

EE 465: DIGITAL VLSI DESIGN

Linear Regression Module: *Design, Verification and Layout*

Final Project | Fall 2017

Saunak Saha

(saha@iastate.edu)

12/13/2017



Table of Contents

Mathematical Specification	4
Hardware specification	5
Special Notes.....	6
RTL Level Design.....	8
Modularity.....	9
Robustness.....	9
Logic Optimizations.....	9
Testing.....	10
Input 1 file:	10
Input 2 file:	11
Input 3 file:	12
.....	12
Simulation:	13
.....	13
Functional Specifications	14
Pre-Logic Optimization.....	15
Post-Logic Optimization	15
Synthesized circuit	17
Final Synthesis reports:	18
Timing Report (Worst Slack):	18
Area Report: (Data path).....	19
Area report: (Netlist statistics).....	19
Power report: (Detailed)	20
Floorplan Values:	22
Encounter Layout Report:	22
Encounter Reports:	23
Area report: Physical View:	23
Area report: Amoeba View	24
Power Report:	25
Timing Report:.....	26
Learning Experience	27
Difficulties	27
Feedback	28



Verilog code for Linear Regression Module:.....	29
Verilog code for Multiplier:.....	34
Verilog code for Rounder:.....	34
Verilog code for Up Counter:	35
Test bench: (uses revised data)	35
Script for synthesis:.....	39
Script for Layout:.....	42

Introduction

In the era of Machine Learning, every device is better utilized when it manifests some kind of intelligence in its operations. An intelligent device, can, with the help of machine learning, simply 'learn' the trend of data and after it has learnt the distribution or occurrence, it can generate the data by itself. This, albeit in extremely simplified words, is one of the fundamental blocks of machine intelligence. A way to predict data is provided through statistical analysis of previously sampled data. Many different statistical models are used for this purpose and each have their own advantages and disadvantages. It is how quick the machine can learn the pattern or correlation between data, that determines its efficacy. Linear Regression is one such statistical model.

Linear regression determines whether there exists a linear or linear-like relationship between a dependent and an independent variable and if so, what is the extent of linearity. It is very frequently used as a model for data prediction. It takes in a finite number of values and builds a regression line. This line roughly describes the correlation that exists between the data pairs. Now as the number of data pairs accepted becomes large, the model gets more and more accurate. Now, a single value of independent variable may be given to the system which uses the model, thereby developed, to compute the expected value of the dependent variable. As the machine takes in more and more data pairs in the process, the error in prediction reduces.

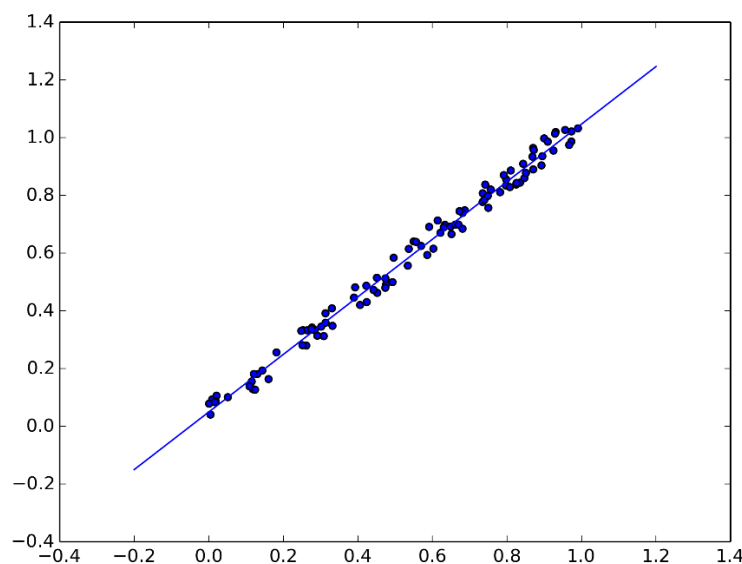


Fig.1 A Linear Regression Line developed from sampled datasets

Project Description

Mathematical Specification

This project aims to design a Linear Regression Core which samples data from the user at its own pace and will collect that data pair up to a point when the user requires a predicted value of the dependent variable. The Linear Regression Core will build the Regression model in real-time as it collects each and every dataset. Therefore, when it is asked to predict a value, it can compute the most accurate value possible at that time. Mathematically, this model will be built by computing two Regression Parameters A and B . These values of A and B are the ones that will make the circuit's linear function as close as possible to the correlation detected by the circuit. The predicted value of the dependent variable will then become a linear function of these parameters and the sampled independent variable.

Mathematically, this core implements the following mathematical functions:

$$A = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{\sum_{i=1}^n x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^n (x_i^2) - n\bar{x}^2}$$

$$B = \bar{y} - A \bar{x}$$

where $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ and $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$

Now, for the prediction, the circuit will simply implement:

$$Y = AX + B$$

The circuit samples 8-bit data pairs and therefore, mathematically, there is a limit to the precision with which prediction is possible. Also, the use of division creates some error in the computation but it has been minimized and rounded to maintain accuracy.

Hardware specification

The top-level design specification is provided below in Fig 2. The Linear Regression Module is a mathematical processor that will take in 8-bit data pairs as inputs (X_i , Y_i). The circuit provides a READY output to inform the user that it can accept an input pair at the next clock (CLK) edge. This core has been designed to be able to accept one input in every four clock edges which means there is a gap of three clock edges during which it performs the necessary computations to update the regression model. Of course, the user has the freedom to assert a PREDICT signal at any time during the operation. An active high RESET has also been provided to clear the model and start from scratch.

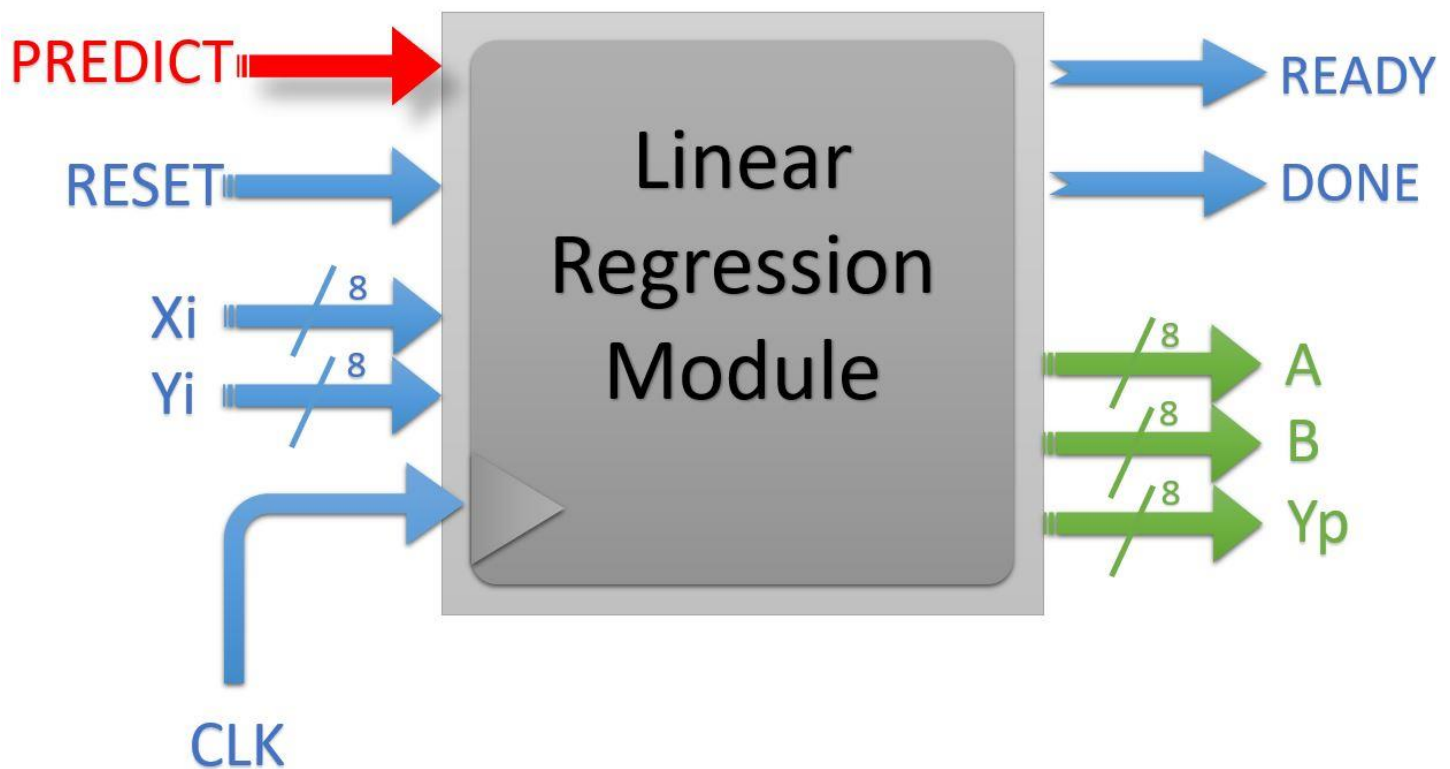


Fig 2. Top level hardware specification

The output is sampled by the user when the DONE signal is asserted by the circuitry. It means that the machine has completed all its computations and the output available at the pins A, B and Y_p are the actual outputs provided by the system.



Special Notes

1. If predict is asserted before providing any input data, the circuit will produce a 128 at the Yp signal and zeros will be provided at the A and B terminal.
2. If predict is asserted immediately after the first data pair, the circuit will use the Yi value from the last input sampled and route that directly to the Yp terminal. (Note that A and B computation are not possible in these two cases)
3. The circuit is able to read data pairs continuously up to a limit of 32,768 pairs. This is the learn capacity of this module.
4. The circuit performs best when the accepted values generate A and B values within 0 and 255. If not, some overflow will occur.



Work Summary

- In this project, I have written the RTL level design of the 8-bit Linear Regression module. Functional correctness of the module has been tested against the provided test bench (REVISED version) and all three I/O Files. Simulation, Testing and Verification has been performed in ModelSim platform. The Verilog 2001 code/s have been provided.
- Post functional testing of the basic RTL design, some design-level optimizations were carried out. This includes hardware sharing/reuse, power optimizations etc.
- I have synthesized the design using a ***tsmc 65nm CMOS library*** in Cadence RTL Compiler environment. The synthesis reports and optimizations have been reported.
- I have carried out the Layout I.e. Placement and Routing of the design using the same library and a tsmc 65nm LEF provided by the department. This has been done in Cadence Encounter environment. The Layout optimizations and reports have been provided.

The next section of the report will describe in detail, about the design methodology that has been used in this project.

Design methodology

RTL Level Design

The design style that has been used is mostly behavioral in nature. The circuit operates in a simple and step by step fashion. First, it checks for new user input pairs at every four clock cycles. If there is an input at the input terminal the circuit will sample it and add it to the model building.

The model building is done with by calculating four different coefficients:

- **ΣXY** : This is the summation of data pair products
- **ΣX^2** : This the sum of all the X_i squared in real time
- **ΣX** : This is the sum of all X calculated in real time
- **ΣY** : This is the sum of all Y calculated in real time


Using these models and these only, all the calculation can be completed. The circuit will also keep updating the number of input pairs that have been samples in a counter **N** . Every time this counter switches a value, the circuit will run the calculation of all these coefficients once and update the model. This goes on until the PREDICT signal is asserted by the user.

When the user asserts the PREDICT signal, the N counter is not updated as this new data pair will not be used to build the model further. Accordingly, the calculation of coefficients is also stopped for the time being. But, nevertheless, the circuit does not lose the model that has been built so far. It keeps the model saved for future update after the prediction cycle is over. Whenever PREDICT is asserted, the circuit will start calculating the regression parameters A and B .

The optimized formula for A and B reducing the number of divisions was used:

$$A = \frac{N \times \sum XY - \sum X \times \sum Y}{N \times \sum X^2 - \sum X \times \sum X}$$
$$B = \frac{den \times \sum Y - num \times \sum X}{den \times N}$$

Where *num* and *den* are the numerator and denominator respectively from A computation



As mentioned before, rounding logic has been used to minimize error in computations. A module '**Rounder**' has been designed to handle the Rounding of A and B. The Y_p value is computed using the rounded values of A and B. Therefore, no more divisions were encountered in the computation of Y_p .

Modularity

The code has been modularized through the use of separate modules for various functions that could be used as many number of times as required. I have designed a Multiplier module, A Rounder module, an Up-counter module. These modules can be used as many number of times as required and wherever required. Also, the code is written in behavioral style because it is easier and less verbose for this application, rather than the structural style of coding. Therefore, the operations involved in calculating the 4 coefficients for computation, have not been put into separate modules.

Robustness

The design has been made to show absolute robustness for all distributions of data pairs. The different kinds of optimizations used does not affect the original accuracy of operation, neither does it cause the circuit to go to some previously unknown state. All exceptions have been handled so as to keep the design more generalized for any kind of use.

Logic Optimizations

The design was optimized to obtain some logic level improvement over the original functional code. The following optimizations were carried out:

- The Multiplier in the circuit used in calculating the product and the squares of the sampled inputs were shared into one. This multiplier has been defined in the Multiplier module.
- The calculation of various numerators and denominators were limited to only when predict signal is present. So, in a way, this will save switching power related to unnecessary calculation of bulky values, using multipliers to do that.

Simulation, Testing and Verification

Testing

The design was tested using the REVISED test bench provided on the Lab page. This was done for all the input files and against all the output files. The following screens show Successful verification of the module.

Input 1 file:

```
# ***** START to VERIFY the Linear Regression Module OPERATION *****
# <Xi,Yi,PREDICT> = < 26, 65,1>
# <Xi,Yi,PREDICT> = < 22, 55,0>
# <Xi,Yi,PREDICT> = < 24, 59,1>
# <Xi,Yi,PREDICT> = < 8, 27,1>
# <Xi,Yi,PREDICT> = < 38, 88,0>
# <Xi,Yi,PREDICT> = < 29, 70,0>
# <Xi,Yi,PREDICT> = < 15, 43,0>
# <Xi,Yi,PREDICT> = < 5, 21,0>
# <Xi,Yi,PREDICT> = < 37, 86,0>
# <Xi,Yi,PREDICT> = < 30, 71,1>
# <Xi,Yi,PREDICT> = < 36, 84,0>
# <Xi,Yi,PREDICT> = < 36, 85,0>
# <Xi,Yi,PREDICT> = < 5, 22,0>
# <Xi,Yi,PREDICT> = < 5, 23,0>
# <Xi,Yi,PREDICT> = < 26, 64,0>
# <Xi,Yi,PREDICT> = < 35, 81,0>
# <Xi,Yi,PREDICT> = < 31, 73,0>
# <Xi,Yi,PREDICT> = < 33, 79,0>
# <Xi,Yi,PREDICT> = < 23, 57,0>
# <Xi,Yi,PREDICT> = < 22, 56,0>
# <Xi,Yi,PREDICT> = < 17, 47,0>
# <Xi,Yi,PREDICT> = < 7, 25,0>
# <Xi,Yi,PREDICT> = < 38, 89,0>
# <Xi,Yi,PREDICT> = < 3, 18,0>
# <Xi,Yi,PREDICT> = < 37, 85,0>
# <Xi,Yi,PREDICT> = < 32, 75,0>
# <Xi,Yi,PREDICT> = < 42, 97,0>
# <Xi,Yi,PREDICT> = < 23, 58,0>
# <Xi,Yi,PREDICT> = < 20, 52,0>
# <Xi,Yi,PREDICT> = < 31, 73,0>
# <Xi,Yi,PREDICT> = < 8, 28,0>
# <Xi,Yi,PREDICT> = < 11, 33,0>
# <Xi,Yi,PREDICT> = < 3, 17,0>
# <Xi,Yi,PREDICT> = < 26, 65,0>
# <Xi,Yi,PREDICT> = < 39, 89,0>
# <Xi,Yi,PREDICT> = < 4, 21,0>
# <Xi,Yi,PREDICT> = < 25, 62,0>
# <Xi,Yi,PREDICT> = < 16, 43,0>
# <Xi,Yi,PREDICT> = < 38, 87,0>
# <Xi,Yi,PREDICT> = < 5, 21,1>
# <Xi,Yi,PREDICT> = < 28, 68,0>
# <Xi,Yi,PREDICT> = < 9, 30,0>
# <Xi,Yi,PREDICT> = < 17, 45,0>
# <Xi,Yi,PREDICT> = < 41, 94,0>
# <Xi,Yi,PREDICT> = < 33, 78,0>
# <Xi,Yi,PREDICT> = < 1, 14,0>
# <Xi,Yi,PREDICT> = < 16, 43,0>
# <Xi,Yi,PREDICT> = < 24, 60,0>
# <Xi,Yi,PREDICT> = < 41, 94,0>
# <Xi,Yi,PREDICT> = < 2, 17,0>
# <Xi,Yi,PREDICT> = < 14, 41,0>
# <Xi,Yi,PREDICT> = < 35, 82,0>
# <Xi,Yi,PREDICT> = < 20, 51,0>
# <Xi,Yi,PREDICT> = < 12, 35,0>
# <Xi,Yi,PREDICT> = < 13, 39,0>
# <Xi,Yi,PREDICT> = < 11, 34,0>
# <Xi,Yi,PREDICT> = < 23, 58,0>
# <Xi,Yi,PREDICT> = < 30, 72,0>
# <Xi,Yi,PREDICT> = < 14, 40,0>
# <Xi,Yi,PREDICT> = < 40, 93,1>
# PASS! All results are correct!
# -----
# Total number of cycles: 238
# -----
# ** Note: $finish : /home/saha/EE465FinalProject/lrm_tb.v(102)
# Time: 4772 ns Iteration: 0 Instance: /lrm_tb
# 1
```

Input 2 file:

```
VSIM 7> run
#
# ***** START to VERIFY the Linear Regression Module OPERATION *****
# <Xi,Yi,PREDICT> = < 45, 86,0>
# <Xi,Yi,PREDICT> = < 47, 88,0>
# <Xi,Yi,PREDICT> = < 22, 63,1>
# <Xi,Yi,PREDICT> = < 12, 55,0>
# <Xi,Yi,PREDICT> = < 35, 76,1>
# <Xi,Yi,PREDICT> = < 17, 58,0>
# <Xi,Yi,PREDICT> = < 3, 46,0>
# <Xi,Yi,PREDICT> = < 17, 59,0>
# <Xi,Yi,PREDICT> = < 48, 91,0>
# <Xi,Yi,PREDICT> = < 24, 66,1>
# <Xi,Yi,PREDICT> = < 34, 75,0>
# <Xi,Yi,PREDICT> = < 41, 83,0>
# <Xi,Yi,PREDICT> = < 22, 65,0>
# <Xi,Yi,PREDICT> = < 28, 69,0>
# <Xi,Yi,PREDICT> = < 4, 46,0>
# <Xi,Yi,PREDICT> = < 34, 76,0>
# <Xi,Yi,PREDICT> = < 27, 69,0>
# <Xi,Yi,PREDICT> = < 17, 59,0>
# <Xi,Yi,PREDICT> = < 39, 82,0>
# <Xi,Yi,PREDICT> = < 14, 57,1>
# <Xi,Yi,PREDICT> = < 7, 48,0>
# <Xi,Yi,PREDICT> = < 23, 65,0>
# <Xi,Yi,PREDICT> = < 12, 53,0>
# <Xi,Yi,PREDICT> = < 34, 76,0>
# <Xi,Yi,PREDICT> = < 26, 69,0>
# <Xi,Yi,PREDICT> = < 30, 72,0>
# <Xi,Yi,PREDICT> = < 39, 82,0>
# <Xi,Yi,PREDICT> = < 16, 59,0>
# <Xi,Yi,PREDICT> = < 34, 76,0>
# <Xi,Yi,PREDICT> = < 1, 44,0>
# <Xi,Yi,PREDICT> = < 6, 49,0>
# <Xi,Yi,PREDICT> = < 17, 59,0>
# <Xi,Yi,PREDICT> = < 40, 81,0>
# <Xi,Yi,PREDICT> = < 31, 74,0>
# <Xi,Yi,PREDICT> = < 32, 73,0>
# <Xi,Yi,PREDICT> = < 35, 76,0>
# <Xi,Yi,PREDICT> = < 35, 77,0>
# <Xi,Yi,PREDICT> = < 28, 69,0>
# <Xi,Yi,PREDICT> = < 46, 88,0>
# <Xi,Yi,PREDICT> = < 4, 45,1>
# <Xi,Yi,PREDICT> = < 32, 75,0>
# <Xi,Yi,PREDICT> = < 16, 59,0>
# <Xi,Yi,PREDICT> = < 15, 56,0>
# <Xi,Yi,PREDICT> = < 26, 68,0>
# <Xi,Yi,PREDICT> = < 49, 90,0>
# <Xi,Yi,PREDICT> = < 30, 72,0>
# <Xi,Yi,PREDICT> = < 31, 73,0>
# <Xi,Yi,PREDICT> = < 14, 55,0>
# <Xi,Yi,PREDICT> = < 37, 78,0>
# <Xi,Yi,PREDICT> = < 41, 83,0>
# <Xi,Yi,PREDICT> = < 42, 85,0>
# <Xi,Yi,PREDICT> = < 22, 63,0>
# <Xi,Yi,PREDICT> = < 40, 82,0>
# <Xi,Yi,PREDICT> = < 26, 68,0>
# <Xi,Yi,PREDICT> = < 16, 59,0>
# <Xi,Yi,PREDICT> = < 17, 60,0>
# <Xi,Yi,PREDICT> = < 17, 60,0>
# <Xi,Yi,PREDICT> = < 41, 82,0>
# <Xi,Yi,PREDICT> = < 21, 64,0>
# <Xi,Yi,PREDICT> = < 21, 62,1>
# PASS! All results are correct!
# -----
# Total number of cycles:          238
# -----
# ** Note: $finish      : /home/saha/EE465FinalProject/lrm_tb.v(102)
# Time: 4772 ns Iteration: 0 Instance: /lrm_tb
```

Input 3 file:

```
# ***** START to VERIFY the Linear Regression Module OPERATION *****
# <Xi,Yi,PREDICT> = < 23, 70,0>
# <Xi,Yi,PREDICT> = < 26, 79,1>
# <Xi,Yi,PREDICT> = < 28, 85,0>
# <Xi,Yi,PREDICT> = < 18, 55,0>
# <Xi,Yi,PREDICT> = < 15, 46,1>
# <Xi,Yi,PREDICT> = < 30, 91,0>
# <Xi,Yi,PREDICT> = < 30, 91,0>
# <Xi,Yi,PREDICT> = < 19, 58,0>
# <Xi,Yi,PREDICT> = < 23, 70,0>
# <Xi,Yi,PREDICT> = < 26, 79,1>
# <Xi,Yi,PREDICT> = < 23, 70,0>
# <Xi,Yi,PREDICT> = < 32, 97,0>
# <Xi,Yi,PREDICT> = < 23, 70,0>
# <Xi,Yi,PREDICT> = < 28, 85,0>
# <Xi,Yi,PREDICT> = < 3, 10,0>
# <Xi,Yi,PREDICT> = < 11, 34,0>
# <Xi,Yi,PREDICT> = < 1, 4,0>
# <Xi,Yi,PREDICT> = < 29, 88,0>
# <Xi,Yi,PREDICT> = < 33,100,0>
# <Xi,Yi,PREDICT> = < 32, 97,1>
# <Xi,Yi,PREDICT> = < 7, 22,0>
# <Xi,Yi,PREDICT> = < 23, 70,0>
# <Xi,Yi,PREDICT> = < 33,100,0>
# <Xi,Yi,PREDICT> = < 33,100,0>
# <Xi,Yi,PREDICT> = < 3, 10,0>
# <Xi,Yi,PREDICT> = < 12, 37,0>
# <Xi,Yi,PREDICT> = < 9, 28,0>
# <Xi,Yi,PREDICT> = < 26, 79,0>
# <Xi,Yi,PREDICT> = < 5, 16,0>
# <Xi,Yi,PREDICT> = < 12, 37,0>
# <Xi,Yi,PREDICT> = < 15, 46,0>
# <Xi,Yi,PREDICT> = < 12, 37,0>
# <Xi,Yi,PREDICT> = < 13, 40,0>
# <Xi,Yi,PREDICT> = < 33,100,0>
# <Xi,Yi,PREDICT> = < 1, 4,0>
# <Xi,Yi,PREDICT> = < 30, 91,0>
# <Xi,Yi,PREDICT> = < 5, 16,0>
# <Xi,Yi,PREDICT> = < 28, 85,0>
# <Xi,Yi,PREDICT> = < 10, 31,0>
# <Xi,Yi,PREDICT> = < 27, 82,1>
# <Xi,Yi,PREDICT> = < 7, 22,0>
# <Xi,Yi,PREDICT> = < 16, 49,0>
# <Xi,Yi,PREDICT> = < 10, 31,0>
# <Xi,Yi,PREDICT> = < 27, 82,0>
# <Xi,Yi,PREDICT> = < 10, 31,0>
# <Xi,Yi,PREDICT> = < 25, 76,0>
# <Xi,Yi,PREDICT> = < 28, 85,0>
# <Xi,Yi,PREDICT> = < 11, 34,0>
# <Xi,Yi,PREDICT> = < 27, 82,0>
# <Xi,Yi,PREDICT> = < 18, 55,0>
# <Xi,Yi,PREDICT> = < 28, 85,0>
# <Xi,Yi,PREDICT> = < 11, 34,0>
# <Xi,Yi,PREDICT> = < 33,100,0>
# <Xi,Yi,PREDICT> = < 1, 4,0>
# <Xi,Yi,PREDICT> = < 26, 79,0>
# <Xi,Yi,PREDICT> = < 29, 88,0>
# <Xi,Yi,PREDICT> = < 1, 4,0>
# <Xi,Yi,PREDICT> = < 19, 58,0>
# <Xi,Yi,PREDICT> = < 8, 25,0>
# <Xi,Yi,PREDICT> = < 18, 55,1>
# PASS! All results are correct!
# -----
# Total number of cycles:          238
# -----
# ** Note: $finish      : /home/saha/EE465FinalProject/lrm_tb.v(102)
# Time: 4772 ns  Iteration: 0  Instance: /lrm_tb
```

[illegible]

13

1. The circuit asserts READY signal once in every 4 clock cycles. So, there is a gap of 3 clock edges between two READY signals. That creates a total of 238 cycles for each test bench.
2. When the PREDICT is asserted by the user, the output (A, B and Yp) is available at the immediately next clock edge. Therefore, there is a Delay of ZERO clock cycles. Accordingly, the DONE signal is also asserted for the test bench to sample the outputs.
3. The design uses 2 divisions. I was unable to share the divider and reuse one hardware because that would upset the Zero clock cycle delay between PREDICT and outputs.

Check Appendix A for the Verilog code

Synthesis

The design was synthesized in Cadence RTL Compiler using a 65nm process. The synthesis loops take about 35 minutes each. The minimum clock period without any Timing Violations was found to be 2750 ps. But Layout with this clock period gives some slack because of how tight it is. After trying Layout for a number of different clock period options, I have used a clock period of 3150 ps and performed synthesis of both optimized and unoptimized designs. The following Synthesis report was generated when the original design was synthesized (without any optimization:

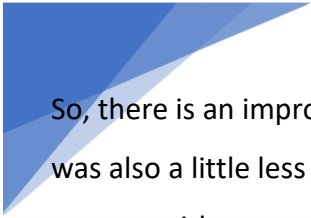
Pre-Logic Optimization

	Data path Area (μm^2)	Switching power (nW)	Worst Slack (ps)	Execution time (min:sec:ms)
<i>Without retiming and RC Clock gating</i>	51277.08	7.01	100 ps	31:14:06
<i>With retiming and RC Clock gating</i>	53117.76	6.99	121 ps	26:36:12

Synthesis was performed after the logic optimizations were made and the values were recorded.

Post-Logic Optimization

	Data path Area (μm^2)	Switching power (mW)	Worst Slack (ps)	Execution time (min:sec:ms)
<i>Without retiming and RC Clock gating</i>	50458.21	6.55	179 ps	22:11:45
<i>With retiming and RC Clock gating</i>	44666.64	6.22	193 ps	21:12:34



So, there is an improvement in Area as well as power, as a result of the logic optimization. The clock period was also a little less tight. Overall based on previous experiences, it was a good slack to start the Layout process with.

See Appendix B for Synthesis script

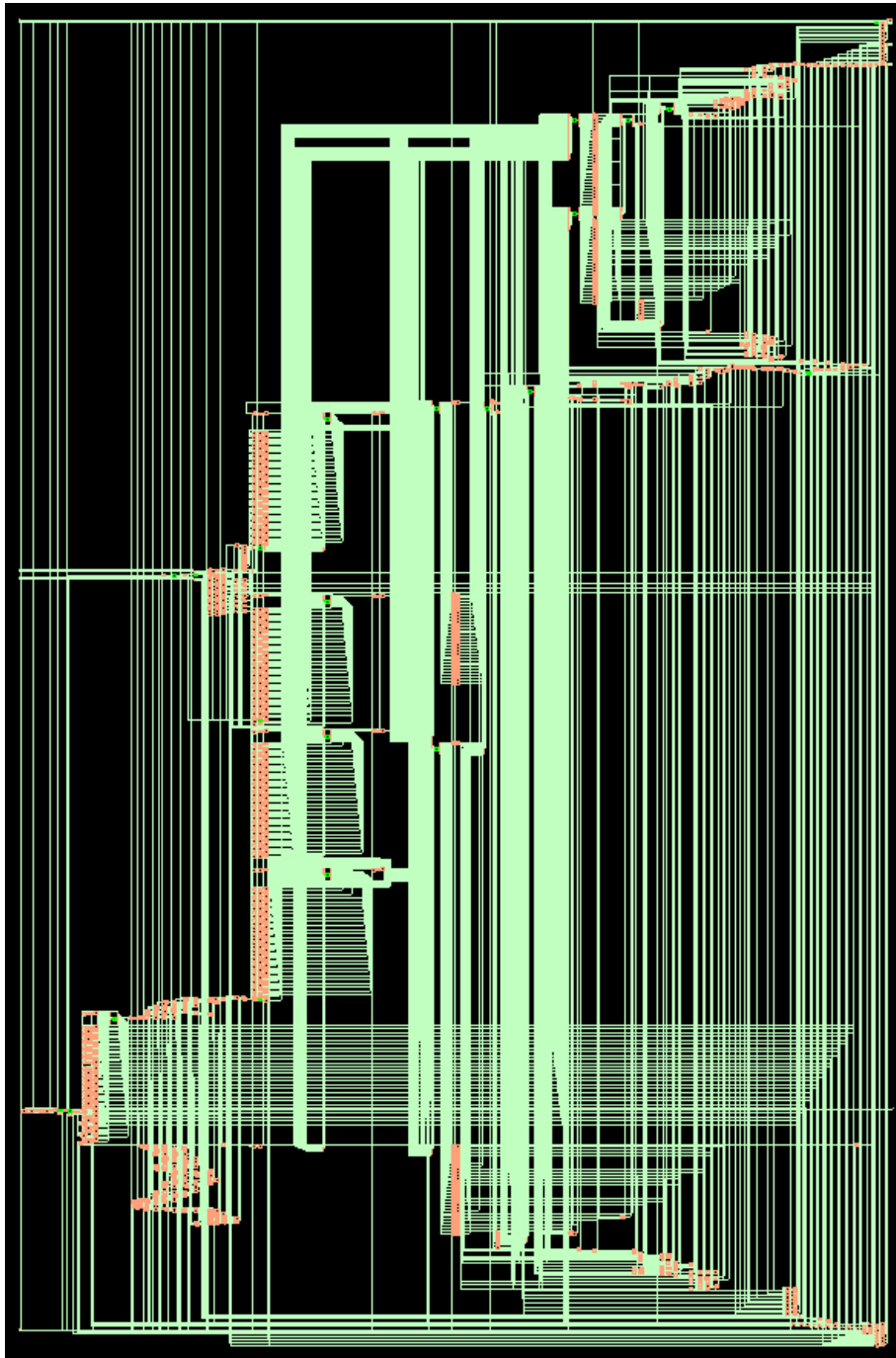


Fig 3. The Linear Regression model as synthesized into mapped gates

Final Synthesis reports:

Timing Report (Worst Slack):

Timing Report - (id: 1)							
Options	Endpoint: Yp_reg[0]/D						
Airlines	Endpoint	Slack (ps)	Rise Slew (ps)	Fall Slew (ps)			
	Yp_reg[0]/D	193.20	45.20	37.60			
Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)	
(clock CLK)	launch					0.00	R
retime_s3_14_reg/CP				0.00		0.00	R
retime_s3_14_reg/Q	DFOD1	2	2.80	34.70	111.70	111.70	R
inc_add_218_33_2/A[1]							
g1176/A1					0.00	111.70	
g1176/ZN	CKND2D1	3	3.40	46.00	38.30	150.00	F
g1174/B1					0.00	150.00	
g1174/ZN	INR2XD0	2	2.90	71.00	52.50	202.50	R
g1171/A1					0.00	202.50	
g1171/ZN	CKND2D1	3	3.40	49.50	47.30	249.80	F
g1169/B1					0.00	249.80	
g1169/ZN	INR2XD0	2	2.90	71.10	53.30	303.10	R
g1166/A1					0.00	303.10	
g1166/ZN	CKND2D1	3	3.30	48.70	46.70	349.80	F
g1195/A1					0.00	349.80	
g1195/ZN	IND2D1	3	3.40	49.40	75.90	425.70	F
g1161/B1					0.00	425.70	
g1161/ZN	INR2XD0	1	3.10	74.40	55.10	480.80	R
g1159/B					0.00	480.80	
g1159/CO	HA1D0	2	2.90	64.60	93.70	574.50	R
g1157/A1					0.00	574.50	
g1157/ZN	CKND2D1	3	3.30	47.70	45.10	619.60	F
g1194/A1					0.00	619.60	
g1194/ZN	IND2D1	3	3.30	48.40	75.00	694.60	F
g1193/A1					0.00	694.60	
g1193/ZN	IND2D1	3	3.30	48.40	75.20	769.80	F
g1192/A1					0.00	769.80	
g1192/ZN	IND2D1	3	3.30	48.40	75.20	845.00	F
g1191/A1					0.00	845.00	
g1191/ZN	IND2D1	3	3.30	48.40	75.20	920.20	F
g1190/A1					0.00	920.20	
g1190/ZN	IND2D1	3	3.30	48.40	75.20	995.40	F
g1189/A1					0.00	995.40	
g1189/ZN	IND2D1	3	3.30	48.40	75.20	1070.60	F
g1188/A1					0.00	1070.60	
g1188/ZN	IND2D1	3	3.30	48.40	75.20	1145.80	F
g1187/A1					0.00	1145.80	
g1187/ZN	IND2D1	3	3.30	48.40	75.20	1221.00	F
g1186/A1					0.00	1221.00	
g1186/ZN	IND2D1	3	3.30	48.40	75.20	1296.20	F
g1185/A1					0.00	1296.20	
g1185/ZN	IND2D1	3	3.30	48.40	75.20	1371.40	F
g1184/A1					0.00	1371.40	
g1184/ZN	IND2D1	3	3.30	48.40	75.20	1446.60	F
g1183/A1					0.00	1446.60	
g1183/ZN	IND2D1	3	3.30	48.40	75.20	1521.80	F
g1182/A1					0.00	1521.80	
g1182/ZN	IND2D1	3	3.30	48.40	75.20	1597.00	F

Fig 5. Timing report shows that the worst slack attained is 193 ps. The clock period is 3150 ps.

Therefore, the circuit runs at a Clock Frequency of **317.4 MHz**

[Area Report: \(Data path\)](#)

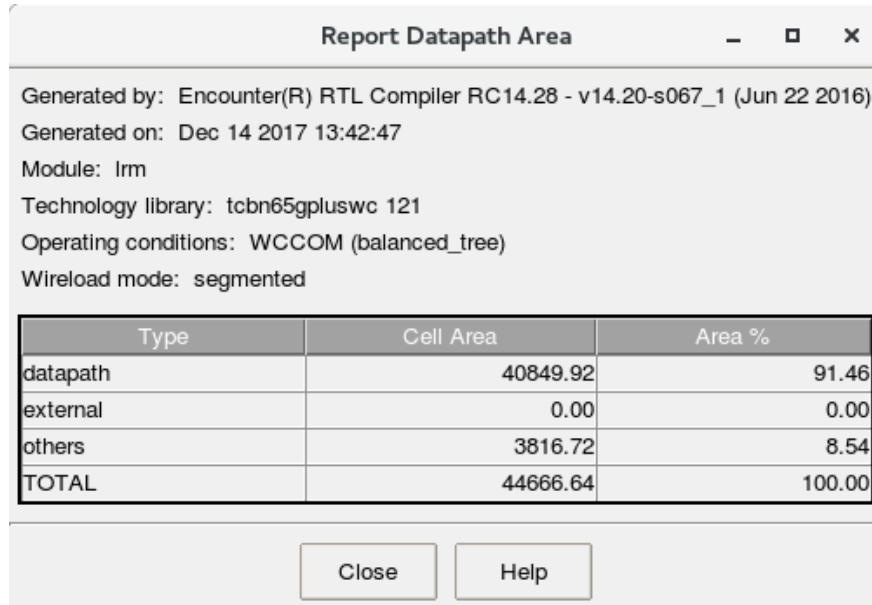


Fig 6. Area report showing Datapath and external areas

[Area report: \(Netlist statistics\)](#)

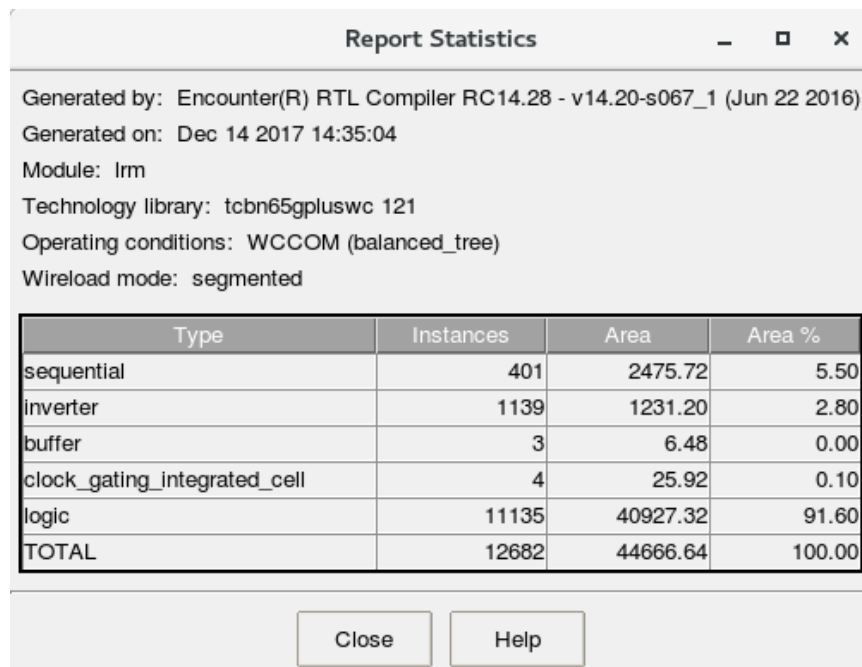


Fig 7. Netlist stat report

Power report: (Detailed)

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
lm	12682	358487.18	4817068.49	1403312.10	6220380.59
lm/add_66_11	68	2267.49	37117.29	8373.37	45490.67
lm/add_67_9	63	1832.35	21709.80	3444.82	25154.62
lm/add_68_9	63	1835.76	19632.88	3367.83	23000.71
lm/add_74_13	68	2266.99	36343.39	8133.96	44477.35
lm/Count1_out_reg[0].state_point	0	0.00	0.00	0.00	0.00
lm/Count1_out_reg[1].state_point	0	0.00	0.00	0.00	0.00
lm/csa_tree_R1_sub_12_18_group1	525	18442.05	1503.34	577.86	2081.21
lm/csa_tree_R2_sub_12_18_group1	525	19234.60	1876.85	438.54	2315.39
lm/csa_tree_sub_171_15_group1	1745	61971.95	1554574.17	445926.28	2000500.45
lm/csa_tree_sub_172_16_group1	1318	58869.65	1430551.30	311482.74	1742034.04
lm/csa_tree_sub_175_16_group1	1750	60962.29	894104.44	232209.27	1126313.71
lm/div_173_8	2181	32671.30	0.00	0.00	0.00
lm/div_177_9	2132	33351.99	0.00	0.00	0.00
lm/EX_reg[0].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[1].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[2].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[3].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[4].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[5].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[6].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[7].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[8].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[9].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[10].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[11].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[12].state_point	0	0.00	0.00	0.00	0.00
lm/EX_reg[13].state_point	0	0.00	0.00	0.00	0.00

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
lm	12682	358487.18	4817068.49	1403312.10	6220380.59

Fig 8. Power report (above) and total power for top module



Layout, Placement and Routing

The placement and routing of the synthesized design was carried out in Cadence Encounter environment where I have used the same library and .LEF provided by the Lab page and the mapped design and constraints that was provided by RTL Compiler. As expected, placement and routing tend to absorb a lot of the extra positive slack that is available at synthesis. Truly, the slack at the end of some of the Layout attempts came out to be negative and hence to keep the area and the performance good at the same times, some floorplan parameters were adjusted. I wrote a Place and Route script which was used by encounter to layout the design, entirely. This process entirely entails the following:

- **Synthesis of mapped RTL provided by RTL Compiler**
- **Specification of Floorplan**
- **Power Planning**
- **Standard Cell placement**
- **Pre-CTS Optimization**
- **Clock tree synthesis**
- **Post -CTS Optimization**
- **Detail Routing**
- **Post-Route Optimization**
- **Generation of Reports**

See Appendix C for layout script

Initially, following Floorplan values were used to perform PnR:

Floorplan Values:

FP Value Set	Core Utilization	Aspect Ratio	Row Spacing	I/O to Row
1	0.799994	0.9	1.8	20.0
2	0.699994	0.8	1.8	20.0
3	0.599994	0.8	1.8	20.0

Encounter Layout Report:

	Area (um ²)	Switching power(mW)	Worst Slack (ps)	Execution time (min:sec:ms)
<i>FP Values 1</i>	121799.67	8.122	-0.015	126:55:34
<i>FP Values 2</i>	131767.88	8.455	-0.002	145:35:02
<i>FP Values 3</i>	143224.00	8.788	0.021	155:50:01

I had the performance preference and therefore, I have chosen the tight clock period over area or power factors. But by tweaking the Floorplan parameters, I have tried to make good tradeoff between area and timing. The following section shows the Encounter PnR reports.

Encounter Reports:

Area report: Physical View:

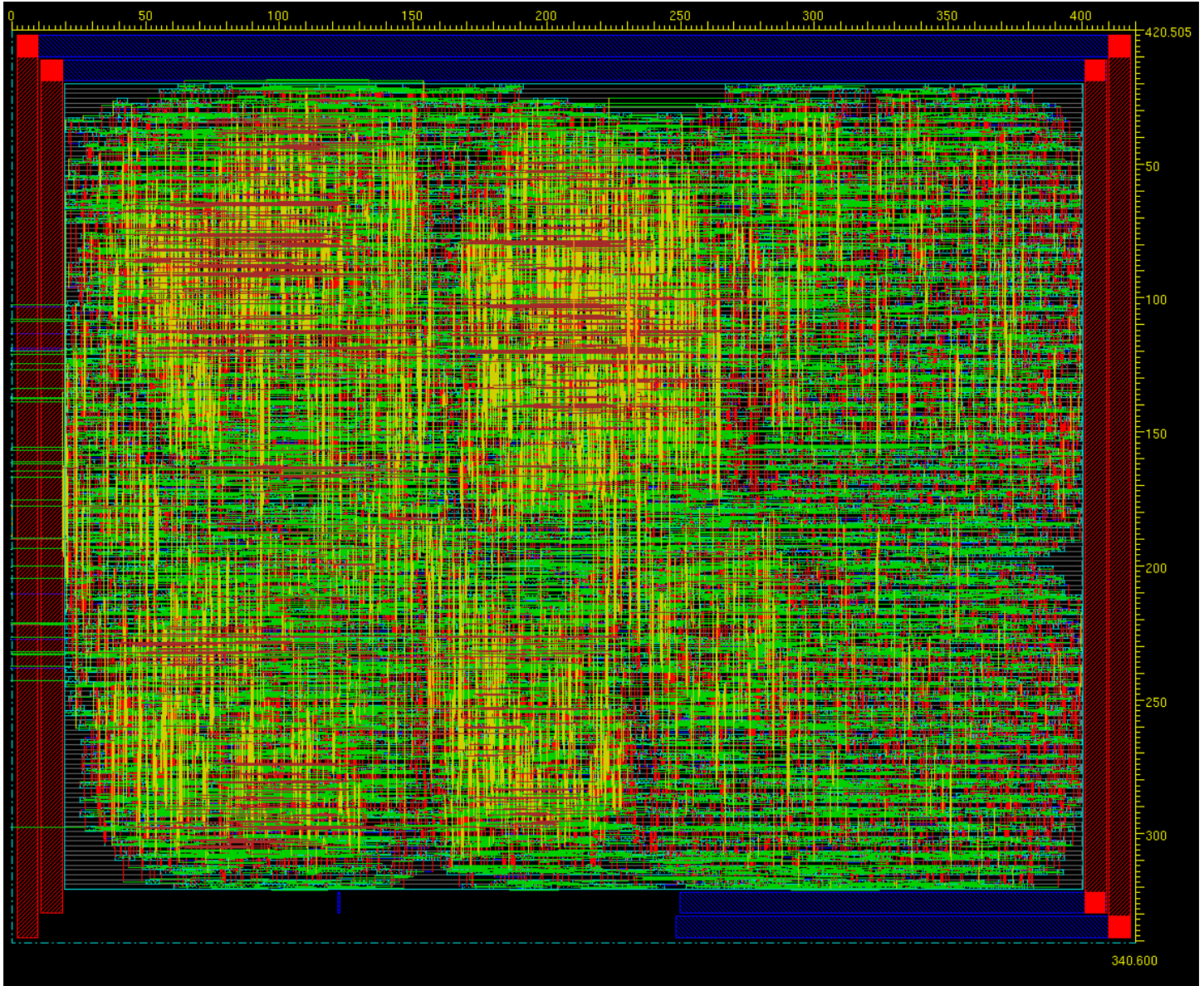


Fig 9. The final Placed and Routed, and optimized design showing die area

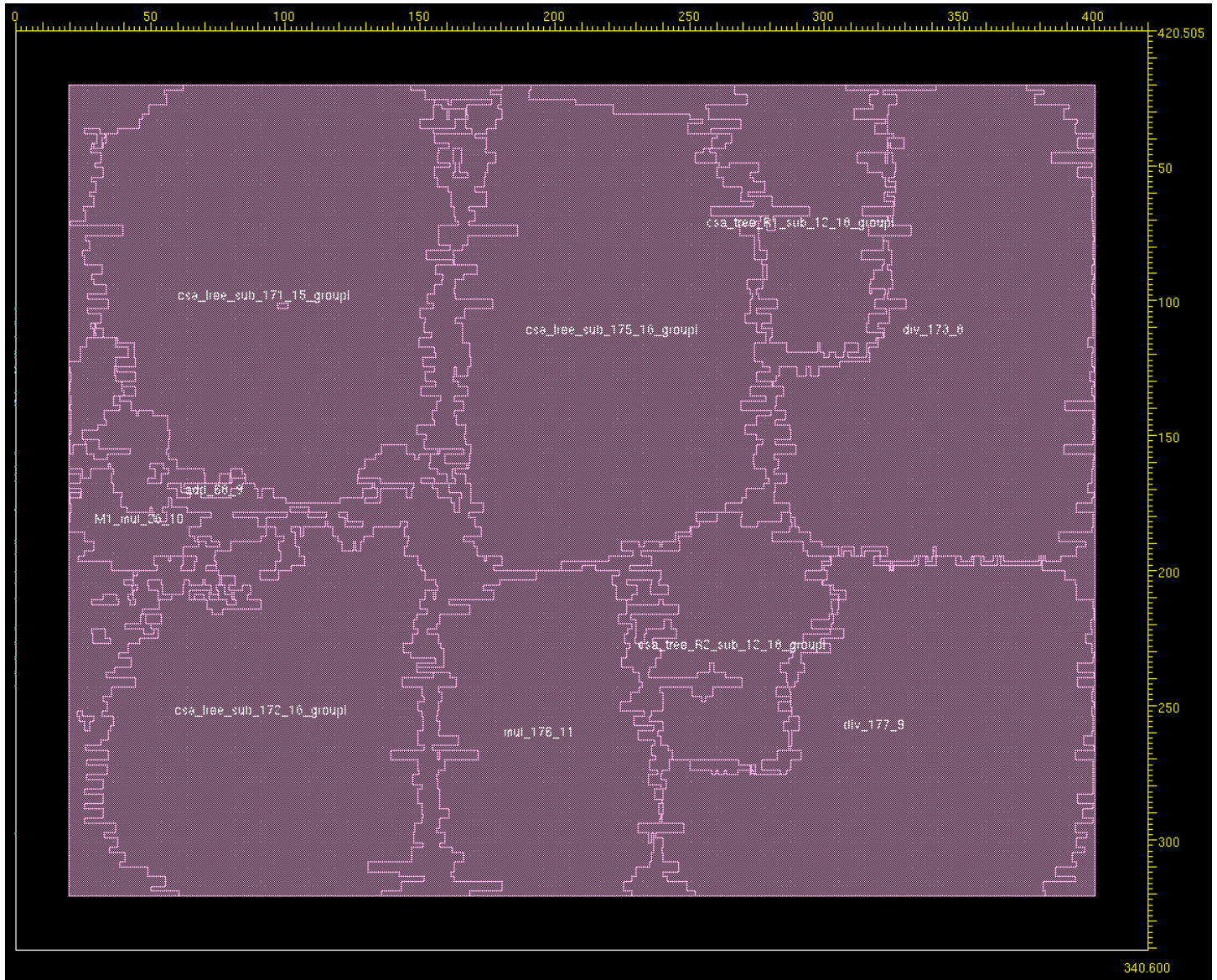


Fig 10. The Final design: Amoeba View

Power Report:

```

Finished Calculating power
2017-Dec-14 14:43:31 (2017-Dec-14 20:43:31 GMT)
-----
*      Encounter 10.12-s181_1 (64bit) 07/28/2011 22:52 (Linux 2.6)
*
*      Date & Time:    2017-Dec-14 14:43:31 (2017-Dec-14 20:43:31 GMT)
*
*-----
*      Design: lrm
*      Liberty Libraries used:
*      ./../encounter/libdir/lib/tcbn65gpluswc.lib
*
*      Power Domain used:
*
*      User-Defined Activity : N.A.
*
*      Activity File: N.A.
*
*      Hierarchical Global Activity: N.A.
*
*      Global Activity: N.A.
*
*      Sequential Element Activity: N.A.
*
*      Clock Gates Output Activity: N.A.
*
*      Clock Gates Enable Activity: N.A.
*
*      Primary Input Activity: 0.200000
*
*      Power Units = 1mW
*
*      Time Units = 1e-09 secs
*
*      report_power
*
*-----
Total Power
-----
Total Internal Power:    8.275    47.49%
Total Switching Power:   8.788    50.44%
Total Leakage Power:     0.3605   2.069%
Total Power:            17.42
-----

Group                  Internal Power    Switching Power    Leakage Power    Total Power    Percentage (%)
-----
Sequential              0.684          1.647            0.01931         2.35           13.49
Macro                   0              0                0               0              0
IO                      0              0                0               0              0
Combinational           7.508          7.008            0.3388          14.85          85.26
Clock (Combinational)   0.0731         0.1276           0.002223        0.2029         1.165
Clock (Sequential)      0.009649       0.005443         0.0001619       0.01525        0.08755
Total                   8.275          8.788            0.3605          17.42          100
-----

Rail                   Voltage    Internal Power    Switching Power    Leakage Power    Total Power    Percentage (%)
-----
Default                0.9       8.275            8.788            0.3605          17.42          100

Clock                  Internal Power    Switching Power    Leakage Power    Total Power    Percentage (%)
-----
CLK                    0.08275        0.1331           0.002385        0.2182         1.252
Total                  0.08275        0.1331           0.002385        0.2182         1.252
-----

*      Power Distribution Summary:
*      Highest Average Power:      rc_gclk_l1_i0 (CKBD24):    0.04931
*      Highest Leakage Power:      rc_gclk_l1_i0 (CKBD24):    0.0003385
*      Total Cap: 9.52392e-11 F
*      Total instances in design: 13541
*      Total instances in design with no power: 0
*      Total instances in design with no activity: 0
*
*      Total Fillers and Decap: 0
*
-----
report power consumed time (real time) 00:00:05 : peak memory (766.867M)

```

Fig 11. Encounter Power Report

Timing Report:

```
#####
# Generated by: Cadence Encounter 10.12-s181_1
# OS: Linux x86 64(Host ID linux-3.ece.iastate.edu)
# Generated on: Thu Dec 14 14:43:31 2017
# Design: lrm
# Command: report_timing
#####
```

Path 1: MET Setup Check with Pin Yp_reg[3]/CP
 Endpoint: Yp_reg[3]/D (^) checked with leading edge of 'CLK'
 Beginpoint: retime_s3_14_reg/Q (^) triggered by leading edge of 'CLK'
 Other End Arrival Time 0.213
 - Setup 0.037
 + Phase Shift 3.150
 = Required Time 3.326
 - Arrival Time 3.305
 = Slack Time 0.021
 Clock Rise Edge 0.000
 + Clock Network Latency (Prop) 0.187
 = Beginpoint Arrival Time 0.187

Instance	Arc	Cell	Delay	Arrival Time	Required Time
retime_s3_14_reg	CP ^			0.187	0.208
retime_s3_14_reg	CP ^ -> Q ^	DFQD1	0.134	0.321	0.342
inc_add_218_33_2/g719	A1 ^ -> ZN v	CKND2D1	0.053	0.375	0.395
inc_add_218_33_2/g729	B1 v -> ZN ^	INR2D2	0.049	0.423	0.444
inc_add_218_33_2/g714	B ^ -> CO ^	HA1D2	0.070	0.494	0.514
inc_add_218_33_2/g713	A1 ^ -> ZN v	ND2D2	0.033	0.527	0.548
inc_add_218_33_2/g710	B1 v -> ZN ^	INR2D2	0.038	0.565	0.586
inc_add_218_33_2/g708	A1 ^ -> ZN v	CKND2D1	0.053	0.618	0.639
inc_add_218_33_2/g705	B1 v -> ZN ^	INR2D2	0.061	0.679	0.700
inc_add_218_33_2/g703	A1 ^ -> ZN v	CKND2D2	0.041	0.720	0.741
inc_add_218_33_2/g700	B1 v -> ZN ^	INR2D2	0.042	0.762	0.783
inc_add_218_33_2/g698	A1 ^ -> ZN v	CKND2D2	0.034	0.797	0.818
inc_add_218_33_2/g695	B1 v -> ZN ^	INR2D2	0.043	0.840	0.861
inc_add_218_33_2/g693	A1 ^ -> ZN v	CKND2D2	0.036	0.876	0.897
inc_add_218_33_2/g690	B1 v -> ZN ^	INR2D2	0.046	0.922	0.943
inc_add_218_33_2/g688	A1 ^ -> ZN v	CKND2D2	0.042	0.964	0.985
inc_add_218_33_2/g685	B1 v -> ZN ^	INR2D2	0.046	1.010	1.031
inc_add_218_33_2/g683	A1 ^ -> ZN v	CKND2D2	0.044	1.054	1.075
inc_add_218_33_2/g680	B1 v -> ZN ^	INR2XD1	0.045	1.098	1.119
inc_add_218_33_2/g2	A2 ^ -> Z ^	AN2XD1	0.066	1.165	1.186
inc_add_218_33_2/g675	B ^ -> CO ^	HA1D0	0.097	1.262	1.283
inc_add_218_33_2/g674	B ^ -> CO ^	HA1D0	0.103	1.364	1.385
inc_add_218_33_2/g673	B ^ -> CO ^	HA1D0	0.101	1.465	1.486
inc_add_218_33_2/g672	B ^ -> CO ^	HA1D0	0.104	1.570	1.590
inc_add_218_33_2/g671	B ^ -> CO ^	HA1D0	0.107	1.676	1.697
inc_add_218_33_2/g670	B ^ -> CO ^	HA1D0	0.103	1.779	1.800
inc_add_218_33_2/g669	B ^ -> CO ^	HA1D0	0.104	1.883	1.904
inc_add_218_33_2/g668	B ^ -> CO ^	HA1D0	0.100	1.983	2.004
inc_add_218_33_2/g667	B ^ -> CO ^	HA1D0	0.113	2.096	2.117
inc_add_218_33_2/g666	B ^ -> CO ^	HA1D0	0.110	2.207	2.228
inc_add_218_33_2/g665	B ^ -> CO ^	HA1D0	0.101	2.307	2.328
inc_add_218_33_2/g664	B ^ -> CO ^	HA1D0	0.093	2.400	2.421
inc_add_218_33_2/g663	A1 ^ -> Z v	CKX0R2D1	0.095	2.495	2.516
g7182	A1 v -> Z v	AN2D2	0.098	2.593	2.614
g12636	A4 v -> Z v	OR4D1	0.108	2.701	2.722
FE_0FC52_n_258	I v -> Z v	CKBD1	0.236	2.937	2.958
g12635	I v -> ZN ^	INVD1	0.094	3.031	3.052
g12634	A1 ^ -> Z ^	AN3XD1	0.131	3.161	3.182
g12589	B2 ^ -> Z ^	A022D0	0.144	3.305	3.326
Yp_reg[3]	D ^	DFQD1	0.000	3.305	3.326

Fig 12. Encounter Timing Report showing 0.021 ns slack



Conclusions

Learning Experience

This project is a culmination of all the processes that are actively used in the industry throughout the VLSI Design flow. As such, I am a first timer in the tools that are being used. My experience with RTL Level coding is also not exemplary. But the amount of learning in this project and the associated labs, is immense. There can be several ways of tackling a problem. I started the coding process in a structural manner and then I realized that a behavioral style of coding can be much more suitable for this application.

I have learnt a lot of debugging in the course of this project. ModelSim has a very clear wave form display which can be used very effectively to debug any kind of problems in the design. This has helped me a lot in this project.

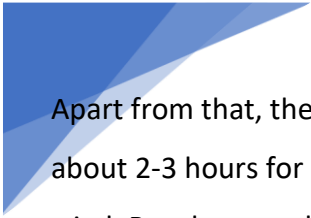
Many constructs in an HDL code are not synthesizable. For example, using real types could have saved us the trouble of rounding since precision would be perfect. But real numbers aren't synthesizable. Other examples include, floating point numbers, for loops etc. Also, some blocks might lead to functional mismatch of code and synthesized circuit. These should be avoided. I learnt about the combinational loops and sensitivity lists and how they can affect synthesizability.

Overall, the learning experience from this project was amazing and it will surely help me a lot in the future.

Difficulties

As my experience with Verilog and Cadence tools is not the best, and certainly not exemplary, the design of this core was full of challenges. The first challenge was to debug the code. When I was testing the design on the previous test bench, I had to tweak my formulas a number of times and to debug and recognize the bug was a challenge. Thankfully, in the revised test bench, my formulas quickly led to successful results.

Unfortunately, I am submitting this project report alone as there was absolutely zero work done by the student who was supposed to be my teammate. That is also why I have used the word "I" throughout the report. Also, I include only my own name in the report for the same reason. It is an extremely interesting project and I am glad I was a part of this. But at times, it was very difficult to manage the entire work alone. I am confident I could have provided double the quality of this work had it been a team project.



Apart from that, the time taken by Cadence tools to synthesize and especially, Place and Route the design was about 2-3 hours for each attempt. This made it very time consuming if more and more variations were to be tried. But the overall experience from the labs have been good and this project and labs have motivated me a lot into using these tools. I have successfully used these tools in my other term projects as well!

Feedback

The project, on the whole, is very suitable for a course on Digital VLSI Design and it has further implications on other futuristic fields as well. So, the choice of topic was very suitable. Also, for someone who is willing to learn VLSI Design from scratch, this project can be a handy way to develop interest in the subject. But that person should be prepared to spend more time with this.

I sincerely look forward to polish this code in the upcoming holidays and subsequently synthesize and lay it out to see how much better this core can be. Other interesting future work can include Designing of a hybrid processor with modules for Linear Regression, Logistic Regression, Support Vector and other basic machine learning algorithms and a controller and I/O peripherals. I have done a similar project with DSP application so this idea really attracts me!



Appendix A

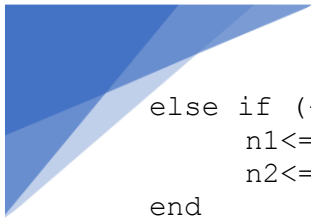
Verilog code for Linear Regression Module:

```
`timescale 1ns/100ps
module lrm(
    input RESET,
    input [7:0] Xi,
    input [7:0] Yi,
    input CLK,
    input PREDICT,
    output reg READY,
    output reg DONE,
    output reg [7:0] A,
    output reg [7:0] B,
    output reg [7:0] Yp
);

reg [31:0] N,N1;
reg [7:0] n1,n2;
reg nzero,nfirst;
reg [31:0] EXY,EX,EY,EXsq;
wire [15:0] out1;
wire [15:0] product1;
reg [15:0] O1;
reg [7:0] X,Y;
reg EN;
reg [7:0] nA,nB,nYp;
wire [7:0] An,Bn;
reg [31:0] num,num2,den,den2;
reg [31:0] GatedNumerator,GatedNumerator2,GatedDenominator,GatedDenominator2;
reg [31:0] rem,halfden,rem2,halfden2;

always @(*) begin
    //Enable for upcounter in READY Logic
    EN<=(~RESET);
end

always @(posedge CLK) begin
    //Sampling inputs at clock edge
    if(RESET) begin
        n1<=0;
        n2<=0;
    end
end
```




```
else if (~PREDICT && READY) begin
    n1<=Xi;
    n2<=Yi;
end
end

always @(posedge CLK) begin
    if(RESET) N1<=0;
    //Last edge versions of N: to detect change in N
else N1<=N;
end

always @(posedge CLK) begin
    //Logic for calculating coefficients
    if(RESET) begin
        EXY<=0;
        EX<=0;
        EY<=0;
        EXsq<=0;
    end
    else if (N1!=N)begin
        if(nzero) begin
            EXY<=EXY+product1;
            EX<=EX+(n1);
            EY<=EY+(n2);
            EXsq<=EXsq;
        end
    end

    else if(nfirst) begin
        EXsq<=EXsq+product1;
        // 'product1' used to
        // compute both n1*n1 and n1*n2:
        // Hardware reuse performed for Area optimization
    end
end

always @(posedge CLK) begin
    //Following three always block generate control logic for MUX select in HW
reuse
    if(RESET) nzero<=0;
    else if(READY && ~PREDICT) nzero<=1;
    else nzero<=0;
end
always @(posedge CLK) begin
    nfirst<=nzero;
end
```



```
Multiplier M1(.n1(n1),.n2(n2),.select1(nzero),.select2(nfirst),.out(product1));  
//Multiplier module: Calculates simple product
```

```
always @(posedge CLK) begin  
    //DONE Logic  
    if(RESET) DONE<=0;  
    else if(PREDICT) DONE<=1;  
    else DONE<=0;  
end
```

```
UpCounter Count1(.out(out1),.enable(EN),.clk(CLK),.reset(RESET));  
//16 bit Up counter: put any value after modulus (%) to flexibly change the  
frequency of READY signal  
always @(*) begin  
    O1<=out1%4;  
end
```

```
always @(posedge CLK) begin  
    //READY Logic  
    if(O1==0)begin  
        READY<=1;  
    end  
    else READY<=0;  
end
```

```
always @(posedge CLK) begin  
    //Outputs A and B: Calculated from Dummy Outputs nA and nB  
    if(PREDICT) begin  
        if(N==0 || N==1) begin  
            A<=0;  
            B<=0;  
        end  
        else begin  
            A<=nA;  
            B<=nB;  
        end  
    end  
end
```



```

always @(posedge CLK) begin
    //Outputs Yp: Calculated from Dummy outputs nYp
    if(PREDICT) begin
        if(N==0 || N==1) begin
            Yp<=nYp;
        end
    else begin
        Yp<=(nYp);
    end
end
end

Rounder R1(.X(X),.num(num),.den(den),.nA(An));
    //Rounder: implements rounding logic. returns ceiling if fractional part >=
0.5 otherwise floor
Rounder R2(.X(Y),.num(num2),.den(den2),.nA(Bn));


always @(*) begin
    if(RESET)begin
        nA<=0;
        nB<=0;
    end
else begin
    nA<=An;
    nB<=Bn;
end
end

always @(*) begin
//Dummy Outputs for A and B
    if(RESET) begin
        num<=0;
        den<=0;
    end
    else if(PREDICT) begin
        num<=((N*EXY)-(EX*EY));
        den<=((N*EXsq)-(EX*EX));
        X<=num/den;
//1 division

        num2<=(den*EY)-(num*EX);
        den2<=den*N;
        Y<=num2/den2;
//1 division
    end
end

/*
always @(*) begin
    section: will disallow division when predict is not asserted and hence saves
power.
    if (PREDICT) begin

```



```

    GatedNumerator <= num;
    GatedDenominator <= den;
    GatedNumerator2 <= num2;
    GatedDenominator2 <= den2;
end
else begin
    GatedNumerator <= 0;
    GatedDenominator <= 0;
    GatedNumerator2 <= 0;
    GatedDenominator2 <= 0;
end
end
*/

always @(*) begin
    //Dummy Outputs for Yp
    if(RESET) begin
        nYp<=0;
    end
    else if(N==0) begin
        nYp<=8'b10000000;
    end
    else if(N==1) begin
        nYp<=n2;
    end
    else begin
        nYp<=nA*Xi+nB;
        //Using rounded A and B values for Yp: NO division
    end
end

always @(posedge CLK) begin                                //N-
Logic
    if (RESET)N<=0;
    else if(READY && ~PREDICT) N<=N+1;
    else N<=N;
end

endmodule

```

Verilog code for Multiplier:

```
`timescale 1ns/100ps
module Multiplier(
    input [7:0] n1,
    input [7:0] n2,
    input select1,
    input select2,
    output reg [15:0] out
);

    reg[7:0] x,y;

    always @(*) begin
        if(select1==1 && select2==0) begin
            x=n1;
            y=n2;
        end

        else if(select2==1 && select1==0) begin
            x=n1;
            y=n1;
        end
    end

    always @(*) begin
        out=x*y;
    end

endmodule
```

Verilog code for Rounder:

```
`timescale 1ns/100ps
module Rounder(
    input [7:0] X,
    input [31:0] num,
    input [31:0] den,
    output reg [7:0] nA
);

    reg [31:0] remainder, halfden;

    always@(*) begin
        remainder=num-(den*X);
        halfden=(den>>1);
    end

    always@(*) begin
```

```

        if(remainder>halfden) nA<=X+1;
        else nA<=X;
    end

```

```
endmodule
```

Verilog code for Up Counter:

```

`timescale 1ns/100ps
module UpCounter(                                     //Upcounter for
READY-Logic
    out,
    enable,
    clk,
    reset
);

output reg [15:0] out;
input enable,reset,clk;

always @(posedge clk)
if (reset) begin
    out <= 8'b0 ;
end
else if (enable) begin
    out <= out + 1;
end
endmodule

```

Test bench: (uses revised data)

```

`timescale 1ns/100ps
`define INFILE1      "data/input_3_binary"
`define INFILE2      "data/output_3_binary"
`define IN_LENGTH    60
`define OUT_LENGTH   6

module lrm_tb;
parameter period = 20;

reg clk;
reg rst_hi;
reg predict;
reg [7:0] xi;
reg [7:0] yi;
wire ready;
wire done;

```

```

wire [7:0] yp;
wire [7:0] out_a;
wire [7:0] out_b;

integer i, k, l, err, cnt, total_cycle_num;
reg [16:0] data_base [0:`IN_LENGTH - 1];
reg [23:0] data_base_expect [0:`OUT_LENGTH - 1];
reg [16:0] data_tmp_i;
reg [23:0] data_tmp_expect;

initial $readmemb(`INFILE1, data_base);
initial $readmemb(`INFILE2, data_base_expect);

initial clk = 1'b0;
always #(period/2) clk = ~clk;

initial begin
    i = 0;
    k = 0;
    l = 0;
    err = 0;
    cnt = 0;
end

initial begin
    rst_hi = 1'b1;
    #(period)
    rst_hi = 1'b0;

    $display ("\n***** START to VERIFY the Linear Regresssion Module OPERATION
*****");
    for(i = 0; i < `IN_LENGTH; i = i + k) begin
        @(negedge clk)
            if(ready == 1'b1) begin
                data_tmp_i = data_base[i];
                xi = data_tmp_i[16:9];
                yi = data_tmp_i[8:1];
                predict = data_tmp_i[0];
                if (predict == 1'b0) begin
                    cnt = cnt + 1;
                end
                k = 1;
                $display("<Xi,Yi,PREDICT> = <%d,%d,%d>",xi,yi,predict);
            end else begin
                xi = 8'bz;
                yi = 8'bz;
                predict = 1'bz;
                k = 0;
            end
        end
    end
end

always @(posedge clk) begin

```

```

#(period*0.1)
if (done == 1'b1) begin
    data_tmp_expect = data_base_expect[1];
    if (cnt < 2) begin
        if (yp != data_tmp_expect[23:16]) begin
            $display("ERROR at %d: Yp=%d != expect Yp=%d (A and B are don't care so
far)",
                1, yp, data_tmp_expect[23:16]);
            err = err + 1 ;
        end
    end else begin
        if ((yp != data_tmp_expect[23:16]) ||
            (out_a != data_tmp_expect[15:8]) ||
            (out_b != data_tmp_expect[7:0])) begin

            $display("ERROR at %d: Yp=%d A=%d B=%d != expect Yp=%d A=%d B=%d",
                1, yp, out_a, out_b, data_tmp_expect[23:16], data_tmp_expect[15:8],
data_tmp_expect[7:0]);
            err = err + 1 ;
        end
    end
    l = l + 1;
end

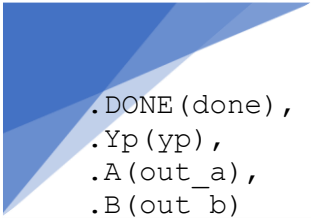
if( l == `OUT_LENGTH ) begin
    if (err == 0)
        $display("PASS! All results are correct!");
    else begin
        $display("-----");
        $display("There are %d errors!", err);
        $display("-----");
    end
    $display("-----");
    $display("Total number of cycles: %d", total_cycle_num);
    $display("-----");
    $finish;
end

end

always @(posedge clk) begin
    if (rst_hi == 1'b1)
        total_cycle_num = 0 ;
    else
        total_cycle_num = total_cycle_num + 1 ;
end

lrm lrm_0 (
    .CLK(clk),
    .RESET(rst_hi),
    .PREDICT(predict),
    .Yi(yi),
    .Xi(xi),
    .READY(ready),

```

A decorative graphic in the top-left corner consisting of several overlapping triangles in various shades of blue, creating a dynamic, abstract shape.

```
.DONE(done),  
.Yp(yp),  
.A(out_a),  
.B(out_b)  
);
```

```
endmodule
```



Appendix B

Script for synthesis:

```
## This sets the name of the directory in which area/timing/power reports
## and synthesized (mapped) netlists are stored.
set OUTPUT_DIR ./run_dir
if { ![file exists ${OUTPUT_DIR}] } { sh mkdir ${OUTPUT_DIR} }

#### Step 1 ####
## This tells the compiler where to look for the libraries
set_attribute lib_search_path ../libdir

## This defines the libraries to use
set_attribute library {tcbn65gpluswc.lib}
##set_attribute library {tcbn65gplustc.lib}
##set_attribute library {tcbn90ghpbc_ccs.lib}
##set_attribute lp_insert_clock_gating true

#set_attribute lp_insert_operand_isolation true

#### Step 2 ####
##This must point to your VHDL/verilog file
source ./scripts/read_rtl.tcl

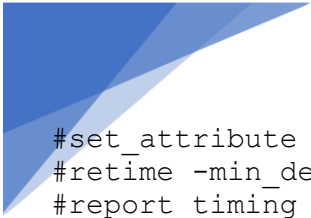
set_attribute lp_insert_clock_gating true

#### Step 3 ####
## This builds the general block
elaborate

set_attribute retime true /designs/lrm
#set_attribute dft_scan_style {muxed_scan|clocked_lssd_scan}
#define_dft test_mode test_mode_signal -active high
#define_dft shift_enable shift_enable_signal -active high
#check_dft_rules

#### Step 4 ####
## This allows you to define a clock and the maximum allowable delays
## READ MORE ABOUT THIS SO THAT YOU CAN PROPERLY CREATE A TIMING FILE
#set clock [define_clock -period 300 -name clk]
#external delay -input 300 -edge rise clk
#external delay -output 2000 -edge rise p1
read_sdc ./scripts/design.sdc

#### Step 5 ####
##This synthesizes your code
synthesize -to_mapped -effort high
```

```
#set_attribute unmap_scan_flops true
#retime -min_delay
#report timing
#replace_scan
#connect_scan_chain -auto_create
#report dft_chains
#report clock_gating
#report dft_setup
#synthesize -to_mapped -incremental
#report timing
```

```
#### Step 6 ####
## This writes area/timing/power reports
report area > ${OUTPUT_DIR}/area.rpt
report gates > ${OUTPUT_DIR}/gates.rpt
report timing > ${OUTPUT_DIR}/timing.rpt
report timing -lint > ${OUTPUT_DIR}/lint.rpt
report summary > ${OUTPUT_DIR}/summary.rpt
report power > ${OUTPUT_DIR}/power.rpt
```

```
#### Step 7 ####
## This writes synthesized design files
write_sdc > ${OUTPUT_DIR}/design_mapped.sdc
write -mapped > ${OUTPUT_DIR}/design_mapped.v
write_script > ${OUTPUT_DIR}/design_mapped.g
```





Appendix C

Script for Layout:

```
#####
#
#   Encounter Command Logging File
#   Created on Fri Sep 29 23:22:33 2017
#
#####

#@(#)CDS: Encounter v10.12-s181_1 (64bit) 07/28/2011 22:52 (Linux 2.6)
#@(#)CDS: NanoRoute v10.12-s010 NR110720-1815/10_10_USR2-UB (database version
2.30, 124.2.1) {superthreading v1.15}
#@(#)CDS: CeltIC v10.12-s013_1 (64bit) 07/27/2011 04:14:35 (Linux 2.6.9-
89.0.19.ELsmp)
#@(#)CDS: AAE 10.12-s001 (64bit) 07/28/2011 (Linux 2.6.9-89.0.19.ELsmp)
#@(#)CDS: CTE 10.12-s010_1 (64bit) Jul 18 2011 22:58:43 (Linux 2.6.9-
89.0.19.ELsmp)
#@(#)CDS: CPE v10.12-s007

# Open the GUI window
win

# Synthesize and then place the design (step 4 of tutorial)
compileDesign
placeDesign -noPrePlaceOpt

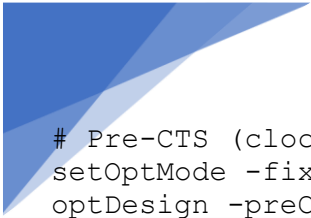
# Select different views
setDrawView place
setDrawView ameba

# Fit the design in the window
fit

# Set floorplan parameters and row spacing (step 5 of tutorial)
getIoFlowFlag
setFPlanRowSpacingAndType 1.8 2
setIoFlowFlag 0
floorPlan -site core -r 0.8 0.599994 20.0 20.0 20.0 20.0

# Add power rings (step 6 of tutorial)
addRing -spacing_bottom 1 -width_left 8 -width_bottom 8 -width_top 8 -spacing_top
1 -layer_bottom M1 -stacked_via_top_layer M8 -width_right 8 -around core -
jog_distance 0.1 -offset_bottom 1 -layer_top M1 -threshold 0.1 -offset_left 1 -
spacing_right 1 -spacing_left 1 -offset_right 1 -offset_top 1 -layer_right M2 -
nets {VDD VSS} -stacked_via_bottom_layer M1 -layer_left M2

# Place the design into the rows (step 7 of tutorial)
setPlaceMode -fp false
placeDesign -prePlaceOpt
```



```
# Pre-CTS (clock tree synthesis) optimization (step 8 of tutorial)
setOptMode -fixCap true -fixTran true -fixFanoutLoad false
optDesign -preCTS

# CTS (step 9 of tutorial)
clockDesign -genSpecOnly Clock.ctstch
clockDesign -specFile Clock.ctstch -outDir clock_report -fixedInstBeforeCTS
displayClockPhaseDelay -preRoute

# Post-CTS optimization (step 10 of tutorial)
setOptMode -fixCap true -fixTran true -fixFanoutLoad false
optDesign -postCTS

# Route wires in the design (step 11 of tutorial)
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven false
setNanoRouteMode -quiet -routeWithSiDriven false
routeDesign -globalDetail

# Post-route optimization (step 12 of tutorial)
setOptMode -fixCap true -fixTran true -fixFanoutLoad false
optDesign -postRoute

setDrawView place
fit

report_power
report_timing
```



Thank You