. Go to Power >Power Planning > Add Stripes. In the Net(s) box, type VDD VSS. In the Set-to-set distance box, type 25. In the Start box, type 25. Make sure that you choose M2 as the metal layer. Press OK.

You can use the Undo button to remove the stripes. Try to route the stripes in metal layer 1. Do you notice anything different?

Go to Route > Special Route... Make sure that VDD and VSS are in the Net(s) field. In the Via Generation tab select all the boxes in the Make Via Connections To box. Press OK. Notice how the power and ground lines are now routed.

Question 3: What is the purpose of the Special routing command? Why are the power and ground lines routed in regular intervals?

Placement of the design

In order to place the blocks go to Place > Standard Cells.

Press the Mode... button and click the Place IO Pins option. Press OK to close the Mode window and OK again to place the standard cells.

In order to view the blocks, change view by toggling between the Floorplan view, Amoeba view, and Physical view buttons in the upper right corner of the main window.

Question 4: Report the timing and write down the reported slack.

Clock tree synthesis

Now, we will synthesise the clock tree. To do that go to the terminal window in which you run Innovus and run the following command:

```
ccopt_design
```

Click the physical view to see the clock network in the design. Hide the other Nets from the view using the side panel to the right, keep only the clock.

Display the clock tree by selecting Timing > Display Timing Map..., then press OK. Change to the Physical View button. Click Apply to the panel on the left. Different colours will show the clock skew in the different cells of the design.

Question 5: What does the clock tree synthesis do? Report the timing and write down the reported slack.

Routing

Now we will route the design using NanoRoute. Go to Route > NanoRoute > Route...

Select Timing Driven, then press OK. Use the report_timing command to check the timing on your design. If your design has a negative slack go to ECO -> Optimize Design... check Post-Route and press OK. Check the timing again.

Question 6: Write down the reported slack.

Adding Filler cells and metal

We will add filler cells to provide NWell continuity. Go to Place -> Physical Cells-> Add Filler... In the Cell Name(s) field, press Select. Select all the cells in the Cell List (press on the topmost cell then click on the bottommost cell while pressing Shift), then press Add and Close. Press OK.

We will also add metal filler cells to meet the minimum metal coverage density. Go to Route> Metal Fill > Add ... Press OK.

Question 7: Why do we use filler cells and metal?

Question 8: Report the timing and write down the reported slack. Compare the timing and slack to the other steps. Is it the same? Why does it change?

Be ready to show the final layout to the lab assistant during the Lab-check.

- ## GDS File export

Now that the design is ready, we will export it in GDS format that can be used directly for fabrication, or that can be imported into another routing tool to be connected to RF/analog/mixedsignal blocks. Go to File > Save > GDS... In the Output Stream File box, type 2_MAC.gds.

Verilog netlist export

The clock tree synthesis has added buffer cells to the design. Therefore, the new netlist does not correspond exactly to the one obtained from the synthesis process. To save the new Verilog netlist of the design, select File > Save > Netlist ... Save the netlist in the file: ~/ASICLABS/LAB3/MAPPED/Phy_2_MAC.v.

Calculating and exporting delays

Select Timing > Extract RC... and press OK on the popup window. This will extract RC data on the design, which is used to calculate delays. Select Timing > Write SDF... Deselect ideal clock in the popup window and select ~/IL2225/LAB3/Mapped/Phy_2_MAC.sdf as SDF output file. Delays to annotate the Verilog netlist will be saved in this file. Press OK.

You also can use the following command after RC extraction to generate the SDF file:

```
>Write_sdf Phy_2_MAC.sdf
```

3 Saving the Design

Before quitting Innovus, save your design by clicking Design-> Save Design and then click ok to save the design. You can now close Innovus to proceed with gatelevel simulations.

4 Simulation

You will now simulate the netlist that you have obtained in the previous task, after having back annotated it with the delays calculated by Innovus. To simulate the system, you will use the Cadence simulator NCSim.

To start NCSim, change to LAB3 directory and type nclaunch. Follow the following steps to simulate in NCSim:

Set design directory by clicking on file->set Design Directory. Now select LAB3 directory as your design directory. Under library mapping file select LAB3 directory and click create cds.lib file. Press ok to continue.

1. Now compile your physical design gatelevel netlist, along with gatelevel library and your test bench. Make sure that there is a match between the name of the top module in the netlist and the testbench. Click on “Enable VHDL 93 features” when compiling the test bench.

2. To compile the sdf file, first select *.sdf in the filters. Now browse to the .sdf file and compileit.

3. Under worklib, right-click on your test bench, and click NCElab.

4. You elaborated test bench will appear under snapshots now. Right-click on it and press NCSim. This will open the simulation window. Now you can simulate your design

5 Creating VCD in NCSim

To generate VCD in NCSim, you should apply the following commands while simulating the design;

```
run -timepoint abc ns –absolute
database –open $HOME/IL2225/LAB3/VCDFile.vcd -vcd -default -timescale ns
probe -create :u1 -vcd -all -depth all
run -timepoint xyz ns -absolute
database -close $HOME/IL2225/LAB3/VCDFile.vcd
```

(Note: Change the values according to your requirement in lab)
Question: Is your gate-level Simulation working correctly? Show it to the assistant.

-- Alternative if you encounter issues you can use Questasim for the gate-level simulation same as in LAB2 (Make sure you disable optimisations)--

6 Power Calculation in Innovus

Restore your design in Innovus by clicking File -> Restore Design, now search for your stored design and then click Ok to restore it. Now you can calculate power in innovus by writing the following commands:

1. Read in the VCD activity file by using the following command

read_activity_file -format VCD ***fileame.vcd*** -scope ***<testbench entity name>/<module instantiation name>*** -start ***0ns*** -end ***354ns***

-scope: Is the name of the toplevel of your design. You can find the name in the VCD file. Just check your VCD file in a text editor to find the scope of your design.

-start/-end: These are the start and end time at which the VCD file is created.
(Note: You should change the operands which are highlighted in bold with respect to the values/names used in your design). If you encounter problems, make sure that the start/end values are inside the simulation interval.

!!!ATTENTION!!! Make sure that the above scope, start and end match your design and simulation. You might have to change it.

2. Report power by -hierarchy > report_power -hierarchy 0

Question 9: Check the command line, what is the reported signal annotation?
Question 10: What does this annotation tells you? (If your annotation is less than ~90% make sure that your scope during the simulation and in the read_activity_file command is correct)

Fill in the following table:

No. of Sample	Simulation Cycles/Time	Power
50		
5000		

Table-III: Power Vs No. of Samples

Note: You can see the help for report_power to see more uses of it. You can also explore power menu to learn more about power planning and power analysis options offered in Innovus.
Question 11: Which is the more accurate power report and why?

Good Luck!!!

Appendix A: Sample Synthesis Script

```
####### Set Global Libraries #########
source synopsys_dc.setup
####### Set Directory #########
set LIB typical
set SYNDIR /home/ASICLABS/IL2225_LAB/LAB1/12_MAC
set SRCDIR /home/ASICLABS/IL2225_LAB/LAB1/12_MAC/SOURCE
set SCRDIR /home/ASICLABS/IL2225_LAB/LAB1/12_MAC/SCRIPTS
set RPTDIR /home/ASICLABS/IL2225_LAB/LAB1/12_MAC/REPORTS
set SYNDB /home/ASICLABS/IL2225_LAB/LAB1/12_MAC/db
######Enviroment############
define_design_lib WORK -path $SYNDIR/WORK
######Read Design###########
analyze -library WORK -format vhdl {$SRCDIR/misc.vhd}
… <analyze the files here in here> …
######Elaborate Design###########
elaborate <top level design> -lib WORK
########Set Constraints#############
… <set up the constraints here> …
#######Compile Option############
compile -map_effort medium
#######Report####################
report_timing > $RPTDIR/report_timing.rpt
report_area > $RPTDIR/report_area.rpt
#######Save  Design###################
write -hierarchy -format ddc -output $SYNDB/main.ddc
write -format verilog -hier -o $SYNDB/main.v
write_sdf -version 2.1 $SYNDB/main.sdf
```

Appendix B: Bottom up pseudo code script

```
all_blocks = {E,D,C,B,A}
/* compile each subblock independently */ foreach (block, all_blocks) {
/* read in block */ block_source = block + ".v"
read_file -format verilog block_source current_design block
link uniquify
/* apply global attributes and constraints */ include defaults.con
/* apply block attributes and constraints */ block_script = block + ".con"
include block_script
/* compile the block */ compile
}
/* read in entire compiled design */ read_file -format verilog TOP.v current_design TOP
link
write -hierarchy -output first_pass.db
/* apply top-level constraints */ include defaults.con
include top_level.con
/* check for violations */ report_constraint
/* characterize all instances in the design */ all_instances = {U1,U2,U2/U3,U2/U4,U2/U5}
characterize -constraint all_instances
/* save characterize information */ foreach (block, all_blocks) { current_design block
char_block_script = block + ".wscr" write_script > char_block_script
}
/* recompile each block */ foreach (block, all_blocks) {
/* clear memory */ remove_design -all
/* read in previously characterized subblock */ block_source = block + ".v"
read -format verilog block_source
/* recompile subblock */ current_design block link
uniquify
/* apply global attributes and constraints */ include defaults.con
/*  apply  characterization  constraints  */  char_block_script  =  block  +  ".wscr"  include
char_block_script
/* apply block attributes and constraints */ block_script = block + ".con"
include block_script
/* recompile the block */ compile
}
```