ELEC 5804: VLSI Design

Prof. Leonard MacEachern

Final project:
FIR Highpass filter: Simulation and Synthesis

Fall 2013

Sergio Espinal 100926721

Carleton University

December 31, 2013

Introduction

The project is about the synthesis of a FIR highpass filter, from the verilog code to the circuit layout conception.

The filter is used by an electromyography data acquisition device. The EMG device (for short) attaches to a part of the human body (like the arm or leg) and measures electric muscle impulses.

Noise is generated by the friction of the electrodes and the skin due to displacement when the muscle contracts or when the body is moving around.

The FIR highpass filter will eliminate that noise.

Background theory

"A *continuous-time signal* is a function of an independent variable that is continuous. It can be a real- or complex- valued function of time. Since the function is well defined for all values of time in $-\infty$ and ∞, it is differentiable at all values of the independent variable t.

A *discrete-time signal* is a function that is defined only at discrete instants of time and undefined at all other values of time. The function is defined at equal intervals of time and defined at $t = nT$, where T is a fixed interval in seconds known as the sampling period and n is an integer variable defined over $-\infty$ to ∞.

When the value of a sample is held constant during a period in T (or a fraction of T) by a zero-order hold circuit as its output, that signal can be converted to a value by a quantizer circuit, with finite levels of value as determined by the binary form of representation. Such process is called *binary coding*."[1]

Normally, the values are express using *signed magnitude fixed-point* binary number representation. The most significant bit is used to represent the sign (positive or negative) and the rest of the bits represent the value as a fraction number.

The finite quantized samples n from the discrete-time function $f(n)$ can be described as elements of a set, $x(n)$.

The length of a DT sequence depends on the order of the polynomial in z^{-1} that represents the function.

Filtering is the most common form of signal processing to remove the frequencies in certain parts and to improve the magnitude, phase, or group delay in some other part(s) of the spectrum of the signal.

Shannon's sampling theorem proved that if a continuous-time signal is bandlimited (i.e. if its Fourier transform is zero for frequencies above a maximum frequency f_m) and it is sampled at a rate that is

more than twice the maximum frequency f_m in the signal, then no information contained in the analog signal is lost, being able to be reconstructed from the samples of the discrete-time signal.

A discrete-time system is a mapping of the set of discrete-time signals considered as the input, to another set of discrete-time signals identified as the output of the system. This system can be composed by three basic components: delay elements, multipliers and adders."

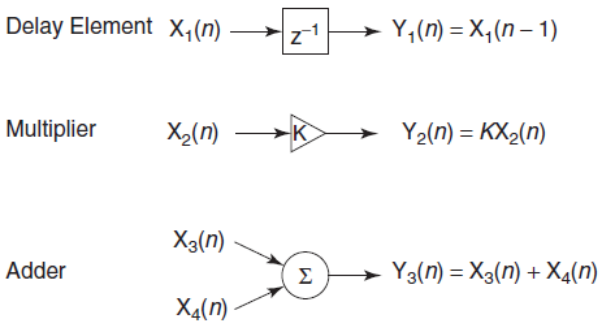


Figure 1. Representation of basic components of a discrete-time system.

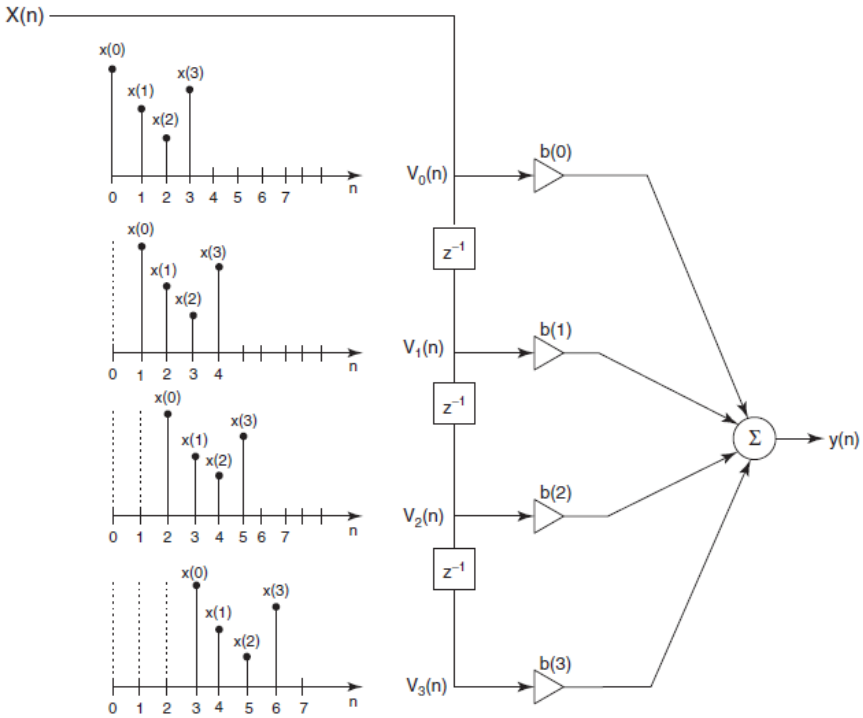


Figure 2. Example of a discrete-time system [1].

The general form of the difference equation for a linear, time invariant, discrete-time system is:

$$y(n) = -\sum_{k=1}^{N} a(k)y(n-k) + \sum_{k=0}^{M} b(k)x(n-k) \qquad (E.1)$$

The FIR filter

$$y(n) = \sum_{k=0}^{M} b(k)x(n-k) \qquad (E.2)$$

$$y(n) = b(0)x(n) + b(1)x(n-1) + \ldots + b(M)x(n-M) \qquad (E.3)$$

The system modeled by the equation (E.2) has a unit impulse response that is finite in length (M samples). This is known as the finite impulse response (FIR) filter where the samples of the unit impulse response $h(n) = b(n)$.

The transfer function is:

$$H(z^{-1}) = b_0 + b(1)z^{-1} + b(2)z^{-2} + \cdots + b(M)z^{-M} \qquad (E.4)$$

$$H(z^{-1}) = \sum_{n=0}^{N} h(n+1)z^{-n} \qquad (E.5)$$

When the input function $x(n)$ is the unit sample function $\delta(n)$, the output $y(n)$ can be obtained by applying the recursive algorithm on equation (E.2).

Some advantages of the FIR filters:

- An FIR filter can be designed to meet the required magnitude response, such that it achieves a constant group delay. The phase response of a filter with a constant group delay is therefore a linear function of frequency. It transmits all frequencies with the same amount of delay, with no phase distortion, and the input signal will be delayed by a constant when it is transmitted to the output.
- The samples of its unit impulse response are the same as the coefficients of the transfer function.
- The FIR filters are always stable and are free from limit cycles.

“The output sequence is calculated as a linear combination of the current and M pas input samples. The input data pass through an M-stage shift register. After each shift operation, the outputs of the shift register are weighted with the coefficients and summed up. The sum is the output value of the filter associated with the respective clock cycle. The weighting coefficients determine the characteristics of the filter.”[2]

How the emgFilter.v file works

The “*emgFilter.v*” file contains the core of the filter. The structure of the filter is the same as **Figure 2**. The filter has 143 coefficients, 16 bits each. The input signal propagates from one delay component to the next. Example:

```
if (clk_enable == 1'b1) begin
          delay_pipeline[0] <= filter_in;
          delay_pipeline[1] <= delay_pipeline[0];
          delay_pipeline[2] <= delay_pipeline[1];
…
end
```

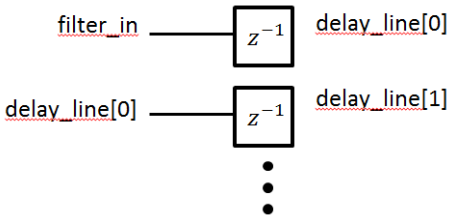


Figure 3. Block diagram of the filter_in data propagating through the delay_line.

A *delay_line_(n)* register is multiplied with a *coefficient_(n+1)* parameter. The result is “saved” in a temporary wire called *mult_temp_(0)* and the result is sent to a wire array called *product_(n+1)*.

Example:

```
assign mul_temp = delay_pipeline[142] * coeff143;
assign product143 = mul_temp[30:0];

assign mul_temp_1 = delay_pipeline[141] * coeff142;
assign product142 = mul_temp_1[30:0];
```

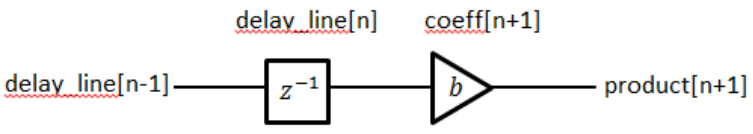


Figure 4. Block diagram of the delay_line and the coeff parameter.

The product of each multiplication (through the product wire array) is then added with the result of the previous summing stage. Example:

```
assign add_signext_280 = sum140;
assign add_signext_281 = $signed({{2{product142[30]}}, product142});
assign add_temp_140 = add_signext_280 + add_signext_281;
assign sum141 = add_temp_140[32:0];

assign add_signext_282 = sum141;
assign add_signext_283 = $signed({{2{product143[30]}}, product143});
assign add_temp_141 = add_signext_282 + add_signext_283;
assign sum142 = add_temp_141[32:0];
```

The final sum is then the output of the filter, saved in the *output_register* and sent through *filter_out* wire. Example:

```
if (clk_enable == 1'b1) begin
       output_register <= sum142;
end
```

Simulation, Synthesis and Layout Generation

In order to simulate the filter, we can use a HDL developer/simulator program. In this case we are going to use *ModelSim*.

The program will simulate the verilog code file of the filter core using a test-bench, which is another verilog coded file that contains the input signals or stimulus for the filter.

The simulation will provide a plot of the input and output as waveforms. This will give us an idea of what the behavior of the filter and its type. But the program does not provide information about the frequency response of the filter.

It is possible to view the frequency response of the filter using a numerical-computing programming software like *MATLAB*. Applying functions as the Fast-Fourier Transformation, we can plot the frequency response based on the sampled data from the waveform.

After this, we proceed to the synthesis of the verilog code file into a circuit layout using *Design Vision*.

We load the filter core file on *Design Vision*, make some settings and we proceed to generate a synthesized circuit, saved as a verilog code file.

This synthesized circuit is employed in the software called *Cadence Encounter* to generate a circuit layout.

Note: These design and simulation programs require component libraries. Setting the libraries for each program is not covered on this project for practical reasons.

Here are the steps on how to complete the simulation, synthesis and layout generation.

Simulation

ModelSim ALTERA Starter Edition 6.6c

Run *ModelSim* and start a new project. Go to *File -> New -> Project*…

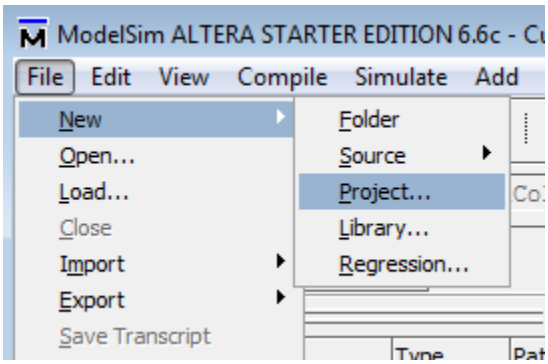


Figure 5. Starting a new project in ModelSim

At the *Create Project* screen, write a name for the project, set the project location and the name of the library you want it to be. Then click OK. Next, add the "core" and the "test-bench" files of the filter. On the Add items to the Project window, select Add Existing File and add both "*emgFilter.v*" and "*emgFilter_tb.v*" files. As an option, mark the "*Copy to project directory*" radio button.

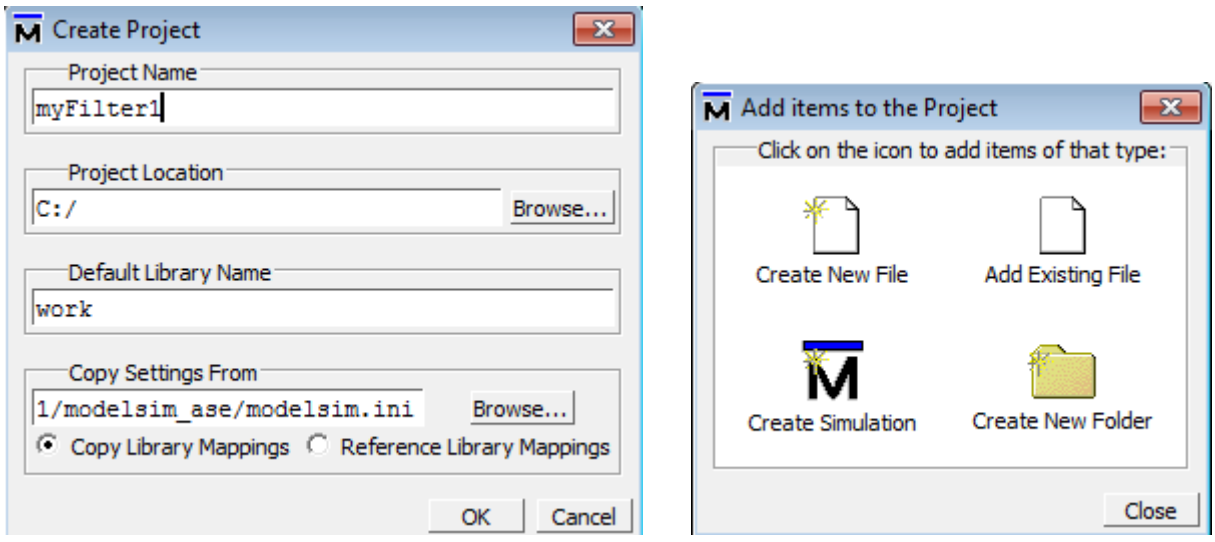


Figure 6. To the left is the *Create Project* screen. After hitting OK, the *Add items to the Project* screen appears.

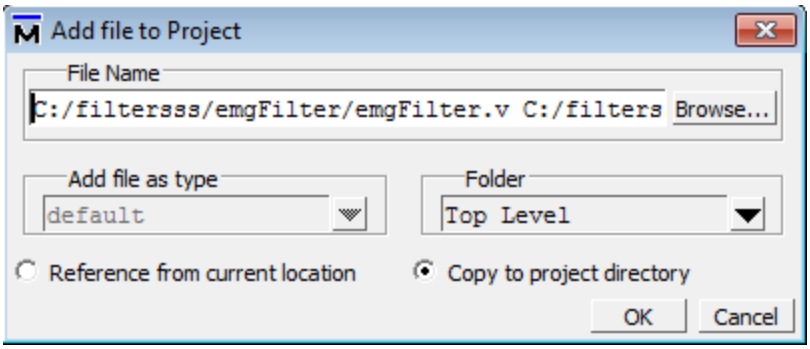


Figure 7. The Add file to Project screen.

The files have to be compiled. Go to *Compile -> Compile All*. To start a simulation, go to *Simulate -> Start Simulation* and locate the test -bench file. Unmark the checkbox "*Enable optimization*". Then click OK.

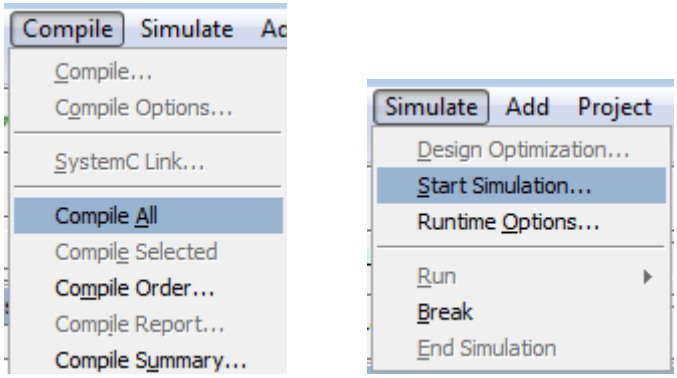


Figure 8. Compile and Simulation commands.

On the Objects window, find the "*filter_in*" and "*filter_out*" instances. Select both, right-click on them and go to *Add -> To Wave -> Selected Signals*. Add some simulation time and then run the simulation.

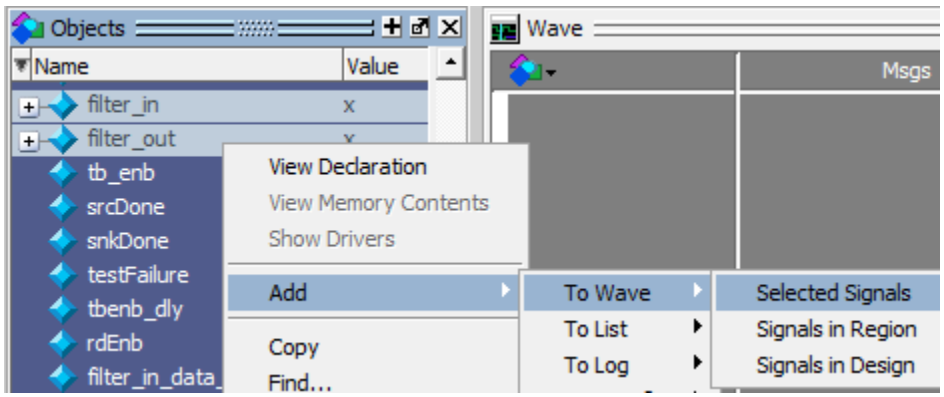


Figure 9. Steps for adding the object to the Wave window to view the transient signal.

After the simulation, select both instances and go to *Wave -> Format -> Analog -> Automatic*. Finally, go to *Wave -> Zoom -> Zoom All*.

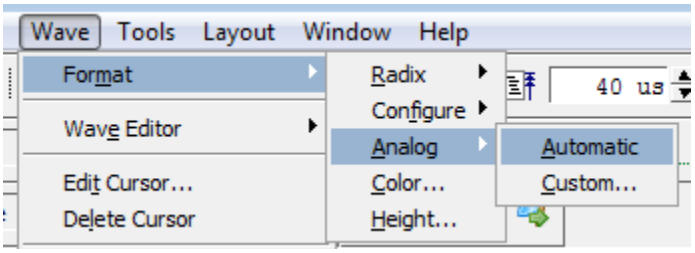


Figure 10. Steps for changing the display of the wave.

Here is the wave form of the filter input and the filter output of the test-bench, in the time domain. It is clear that the filter works as a highpass filter.

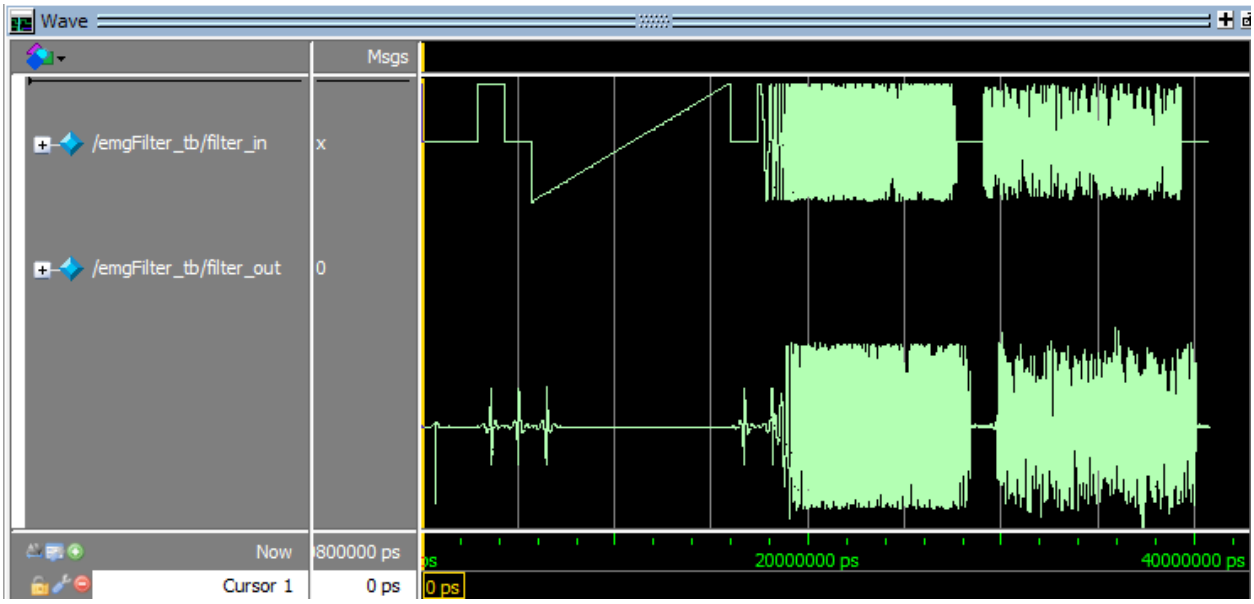


Figure 11. Waveforms of the input data and output data of the filter. Simulation time was of 40 us.

To find the frequency response, the waveform plot has to be converted from the time domain into the frequency domain. We need to take the information of the filter output and plot it on *MATLAB*.

In order to do that, we save the waveform as "*time vs. amplitude*" points on a spreadsheet using *Microsoft Excel*, and load the values on *MATLAB*. The application will turn the columns into vector-type variables for plotting.

Optional: To change the radix into singed fixed-point, select the object instances and either right click or go to *Wave -> Radix -> Fixed Point*... This will turn the decimal values into binary signed fixed point.

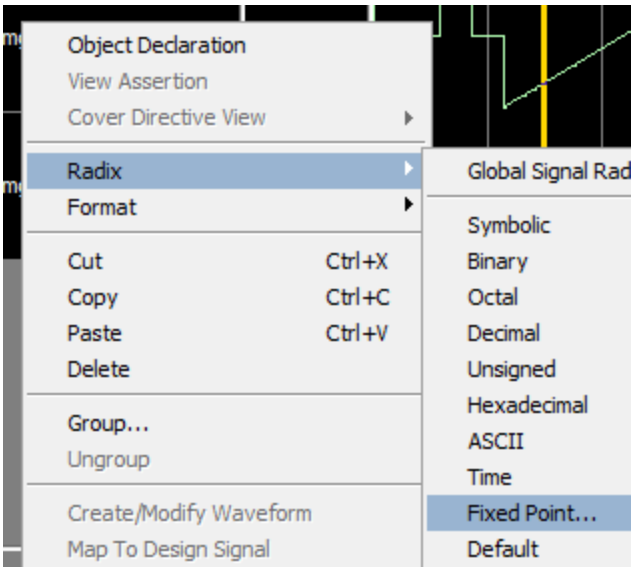


Figure 12. Steps on how to change the numerical base (radix) of the points.

Note: Since the *MATLAB* used for this project does not have the functions available to handle binary signed fixed-point (in any way) due to license issues, then the radix will be left as "*decimal*" on *ModelSim*.

To start saving the points, select the "*filter_out*" instance from the Objects window and go to *Add -> To List -> Selected Items*. This will display the amplitude values vs. time for the instance on the List window.

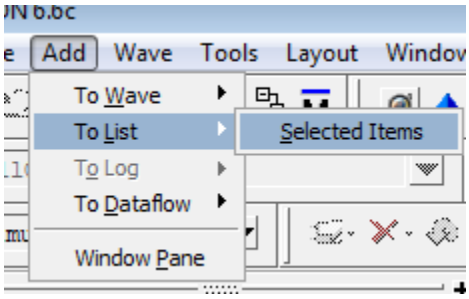


Figure 13. Steps on how to display the waveform as a list of values vs. time.

On the *List* window, go to *File -> Write List -> Tabular*... This will save the data in a readable text file, normally as ".lst" extension but it can be changed to ".txt" extension. Save the file as "*filter_out.txt*". Close the *List* window.

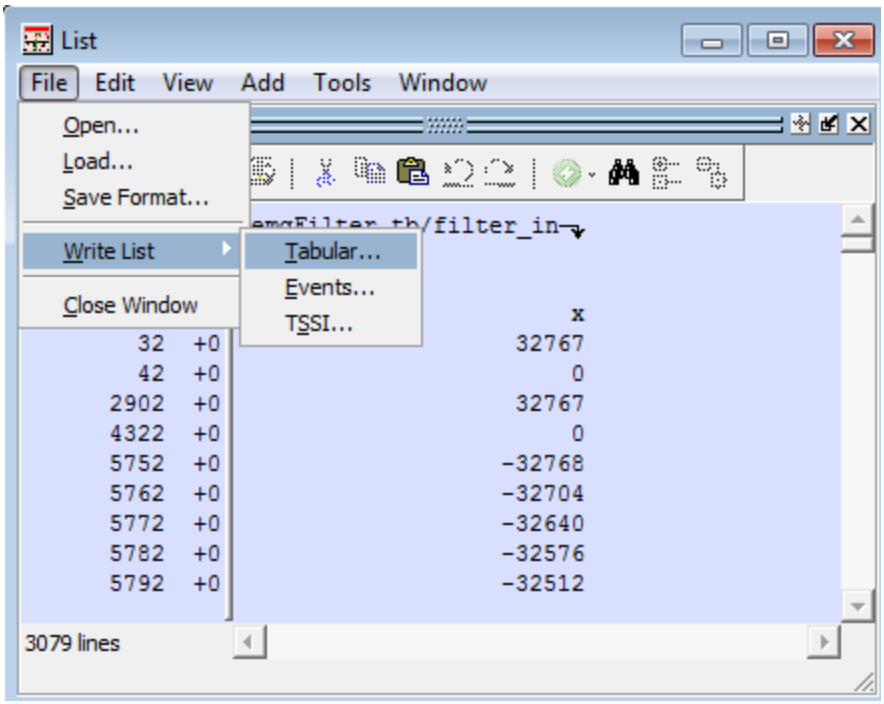


Figure 14. Once the values are listed, they are saved to a file.

Microsoft Excel 2010

Open a new *Microsoft Excel* worksheet and import the data from the "*filter_out.txt*" file. Go to *Data* tab and select *From Text*. On the first window of the *Text Import Wizard*, select the *Delimited* option and click Next. On the next window, select *Space* as delimiter. Click OK and Finish.

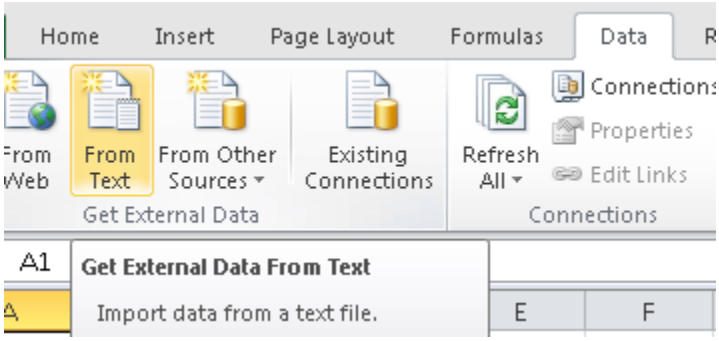


Figure 15. Step on how to import the saved list file.

Erase rows on which text and characters may appear, just remember which column is the time and which is the amplitude values. MATLAB cannot handle characters on numeric type variables.

Save the file as "*filter_highpass.xlsx*" file.

MATLAB R2013 (8.2.0.701) -64 bit

Open *MATLAB*, and go to *Home -> Import Data*. Locate and open the spreadsheet file previously created. On the import wizard, assign a name for the variable which will contain the array of values (i.e. a vector-type variable). Then click on Import Selection to import the data and generate the variables.

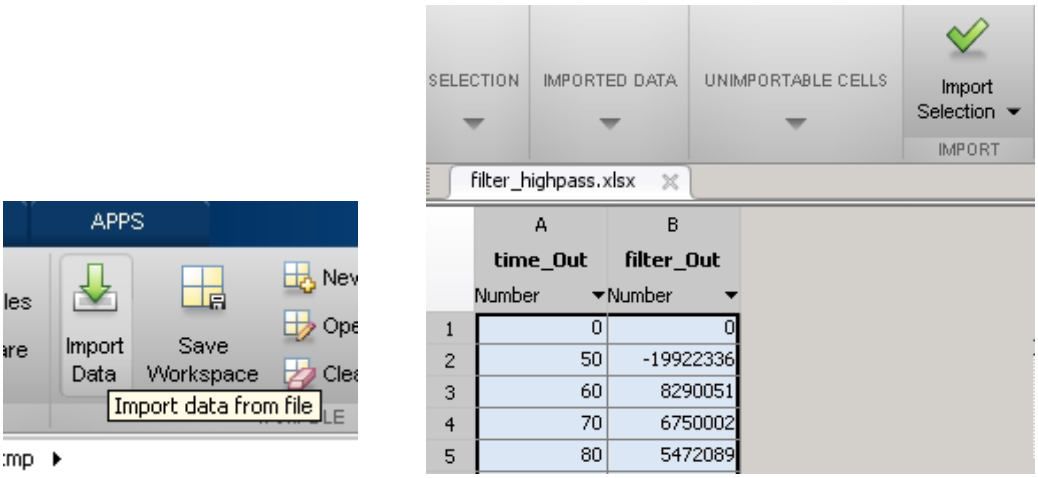


Figure 16. On MATLAB, click Import Data to add the list of values of the waveform for plotting.

The variables are "*time_out*" and "filter_*out*" which corresponds to the filter output waveform. As a test, **Figure 17** shows the plotting of the time domain waveform using the following commands [4][5]:

```
>> plot(time_out,filter_out);
>> xlabel('time ns');
>> ylabel('amplitude');
```

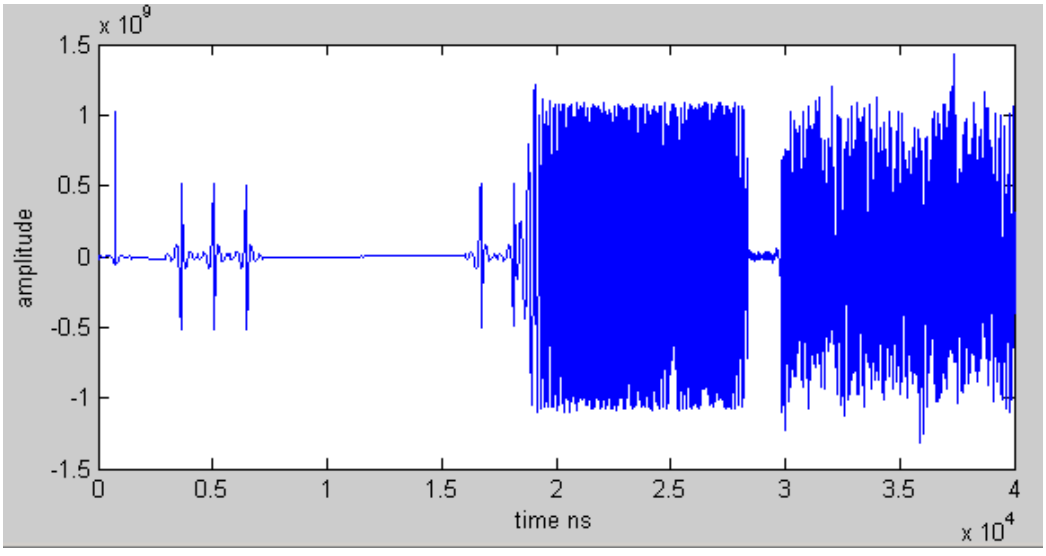


Figure 17. Filter output waveform on time domain using the saved data from ModelSim.

To determine the frequency of the wave at some point, first we have to measure two peak points on the plot using the "`ginput(n)`" command, where *n* is the number of points (x,y) that are going to be measured. A crosshairs appears on the plot, and lets us click on the desired points.

Then take the time component (x-axis) at each point and subtract them. Finally set the frequency = 1/time.

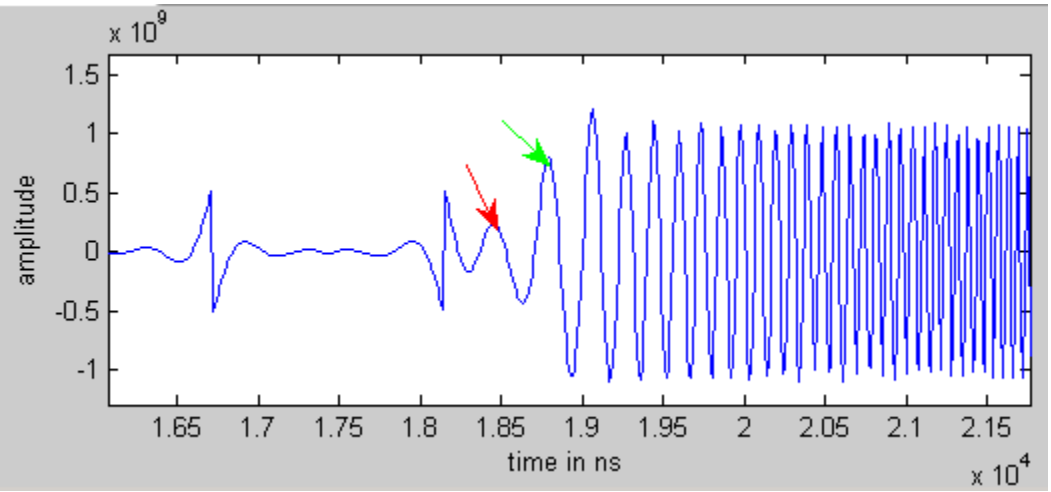


Figure 18. Shows the points being measured.

```
>> points = ginput(2)

points =
   1.0e+08 *

    0.0002    2.4670
    0.0002    7.9311

>> T=(points(2,1)-points(1,1))*10^-9

T =
   3.4042e-07

>> F=1/T

F =
   2.9375e+06
```

T was converted from ns to s. F is in hertz. It seems that the highpass cut-off frequency is around Fc = 3 MHz. Frequencies smaller than 3 MHz are attenuated.

To convert the time domain waveform into a frequency domain waveform, a Fast Fourier Transform is being used on the sampled data, which employs internally a Discrete Fourier Transform.

Using the following code on *MATLAB*, we can compute a plot called *periodogram* (power vs. frequency) of our sampled data:

```
>> dt=mean(diff(time_out))*10^-9;    %mean sampling period, unless known
>> m=length(filter_out);             %number of samples (window length)
>> y=fft(filter_out,m);              %Discrete Fourier Transform
>> fs=1/dt;                          %sampling frequency
>> f=(0:m-1)*(fs/m);                 %frequency range
>> power=y.*conj(y)/m;               %power of the DFT
>> plot(f,power)                     %plot command
>> xlabel('Frequency (Hz)')
>> ylabel('Power')
>> title('{\bf Periodogram}')
```

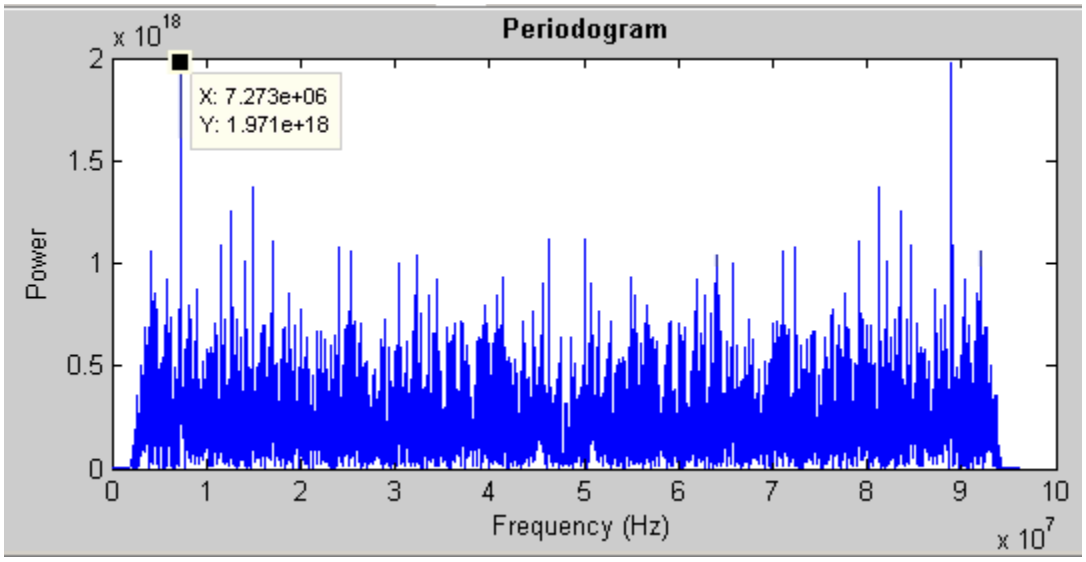


Figure 19. Power vs. Frequency.

Due to the periodicity of the DFT, the first half of the frequency range (from 0 to the Nyquist frequency fs/s) is sufficient to identify the component frequencies in the data, since the second half is just a reflection of the first half. The following code makes the fix:

```
>> y0=fftshift(y);                  %rearrange the y-axis values
>> f0=(-m/2:m/2-1)*(fs/m);          %0-centered frequency range
>> power0=y0.*conj(y0)/m;           %0-centered power
>> plot(f0,power0)                  %plotting command
```

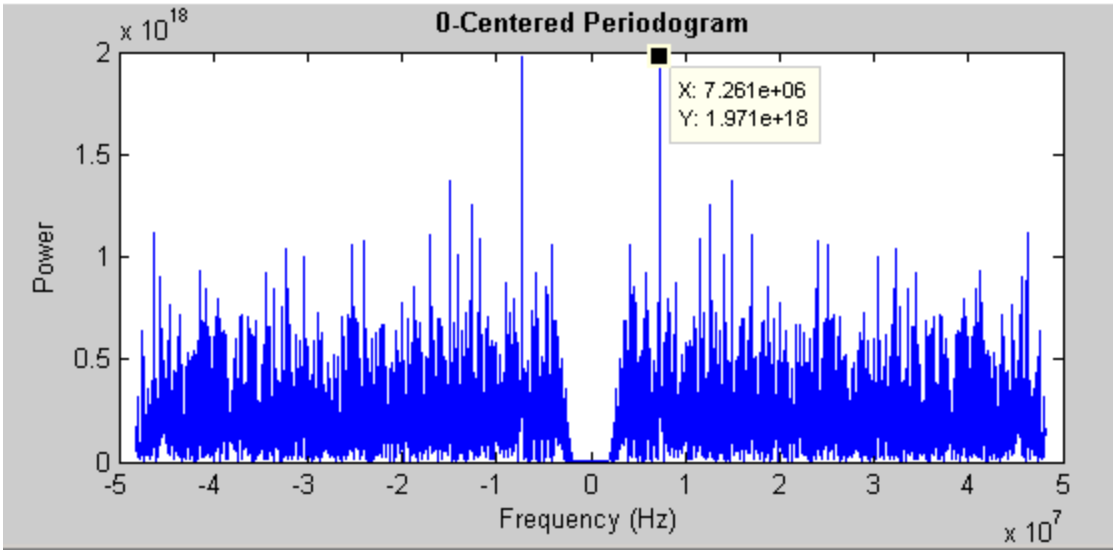


Figure 20. Power vs. Frequency plot centered at 0 Hz.

For a phase plot, use the following commands:

```
>> phase=unwrap(angle(y));
>> plot(f,phase*180/pi);
```

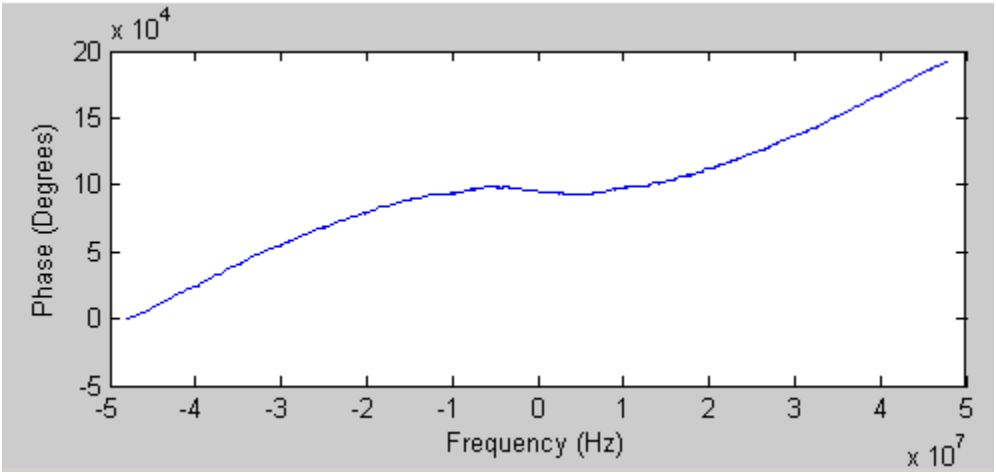


Figure 21. Phase vs. Frequency plot

Synthesis

Design Vision

Start *Design Vision*. Go to *File -> Read…* Locate and load the "*emgFilter.v*" file. Analyze the verilog code for any syntactical error and make the proper corrections. To do so, go to *File -> Analyze*. On the *Analyze Designs* window, add the same verilog file and select *Verilog* as the format. Click OK.

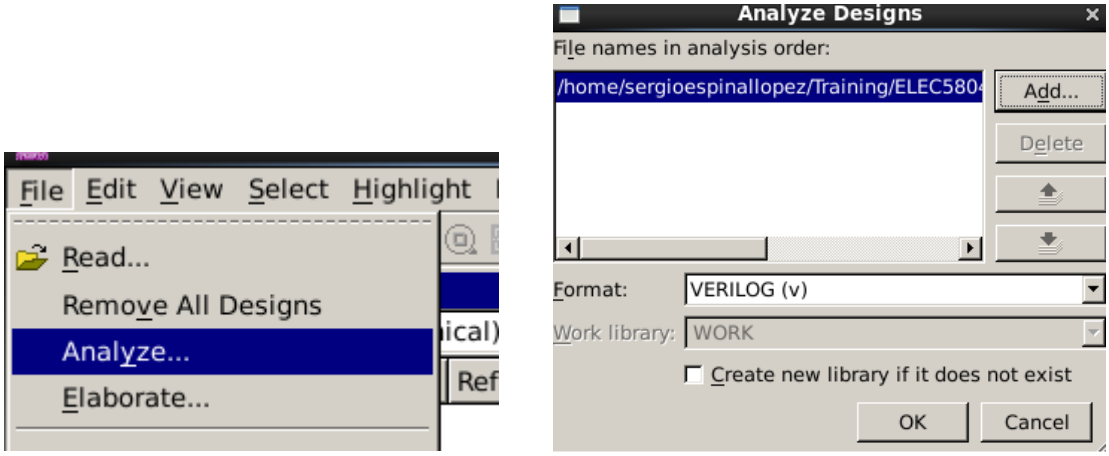


Figure 22. Steps for analyzing the verilog code file.

After analyzing the file, Verilog library objects are created in a Verilog-independent intermediate format for our use. Next step is to translate the design into a technology-independent design (GTECH) from the intermediate files produced during analysis [6]. To do so, go to *File -> Elaborate*.

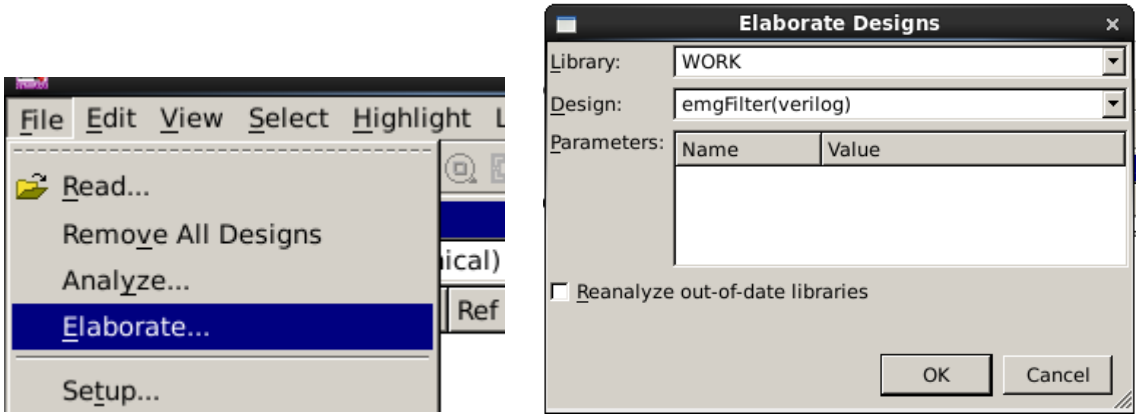


Figure 23. Steps for executing the Elaborate command.

Finally, execute the command "*uniquify*" by writing it on the *console command line* field. This will make all names of each object unique.

A load has to be specified on the output lines of the filter. Go to List -> Ports/Pins View -> Output Ports to load the list of output ports of the filter. The load is understood as a capacitive load, in pf. Go to Attributes -> Operating Environment -> Load. On each output port from the list, set the load to 0.2 pf.

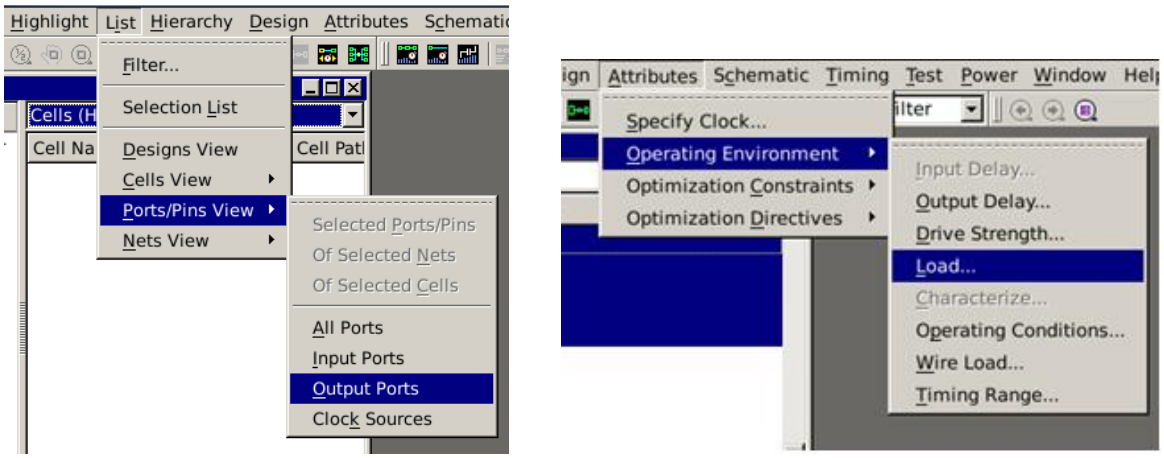


Figure 24. Steps for loading the output port list and for setting the load on each output port.

The clock is an important constraint. The clock is used to synchronize the operation of the circuit. To set a clock rate, go to *Attributes -> Specify Clock*… On the *Specify Clock* window, add a name for the clock (CLK_0), set the period, rising edge and falling edge as shown in **Figure 25**. Also don't forget to check the *Don't touch network* box so that no components are added or removed due to this new clock setting.

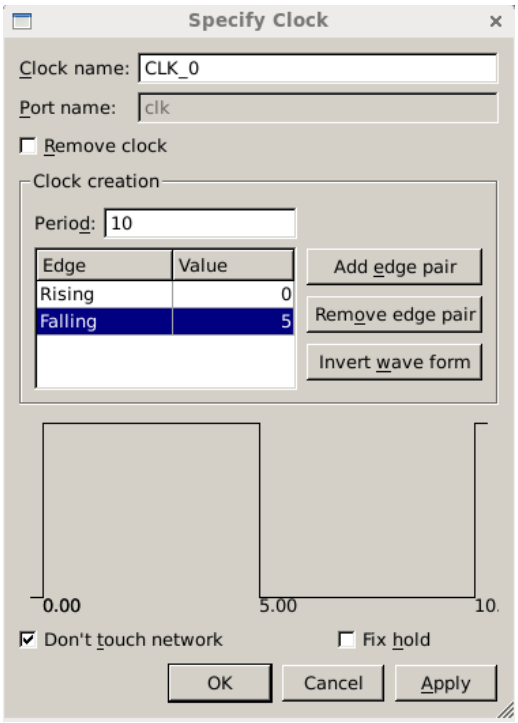


Figure 25. Steps for setting the clock rate for the circuit.

Repeat this step for another clock named CLK_1, but with a smaller clock rate.

On the console command line, write the following command to add uncertainty to the circuit, based on the two clocks:

```
set_clock_uncertainty 0.25 -from CLK_0 -to CLK_1
```

Next, execute the following command to compile the circuit design with the previously loaded component libraries:

```
compile_ultra
```

Save the compiled design file as "*emg1.v*".

Finally, execute the following command to save the Synopsis Design Constraints into a file:

```
write_sdc emg1.sdc
```

Layout Generation

Cadence Encounter

Run *Cadence Encounter*. To start the layout design, go to *File -> Import Design*. Browse for the synthesized verilog file from *Design Vision*. The design requires some "*.lef*" files, which are files that contain information about the layer specifications for the technology used by the components library.

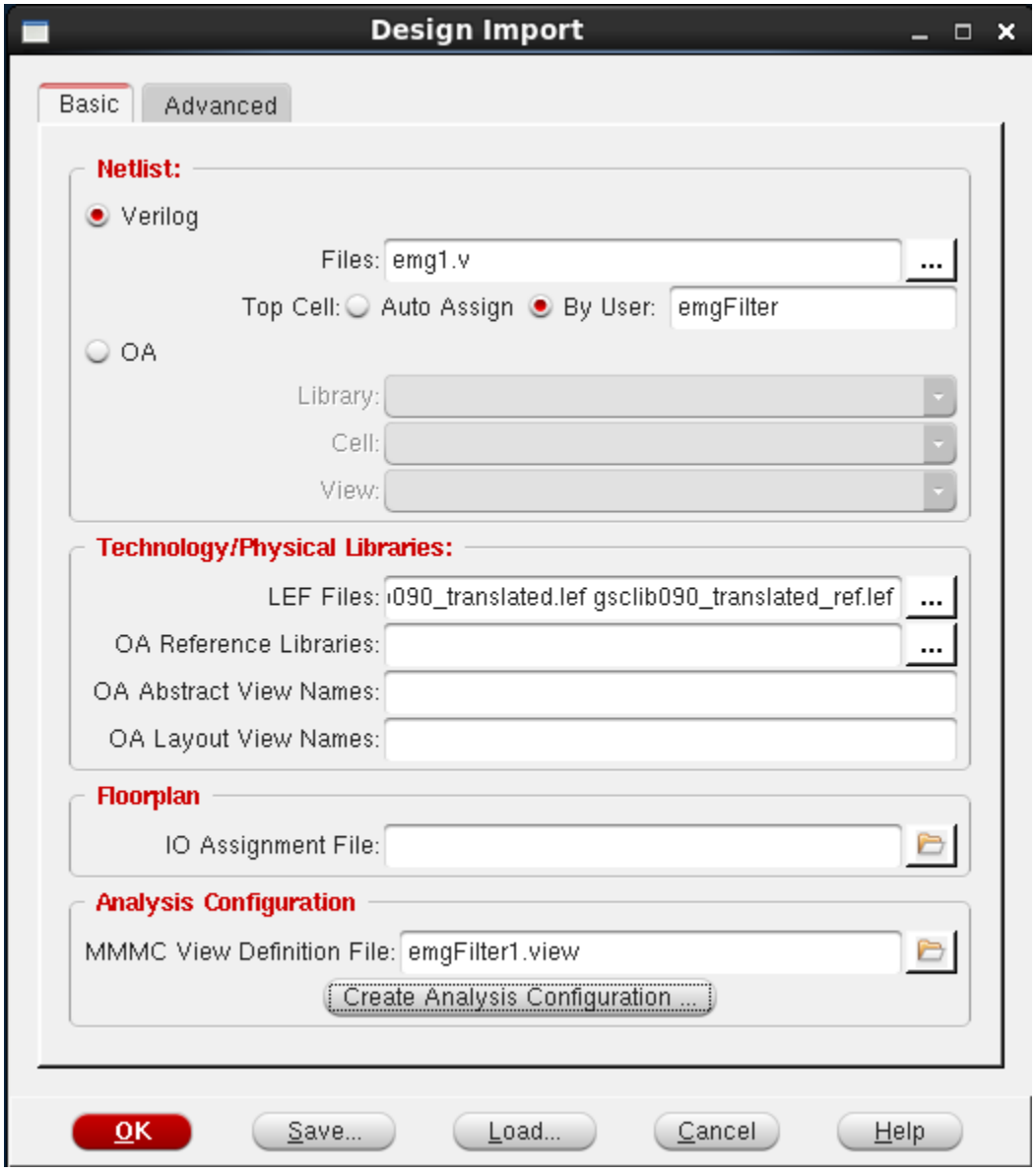


Figure 26. Design Import window

On the *Advanced* tab, select Power and set "VDD" as *Power Nets* and "VSS" as *Ground Nets*.

To configure the MMMC, click on the *Create Analysis Configuration* button. On the *MMMC Objects*, select Library Sets. Add a name and attach the necessary libraries. Go to *Constraint Modes*, add a name and attach the *Synopsis Design Constraint* file(s). Click Save & Close to save the configuration.

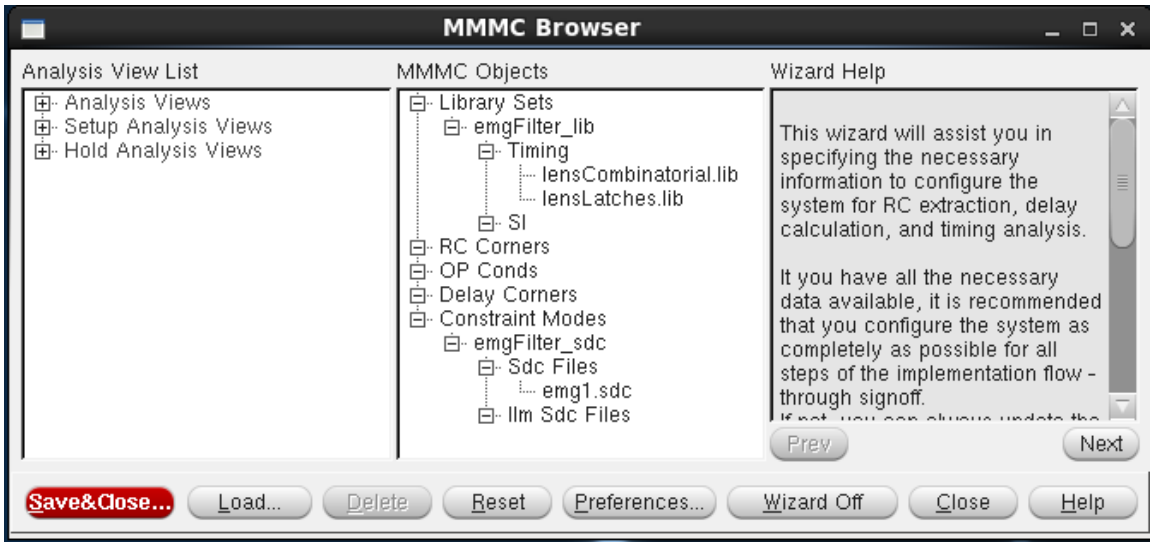


Figure 27. MMMC Browser.

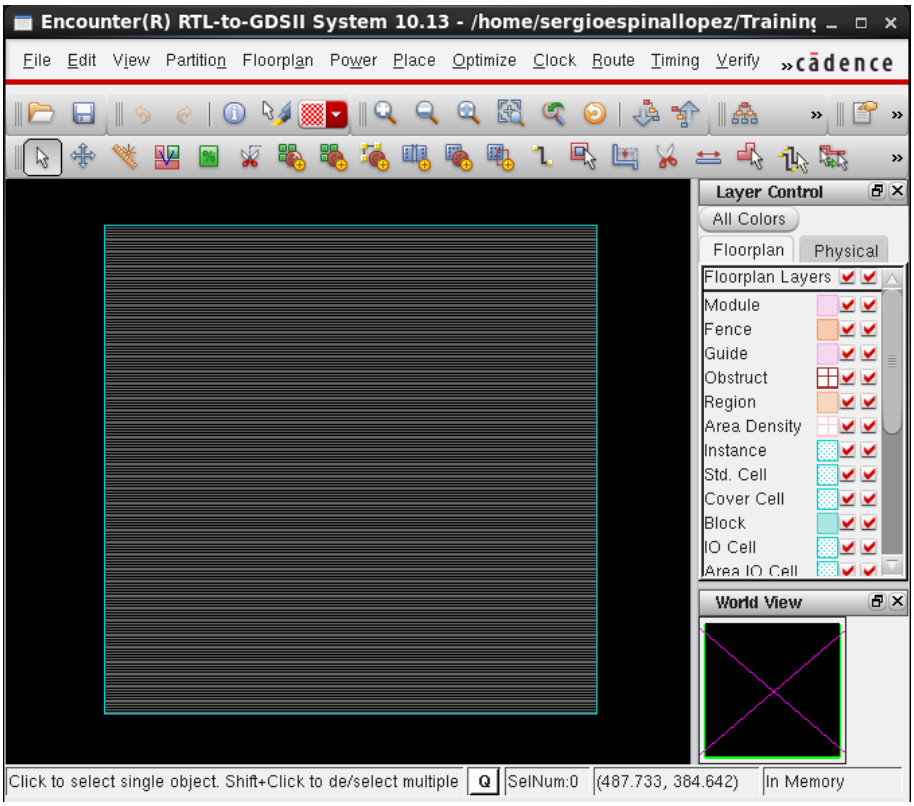
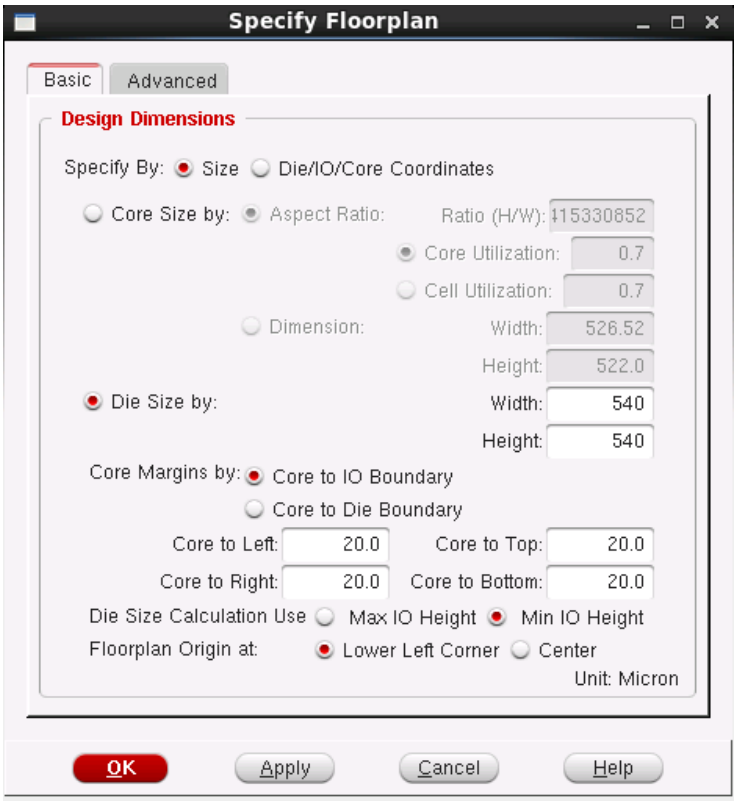


Figure 28. Design area of the layout.



To specify the floorplan, go to *Floorplan -> Specify Floorplan…* On the *Specify Floorplan* screen, specify by Size. Also set the Die Size by width and height as shown in **Figure 29**.

Figure 29. Specifying the floorplan.

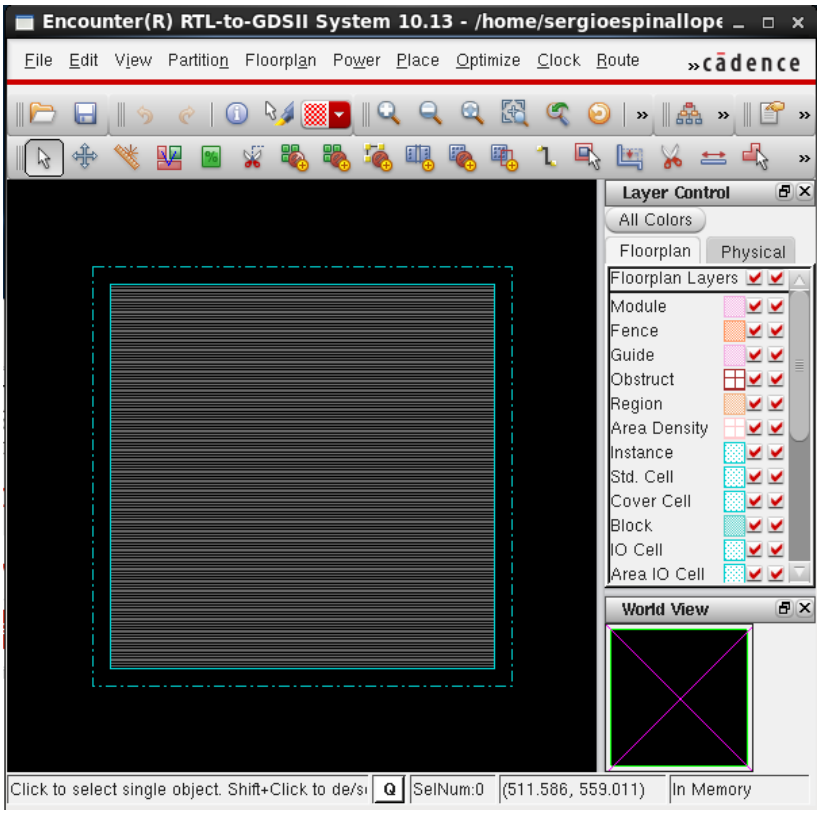


Figure 30. Design area after setting the floorplan.

To assign the power rails to where the component power pins will be connected, go to *Power -> Connect Global Nets*. On *Connect* tab: select Pin, * for all instance basenames, and “VDD” (or “VSS”) for the pin name. On *Scope* tab: select *Apply All*. Finally write “VDD” (or “VSS”) on *To Global Net* field. Click Apply and close.

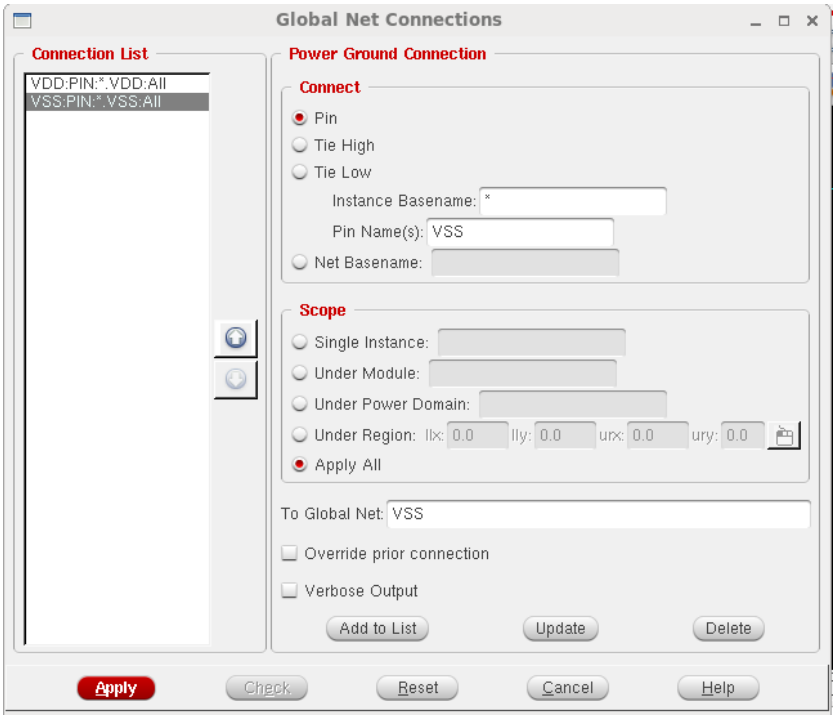


Figure 31. Global Net Connection screen.

To add the power rings around the die, go to *Power -> Power Planning -> Add Ring*… On the Basic tab, write “VDD” and “VSS” on the *Net(s)* field. On *Ring Type*, Select the ring to be placed *Around core boundary*. Set the type of metal, width and spacing of the net on each side of the ring at *Ring Configuration*.

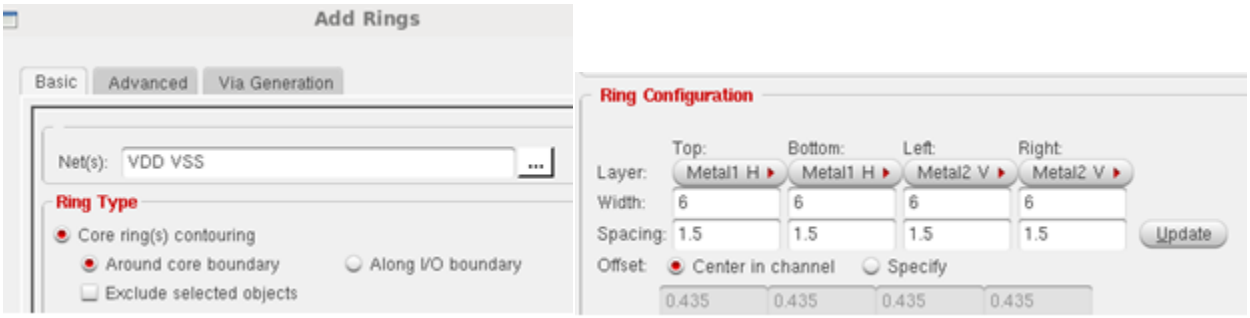


Figure 32. Add ring configuration screen

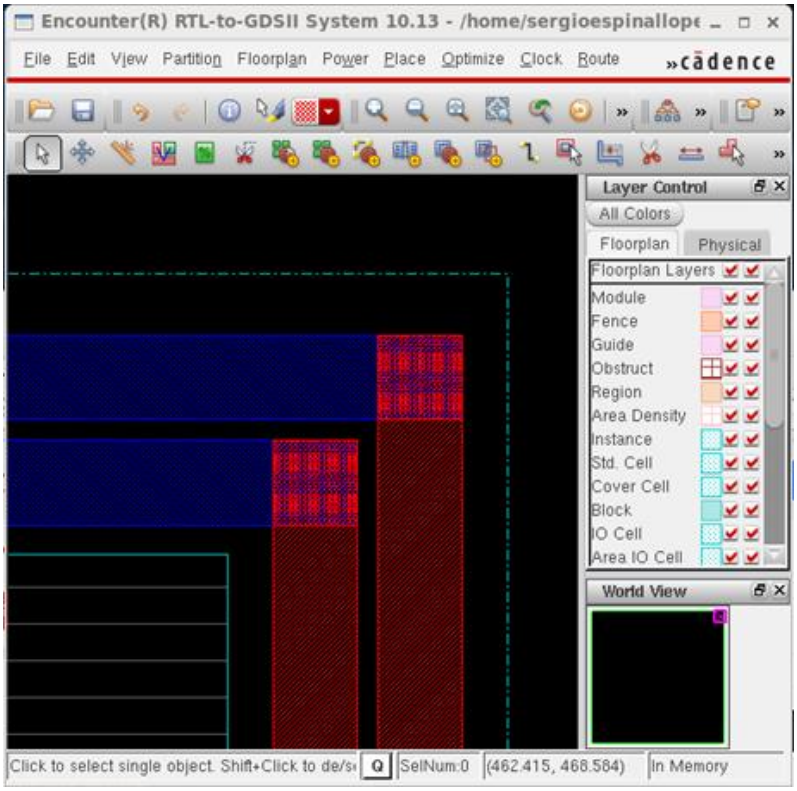


Figure 33. Layout with the added power rings.

The software is using “standard cell” libraries. Basically, the components or “cells” have being designed in a way that their power tabs (VDD and VSS) have the same distance between each other. This standardized design lets them share the same metals used for power nets when they are placed on a same row, two or more components side-by-side (abutted), or components facing each other with the same power tap (on different rows).

To place the metal of the power nets used for this purpose, go to *Route -> Special Route*… On the Net(s) field, add the VDD and VSS nets. Click OK.

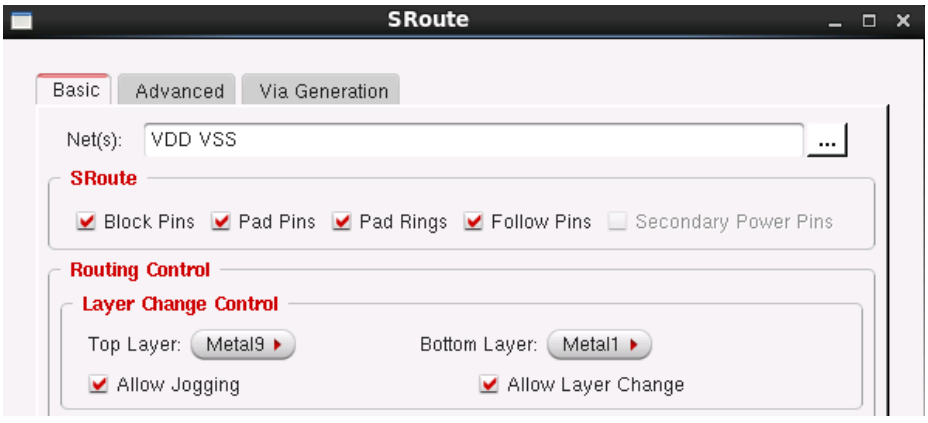


Figure 34. Special Route configuration window.

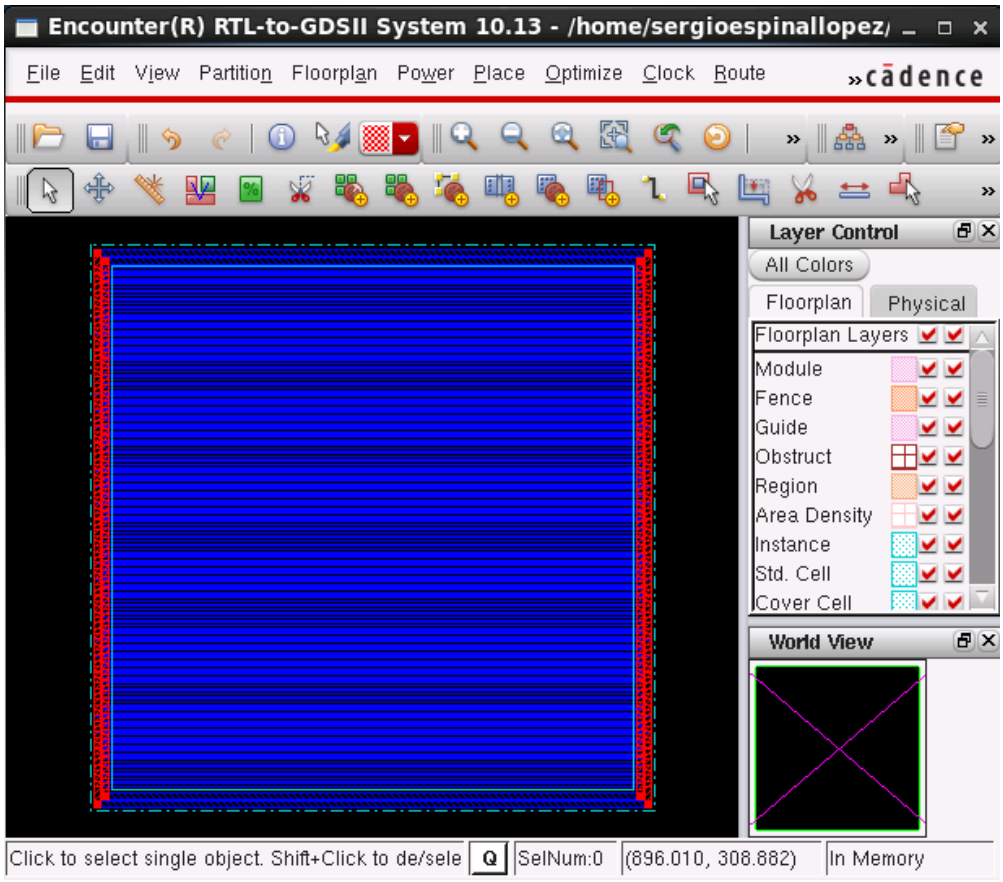


Figure 35. Die after setting the metal rows used for power routing for the standard cells.

To place the standard cells instantiated on the synthesized Verilog code file, go to *Place -> Place Standard Cells*.

Figure 36. Standard cell placement configuration.

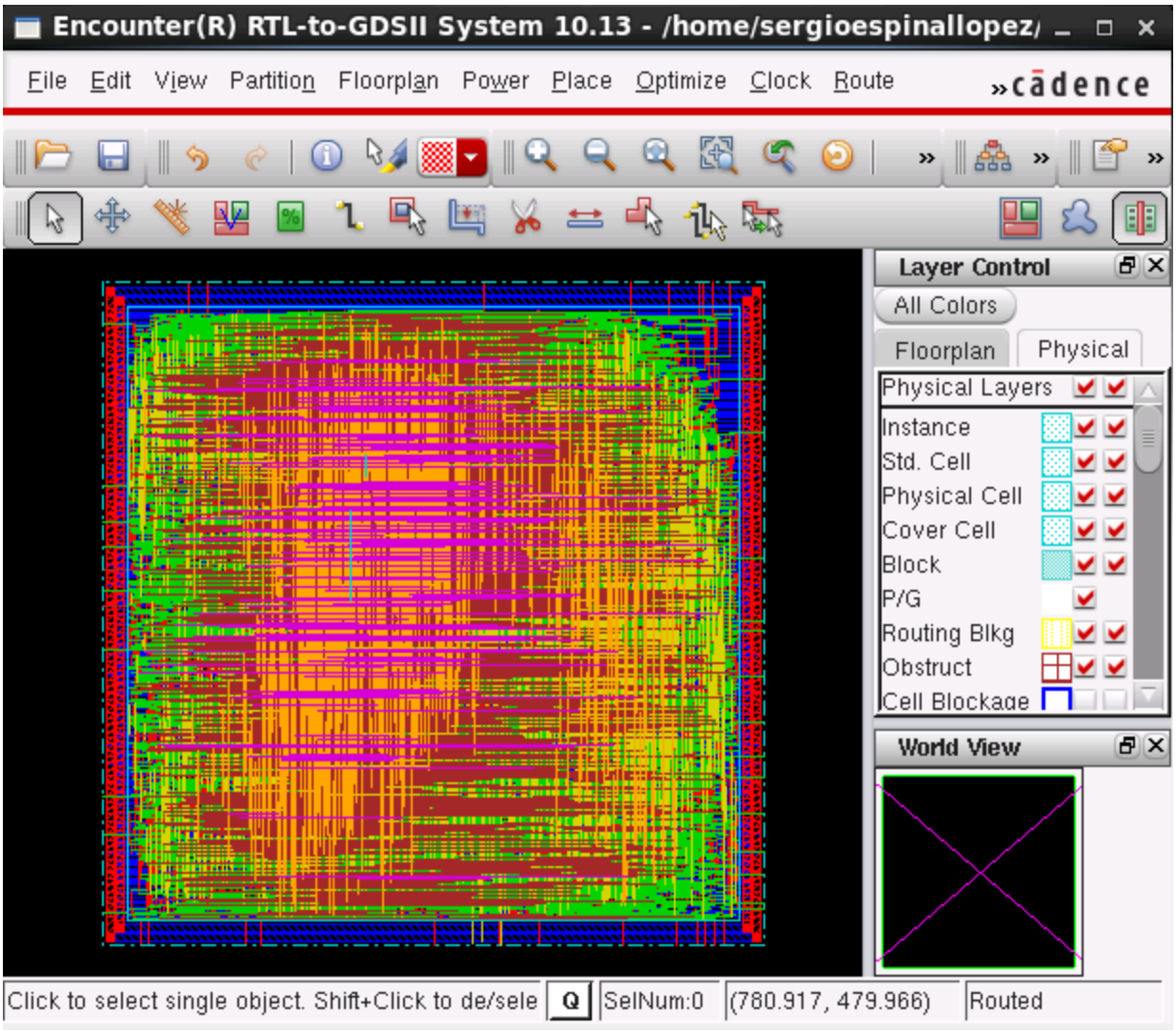


Figure 37. After placing the standard cells, the software performs a trial routing to make all the connections.

In order to further optimize the standard cell routing, go to *Route -> Nano Route -> Route*.

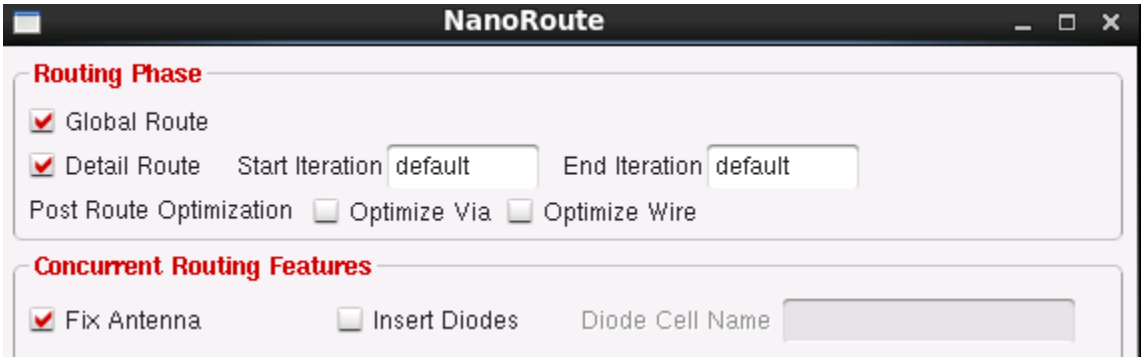


Figure 38. NanoRoute configuration window.

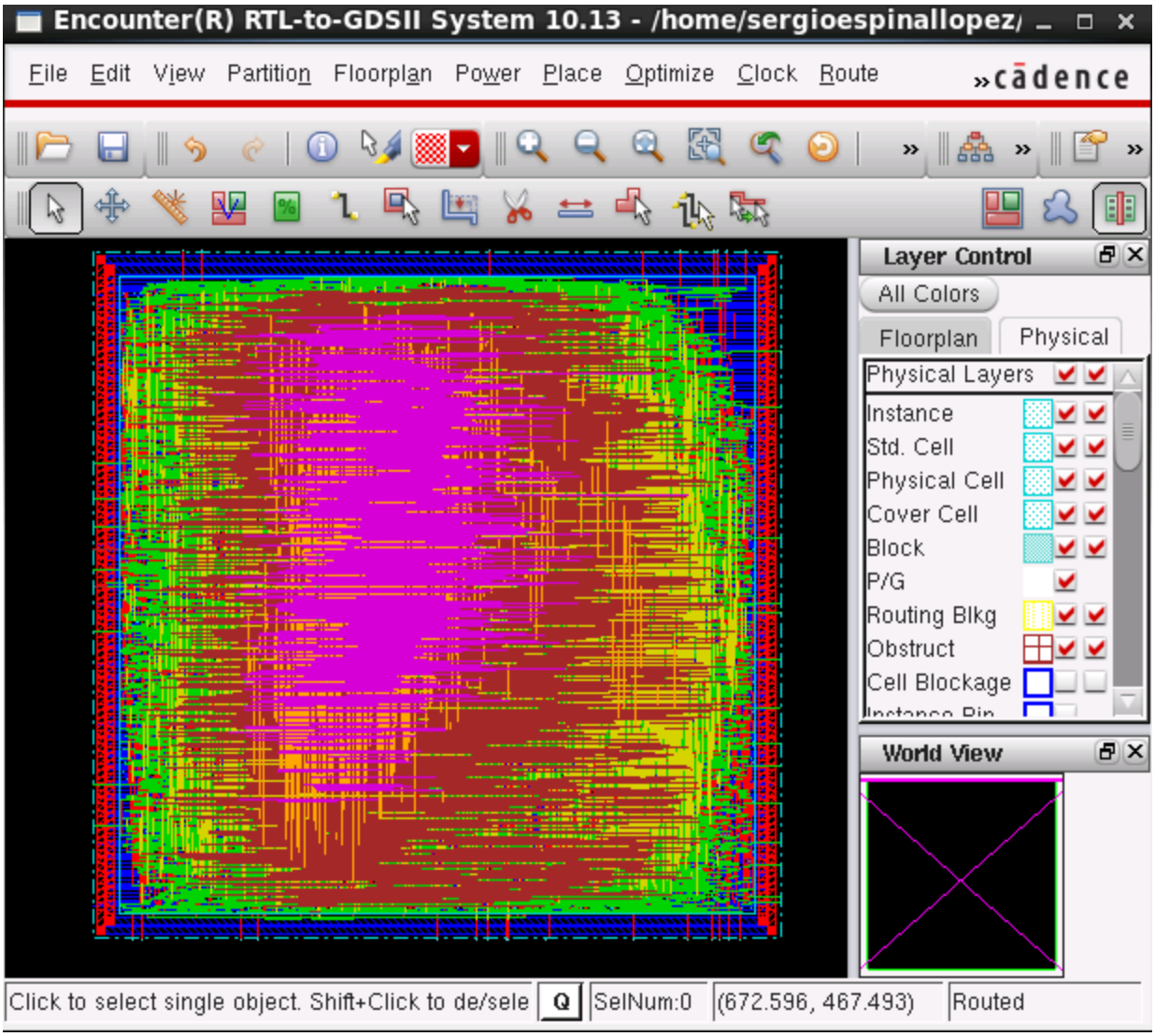


Figure 39. Layout after performing NanoRouting.

For design constraint purposes, it is necessary to fill the gaps between each standard cell. To do so, go to *Place -> Physical Cell -> Add Filler*… This will add "standard dummy cells" to fill the gaps.

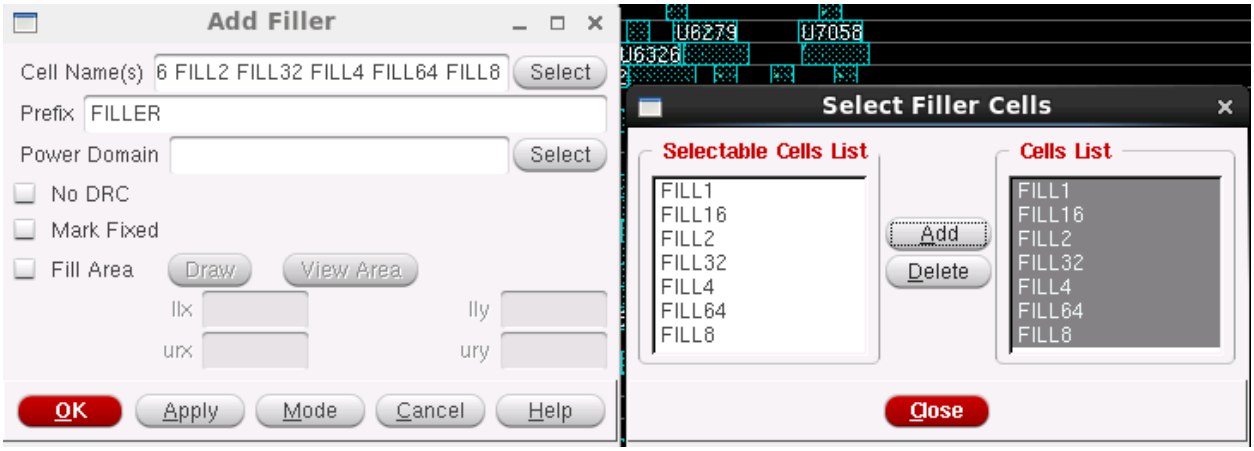


Figure 40. Add Filler configuration window.

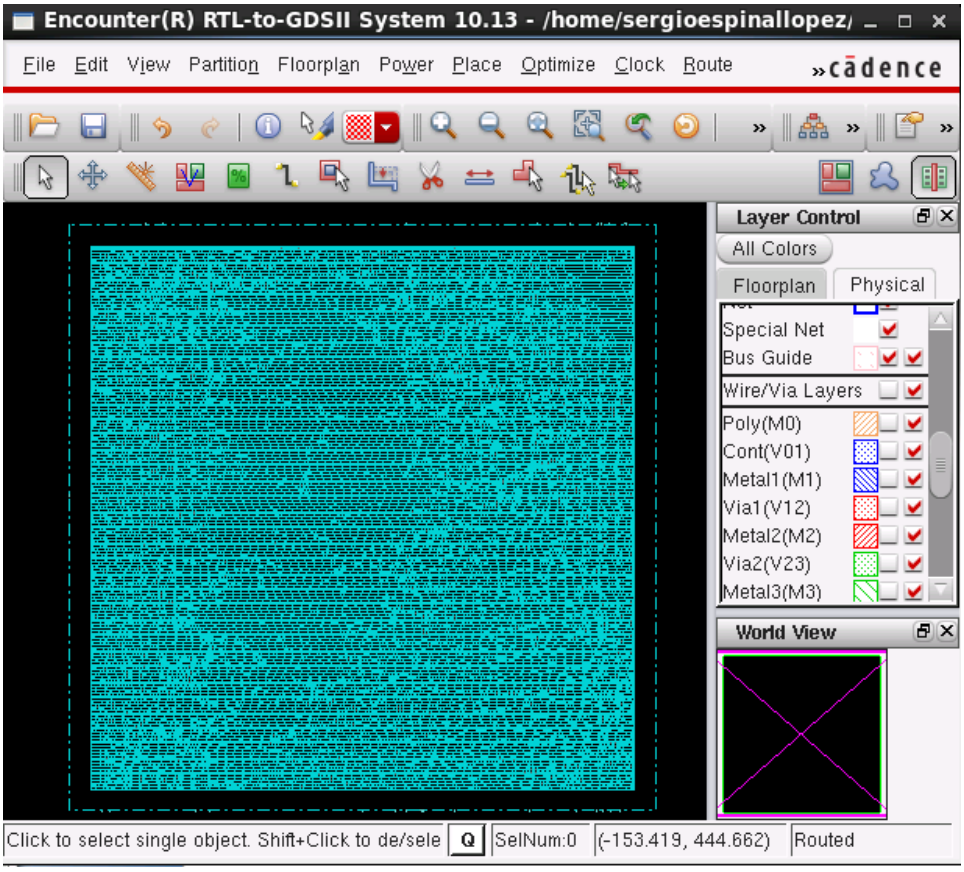


Figure 41. Die after adding the filler cells.

Also, to add metal, go to *Route -> Metal Fill -> Add*. This will fill all the remaining gaps with all posible metals without compromising the layout design.

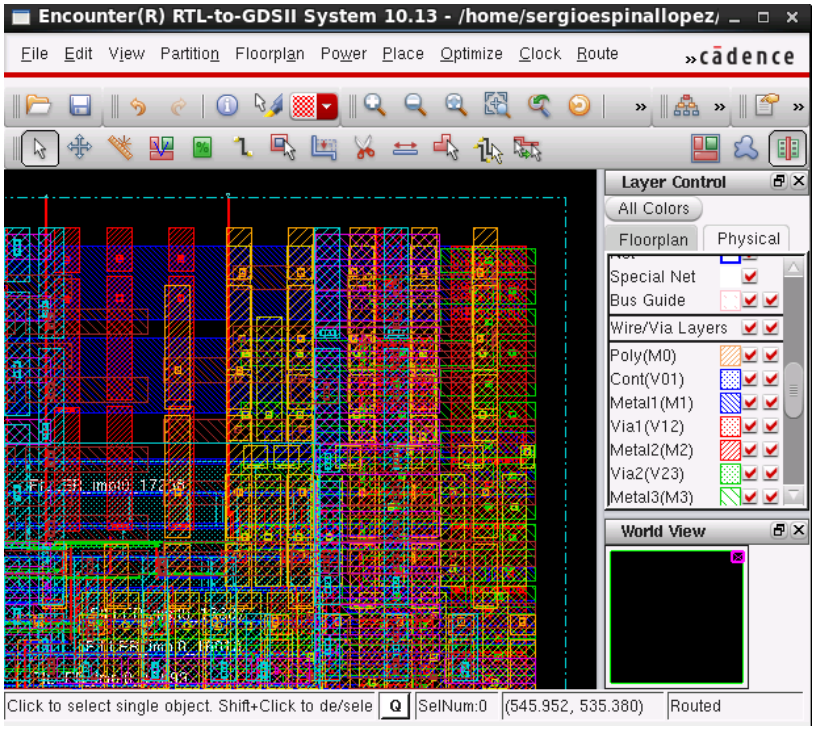


Figure 42. View of the layout with the added layers of metal.

Finally, to export the design for further analysis on Cadence Virtuoso, go to *File -> Save -> GDS/OASI*. This will save a ".gds" stream file.

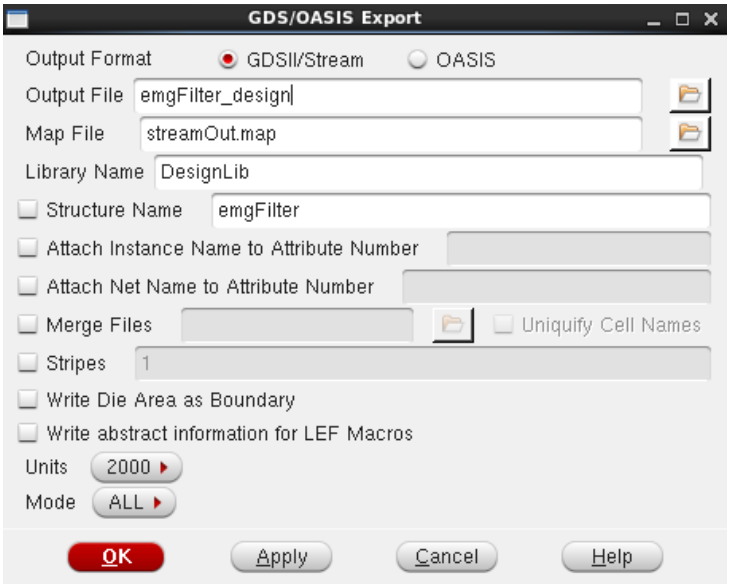


Figure 43. GDS/OASIS Export wizard.

Results and Discussion

- First of all, due to the lack of time this project went only this far.
- The testbench code was not analyzed, as it was with the filter core code.
- The *MATLAB* license couldn´t load some functions that can handle signed fixed point values.
- The cell area being reported by *Design Vision* is approx. 192,390.35 um^2. No errors were reported.
- The cell area being reported by *Cadence Encounter* is approx. 249,235.06 um^2.
- Due to an unresolved issue, it was not possible to make an extracted layout and due a further analysis on the circuit. The error is related to the technology library used during the process. After importing the "*.gds*" file from *Cadence Encounter* into *Cadence Virtuoso*, the standard cells are not present on the generated layout. It seems that the standard cells present on the file are not associated with the library being used by Virtuoso.

Fragment of the error from *Cadence Virtuoso*:

```
****************************************************************************
Product    : Virtuoso(R) XStream In
Program    : @(#)$CDS: strmin version 6.1.5 01/16/2012 21:51 (sjfdl226) $
           : sub-version  IC6.1.5.500.9
Started at: 28-Dec-2013  01:49:38
User Name : sergioespinallopez
Host Name : macopeland.doe.carleton.ca
Directory : /home/sergioespinallopez/Training/ELEC5804/assignment03/DB
CADENCE Design Systems, Inc.
****************************************************************************

WARNING (XSTRM-267): The referenced cell "DFFRHQX1" was not found. The OpenAccess design data was
created for this cell without any reference. Ensure that the referenced cell exists and library
is defined in the cds.lib file.
WARNING (XSTRM-267): The referenced cell "MX2X1" was not found. The OpenAccess design data was
created for this cell without any reference. Ensure that the referenced cell exists and library
is defined in the cds.lib file.
WARNING (XSTRM-267): The referenced cell "ADDFXL" was not found. The OpenAccess design data was
created for this cell without any reference. Ensure that the referenced cell exists and library
is defined in the cds.lib file.
```

Bibliography

1) B.A. Shenoi. *Introduction to Digital Signal Processing and Filter Design*. Hoboken, New Jersey: John Wiley & Sons, Inc. 2006. pp. 3-7, 34-37, 122, 249-260.
2) Li Tan. Digital Signal Processing: Fundamentals and Applications. DeVry University. Decatur, Georgia. Elsevier Inc. 2008. pp. 159-288.
3) Ph.D. Leonard MacEachern. ELEC 5804, Department of Electronics. Carleton University, Canada. Internet: http://elec975804.blogspot.ca/, Sep. 10, 2013 [Dec. 27, 2013].
4) MathWorks (2013). *Discrete Fourier Transform*. Available: http://www.mathworks.com/help/matlab/math/discrete-fourier-transform-dft.html
5) MathWorks (2013). *Fast-Fourier Transform*. Available: http://www.mathworks.com/help/matlab/math/fast-fourier-transform-fft.html
6) Tamperee University of Technology. *Design Compiler Tutorial: Reading in the design*. Available: http://www.tkt.cs.tut.fi/tools/public/tutorials/synopsys/design_compiler/gsdc.html
7) Brett Ninness. *Spectral Analysis using the FFT*. Department of Electrical and Computer Engineering. The University of Newcastle, Australia. Available: http://sigpromu.org/brett/elec2400/matlab3.pdf

Appendix

"emgFilter.tcl" script for Design Vision

```
gui_start
read_file -format verilog
{/home/sergioespinallopez/Training/ELEC5804/assignment04/filters/emgFilter/emgFilter.v}
analyze -format verilog
{/home/sergioespinallopez/Training/ELEC5804/assignment04/filters/emgFilter/emgFilter.v}
elaborate emgFilter -architecture verilog -library WORK
uniquify
set_load 0.200000 [get_ports "filter_out[12]"]
set_load 0.200000 [get_ports "filter_out[19]"]
set_load 0.200000 [get_ports "filter_out[20]"]
set_load 0.200000 [get_ports "filter_out[27]"]
set_load 0.200000 [get_ports "filter_out[9]"]
set_load 0.200000 [get_ports "filter_out[0]"]
set_load 0.200000 [get_ports "filter_out[10]"]
set_load 0.200000 [get_ports "filter_out[17]"]
set_load 0.200000 [get_ports "filter_out[25]"]
set_load 0.200000 [get_ports "filter_out[7]"]
set_load 0.200000 [get_ports "filter_out[6]"]
set_load 0.200000 [get_ports "filter_out[15]"]
set_load 0.200000 [get_ports "filter_out[23]"]
set_load 0.200000 [get_ports "filter_out[4]"]
set_load 0.200000 [get_ports "filter_out[31]"]
set_load 0.200000 [get_ports "filter_out[13]"]
set_load 0.200000 [get_ports "filter_out[21]"]
set_load 0.200000 [get_ports "filter_out[28]"]
set_load 0.200000 [get_ports "filter_out[1]"]
set_load 0.200000 [get_ports "filter_out[11]"]
set_load 0.200000 [get_ports "filter_out[18]"]
set_load 0.200000 [get_ports "filter_out[26]"]
set_load 0.200000 [get_ports "filter_out[8]"]
set_load 0.200000 [get_ports "filter_out[16]"]
set_load 0.200000 [get_ports "filter_out[24]"]
set_load 0.200000 [get_ports "filter_out[5]"]
set_load 0.200000 [get_ports "filter_out[32]"]
set_load 0.200000 [get_ports "filter_out[14]"]
set_load 0.200000 [get_ports "filter_out[22]"]
set_load 0.200000 [get_ports "filter_out[29]"]
set_load 0.200000 [get_ports "filter_out[3]"]
set_load 0.200000 [get_ports "filter_out[30]"]
set_load 0.200000 [get_ports "filter_out[2]"]
create_clock -name "CLK_0" -period 10 -waveform { 0 5  }  { clk  }
set_dont_touch_network  [ find clock CLK_0 ]
create_clock -name "CLK_0" -period 10 -waveform { 0.000 5.000  }  { clk  }
set_dont_touch_network  [ find clock CLK_0 ]
create_clock -name "CLK_1" -period 5 -waveform { 0 2.5  }  { clk_enable  }
set_dont_touch_network  [ find clock CLK_1 ]
set_clock_uncertainty 0.25 -from CLK_0 -to CLK_1
uplevel #0 source
/home/sergioespinallopez/Training/ELEC5804/assignment04/gpdk90_plus_dw_synopsys_dc.setup.tcl
compile_ultra
write -hierarchy -format verilog -output
/home/sergioespinallopez/Training/ELEC5804/assignment04/filters/emg1.v
write_sdc emg1.sdc
```

Area report from Design Vision

```
****************************************
Report : area
Design : emgFilter
Version: F-2011.09-SP4
Date   : Sun Dec 22 18:18:06 2013
****************************************

Information: Updating design information... (UID-85)
Warning: Design 'emgFilter' contains 2 high-fanout nets. A fanout number of 1000 will be used for
delay calculations involving these nets. (TIM-134)
Warning: A non-unate path in clock network for clock 'CLK_1'
 from pin 'U1141/Y' is detected. (TIM-052)
Library(s) Used:

    lensCombinatorial (File:
/home/sergioespinallopez/Training/ELEC5804/assignment04/LIBS/lensCombinatorial.db)
    lensLatches (File:
/home/sergioespinallopez/Training/ELEC5804/assignment04/LIBS/lensLatches.db)

Number of ports:                           52
Number of nets:                         20555
Number of cells:                        14864
Number of combinational cells:          12543
Number of sequential cells:              2321
Number of macros:                           0
Number of buf/inv:                       2791
Number of references:                      32

Combinational area:       144957.700526
Noncombinational area:    47432.652945
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          192390.353470
Total area:                 undefined
```

MMMC configuration

```
# Version:1.0 MMMC View Definition File
# Do Not Remove Above Line
create_library_set -name emgFilter_lib -timing {lensCombinatorial.lib lensLatches.lib}
create_constraint_mode -name emgFilter_sdc -sdc_files {emg1.sdc}
```

checkDesign.rpt from Encounter

```
############################################################
#  Generated by:      Cadence Encounter 10.13-s250_1
#  OS:                Linux x86_64(Host ID macopeland.doe.carleton.ca)
#  Generated on:      Mon Dec 30 16:36:31 2013
#  Design:            emgFilter
#  Command:           checkDesign -io -netlist -physicalLibrary -powerGround...
############################################################


==============================
Design Stats
==============================
Design Name: emgFilter
    ------------------------------
    Cells used in the design
    ------------------------------
    ADDFX1
    ADDFXL
    AO21X1
    AO22X1
    AOI21X1
    AOI21XL
    AOI222XL
    AOI22XL
    BUFX4
    CLKBUFX2
    CLKBUFX3
    CLKBUFX4
    CLKINVX1
    CLKXOR2X1
    DFFRHQX1
    FILL1
    FILL16
    FILL2
    FILL32
    FILL4
    FILL64
    FILL8
    INVX1
    INVXL
    MX2X1
    MXI2XL
    NAND2BX1
    NAND2XL
    NAND3X1
    NOR2BX1
    NOR2XL
    OA21X1
    OAI21X1
    OAI21XL
    OAI222XL
    OAI2BB1X1
    OAI31X1
    OR2X1
    XNOR2X1
    Number of cells used in the design  39

==============================
Physical Library(LEF) Integrity Check
==============================
Cells with missing LEF: 0
Cells with missing PG PIN: 0
Cells with missing dimension: 0
Cells dimesions not multiple integer of site: 0
Cells pin with missing direction: 0
Cells pin with missing geometry: 0
```

```
Cells PG Pins with missing geometry: 0

==============================
Timing information check
==============================
    ------------------------------
    Cells with missing timing data
    ------------------------------
    ADDFX1
    ADDFXL
    AO21X1
    AO22X1
    AOI21X1
    AOI21XL
    AOI222XL
    AOI22XL
    BUFX4
    CLKBUFX2
    CLKBUFX3
    CLKBUFX4
    CLKINVX1
    CLKXOR2X1
    DFFRHQX1
    INVX1
    INVXL
    MX2X1
    MXI2XL
    NAND2BX1
    NAND2XL
    NAND3X1
    NOR2BX1
    NOR2XL
    OA21X1
    OAI21X1
    OAI21XL
    OAI222XL
    OAI2BB1X1
    OAI31X1
    OR2X1
    XNOR2X1
    Cells with missing Timing data  32

==============================
SPEF Coverage
==============================
Annotation to Verilog Netlist: 0%
Annotation to Physical Netlist: 0%

==============================
Top Level Netlist Check
==============================
Floating Ports: 0
Ports Connect to multiple Pads: 0
    ------------------------------
    Port connected to core instances
    ------------------------------
    Port name  # of connections
    filter_out[0]  2
    filter_out[1]  2
    filter_out[2]  2
    filter_out[3]  2
    filter_out[4]  2
    filter_out[5]  2
    filter_out[6]  2
    filter_out[7]  2
    filter_out[8]  2
    filter_out[9]  2
    filter_out[10]  2
    filter_out[11]  2
    filter_out[12]  2
    filter_out[13]  2
```

```
    filter_out[14]  2
    filter_out[15]  2
    filter_out[16]  2
    filter_out[17]  2
    filter_out[18]  2
    filter_out[19]  2
    filter_out[20]  2
    filter_out[21]  2
    filter_out[22]  2
    filter_out[23]  2
    filter_out[24]  2
    filter_out[25]  2
    filter_out[26]  2
    filter_out[27]  2
    filter_out[28]  2
    filter_out[29]  2
    filter_out[30]  2
    filter_out[31]  2
    filter_out[32]  2
    filter_in[0]  1
    filter_in[1]  1
    filter_in[2]  1
    filter_in[3]  1
    filter_in[4]  1
    filter_in[5]  1
    filter_in[6]  1
    filter_in[7]  1
    filter_in[8]  1
    filter_in[9]  1
    filter_in[10]  1
    filter_in[11]  1
    filter_in[12]  1
    filter_in[13]  1
    filter_in[14]  1
    filter_in[15]  1
    reset  1
    clk_enable  2321
    clk  2321
    Ports connected to core instances  52



==============================
Instance Pin Check
==============================

    Instances with input pins tied together  1410
    TieHi/Lo term nets not connected to instance's PG terms  0
    Floating Instance terminals  0
    Floating IO terms  0
    Tie Hi/Lo output terms floating  0
    Output term shorted to Power Ground net  0


==============================
Unplaced IO Pads
==============================
Unplaced I/O Pads: 0

==============================
Power Ground Nets
==============================
VSS : Routed
VDD : Routed

==============================
Power/Ground Pin Connectivity
==============================
    Floating Power Ground terms  0
    Power/Ground pins connected to non Power/Ground net  0
```

```
    Power pin connected to Ground net  0
    Ground pin connected to Power net  0
        1) Number of error=0 & warning=0.

============================
Top level Floorplan Check
============================
Off-Grid Horizontal Tracks: 0
Off-Grid Vertical Tracks: 0
Placement grid on Mfg. grid: FALSE
User grid a multiple of Mfg. grid: FALSE
User grid a multiple of Mfg. grid: FALSE
Core Row grid not a multiple of Mfg. grid: 0
Horizontal GCell Grid off Mfg. grid: 0
Vertical GCell Grid off Mfg. grid: 0
AreaIO rows not on core-grid: 0
BlackBoxes Off Mfg. Grid: 0
Blocks Off Mfg. Grid: 0
BlackBoxes Off placement Grid: 0
Blocks off placement Grid: 0
Instances not snapped to row site: 0
Instances not on Mfg. Grid: 0
PrePlaced hard-macro pins not on routing grid: 0
Floating/Unconnected IO Pins: 0
Unplaced Io Pins: 0
IO Pin off Mfg. grid: 0
IO Pin outside Die area: 0
Overlapping IO pins: 0
IO Pin off track: 0
Modules with off-grid Constraints: 0
Groups with off-grid Constraints: 0
Floating/Unconnected Ptn Pins: 0
Partition Pin off M. Grid: 0
Unplaced Partition Pins: 0
Partition Pin outside Partition box: 0
Overlapping Partition Pins: 0
Partition Pin Off-Track: 0
PartitionPower Domain off Grid: 0
PreRoute not on Mfg. grid: 0
Off Track Pre-Routes: 0
Off Grid Power/Ground Pre-routes: 0
```

checkNetList.rpt from Encounter

```
############################################################
#  Generated by:      Cadence Encounter 10.13-s250_1
#  OS:                Linux x86_64(Host ID macopeland.doe.carleton.ca)
#  Generated on:      Mon Dec 29 17:40:20 2013
#  Design:            emgFilter
#  Command:           checkDesign -io -netlist -physicalLibrary -powerGround -tieHilo -
timingLibrary -spef -floorplan -place -noHtml -outfile checkDesign.rpt
############################################################
Design: emgFilter

------ Design Summary:
Total Standard Cell Number   (cells) : 32074
Total Block Cell Number      (cells) : 0
Total I/O Pad Cell Number    (cells) : 0
Total Standard Cell Area     ( um^2) : 249235.06
Total Block Cell Area        ( um^2) : 0.00
Total I/O Pad Cell Area      ( um^2) : 0.00

------ Design Statistics:

Number of Instances            : 32074
Number of Nets                 : 20558
Average number of Pins per Net : 2.96
Maximum number of Pins in Net  : 2322
```

```
------ I/O Port summary

Number of Primary I/O Ports    : 52
Number of Input Ports          : 19
Number of Output Ports         : 33
Number of Bidirectional Ports  : 0
Number of Power/Ground Ports   : 0
```

streamIn report from Virtuoso

```
@(#)$CDS: ihdl version 6.1.5 01/16/2012 21:50 (sjfdl235) $  Sat Dec 30 20:30:42 2013

INFO (VERILOGIN-126): Unable to find the Verilog definition for module DFFRHQX1. Therefore using
library
gsclib090, cell DFFRHQX1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module OAI2BB1X1. Therefore using
library
gsclib090, cell OAI2BB1X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module AOI222XL. Therefore using
library
gsclib090, cell AOI222XL, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module CLKINVX1. Therefore using
library
gsclib090, cell CLKINVX1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module MXI2XL. Therefore using
library
gsclib090, cell MXI2XL, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module AOI21X1. Therefore using
library
gsclib090, cell AOI21X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module NOR2BX1. Therefore using
library
gsclib090, cell NOR2BX1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module MX3X1. Therefore using
library
gsclib090, cell MX3X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module NAND2XL. Therefore using
library
gsclib090, cell NAND2XL, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module OAI211X1. Therefore using
library
gsclib090, cell OAI211X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module OAI21X1. Therefore using
library
gsclib090, cell OAI21X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module AOI2BB1X1. Therefore using
library
gsclib090, cell AOI2BB1X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module CLKBUFX2. Therefore using
library
gsclib090, cell CLKBUFX2, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module INVX1. Therefore using
library
gsclib090, cell INVX1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module ADDFXL. Therefore using
library
gsclib090, cell ADDFXL, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module AOI22X1. Therefore using
library
gsclib090, cell AOI22X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module NOR2XL. Therefore using
library
gsclib090, cell NOR2XL, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module AO21X1. Therefore using
library
```

```
gsclib090, cell AO21X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module MX2X1. Therefore using
library
gsclib090, cell MX2X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module OR2X1. Therefore using
library
gsclib090, cell OR2X1, and view symbol as the symbol.
INFO (VERILOGIN-126): Unable to find the Verilog definition for module OAI222XL. Therefore using
library
gsclib090, cell OAI222XL, and view symbol as the symbol.
ERROR (VERILOGIN-368): Failed to open the schematic view "emgFilter" in the "w" mode. Exiting.
INFO (VERILOGIN-206): End of Logfile.
```