Reducing Library Development Cycle Time through an Optimum Layout Create Flow

Rituparna Mandal, Dibyendu Goswami, Arup Dash
Texas Instruments India Limited

Abstract— *One of the major roadblocks in reduction of library generation cycle time is the layout generation phase. The two methods of doing automatic layout generation are synthesis and migration. The tools that are available for layout generation, each have it's own limitations. This paper describes how we have developed an integrated methodology for generating standard cell layouts using synthesis and migration. The placement engine of the synthesis tool was replaced by the Simulated Annealing based placer as well as the routing engine of the synthesis tool was made more intelligent and robust by using our own algorithms. The migration flow was also enhanced to suit requirements that were specific to ASIC cell libraries. This paper also presents the strategy we developed of an optimum combination of synthesis and migration for reducing the cycle time for generation of cell layouts. This strategy has enabled us to remove the bottleneck of the layout generation cycle time. The paper also touches upon how we have extended the flow to handle the "what-if" experiments that are carried out at a library definition phase.*

1. INTRODUCTION

With the increasing number of applications demanding ASIC solutions, there is a growing need for ASIC libraries to be developed to address different aspects of cell library. These include requirements like high performance, high density, low power etc. This requires a number of libraries to be delivered for a particular technology node, which are optimized and tuned for different end applications. So, to take advantage of the latest technology shrink, it is very important that the cycle time of the overall library generation process is reduced as far as possible. One of the major bottlenecks in this process is the layout generation phase which traditionally has been manual. In order to bring down the layout generation cycle time, we have come up with an integrated flow for generating layouts, which is an optimal combination of layout synthesis and migration.
Existing solutions and their corresponding problems:

- There are many chip level techniques for placement and routing [1]. However these cannot be directly applied to cell level layout generation where we deal with layout geometries.
- Tools for cell layout synthesis [2][3][8] are available. However none were found to give us layouts of handcrafted quality. In the synthesis flow, the placement engine specifically has the following drawbacks
  - It applies the same weighted cost function for cells of all complexities, which leads to sub optimal placements for many cases.
  - It does not take care of any reliability considerations such as latch-up.

  The routing phase has a number of limitations also as described in [4].
- Layout migration [9] tools are also available. But these do not handle all the standard cell level migration requirements. Particular drawbacks are :
  - Migration focuses on maintaining the topology of the layout that results in increase in either cell height or width (Increase in cell height is not acceptable for standard cell architecture under any condition).
  - Transistor specific sizing cannot be handled automatically through the migration flow (which only performs a linear scaling on all transistors)

We have developed a methodology to generate layouts of handcrafted quality for cells of all complexity. By layout quality, we imply minimum area layouts with minimum wire-length. It also includes other factors like no unnecessary bends in layout geometries, minimized diffusion capacitance and other reliability considerations (Wide metal routing for electromigration, large number of well/substrate contacts to reduce latch-up and so on). Cell complexity is a measure of the number of routable nets in a layout.

In Section 2 of this paper, we will discuss on how our flow addressed each of the possible scenarios that may arise in an ASIC library. In the Sections 3 and 4 respectively we will elaborate on how the existing synthesis and migration flows were enhanced and redefined to be integrated into a single solution. Finally in Section 5 we will describe what the impact our layout create flow had on the whole library development process. In Section 6 we discuss our future work.

2. THE LAYOUT CREATE FLOW

The flow that we developed for generation of cell layouts is based on two techniques, which are synthesis and migration. This is illustrated in Fig1 with the different scenarios that may occur during the course of library development. In sections 2.1 to 2.3 we will elaborate how we have combined the enhanced synthesis and migration flows optimally to develop cell layouts for any library

within the shortest possible cycle time. In order to do this we have chosen three specific cases that are most likely to arise during any layout development phase for any given library.

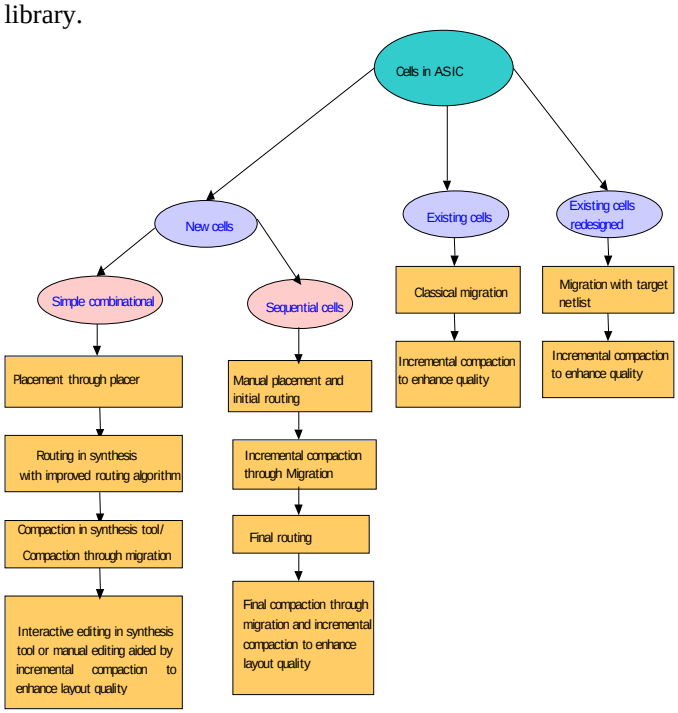

Fig 1

2.1 cells with new functionality:
For any new library there will be a set of combinational as well as sequential cells that either has a new functionality or designed newly. These layouts have to be generated with the netlist as the only input. In this case, the cells need to be taken through the synthesis flow. The compaction engine of the synthesis tool is not able to handle compaction of complex layouts like those of Multiplexers, Adders and some complex Boolean functions. For simple Boolean cells, or other basic gates like NANDs, NORs, ANDs, EXORs etc where the cell is not too complex in terms of routing congestion, it is taken entirely through the synthesis flow. However this may not always be completely automated to get production worthy layouts. In order to improve the layout quality we have introduced some breakpoints into the flow to enable a user to interactively edit through the synthesis tool. This prevents us from using a separate layout editor and in the process saves a lot of time. In cases of the more complex cells as mentioned above, the compaction step fails even if the routing is complete. For such cases, the pre-compacted layout is taken through the migration flow using the incremental compaction methodology as described in Section 4.1. However sequential cells, which require a lot of complex routes in the layout to meet timing, are created manually with the incremental compaction.

2.2 cells existing in the previous library:
This is a case of classical migration in which the layouts already exist for a previous library and need to be ported to the current technology which essentially implies migration from the old to the new technology. This is however possible only when the standard cell architecture does not change drastically from one technology to another. This migration is done by the conventional migration flow as discussed in Section 4.3.

2.3 redesign of the existing cells:
This is the case when the cell layouts already exist but as a result of redesigning the cells to meet the performance goals of the current library, the transistor sizes have changed. An ideal example here would be trying to migrate a cell from a high-density library to a high performance library. In this scenario, we migrate the layouts from the older technology to the new one keeping the target transistor sizes as one of the inputs to the migration flow. So, while the new design rules are enforced during compaction in migration, the transistor sizes are also enforced at the same time. This flow has also been described in Section 4.2.

3. THE LAYOUT SYNTHESIS FLOW

Synthesis automatically generates the cell layouts taking in the netlists and the design rules. The whole synthesis flow can be broadly classified into three main operations: Placement, Routing and Compaction.

The foremost operation that is done in the Synthesis flow is the placement, where all the mosfets in the cell will be placed in such a manner that the final area of the layout will be the minimum whereas the routing congestion will be less. In the next section we have discussed about the placer which was developed as a solution to overcome the existing placement problems. Next step is routing where electrical connectivity is provided to the different signal as well as power nets. We have developed our own routing algorithm described in [4]. The last step of the synthesis flow is compaction. This is a process of removing extra spacing in the layout as much as possible while enforcing the design rules.

The Placer:
High-density libraries need optimal transistor placement because of less routing resources. A bad placement will lead to larger layout area or an un-routed solution or compaction failure. The area of the layout depends mainly on the abutment of transistors. Abutment reduces transistor source/drain diffusion area and hence cell-width by merging same diffusion nets of adjacent transistors [2]. However maximal abutment does not always assure the best layout for routing intensive cells and may even result in unroutable or routing congested solution, causing subsequent compaction failure. Thus, in addition to abutment cost, we also need to consider a cost for gate alignment that will enhance better routing.
The placer developed is based on Simulated Annealing [5][6], a sequential optimization method, which

stochastically simulates the behaviour of a slowly cooling physical system, arriving at an orientation with the lowest cost. Simulated Annealing works in the following way:

```
evaluate(initial state);   //cost of initial state
 T = very high value     //initial temperature
 while(! Terminating condition) {
 current state = random change (initial state);
 evaluate (current state);
   if(Cost_current  <= Cost_initial ) {
    initial state = current state;
    }
  else{
     if(e^(Cost  initial - Cost current) / T  >
       a random number between [0,1])
         { initial state = current state ;}
      }
  reduce T ;
  }          // end while
        optimal state = initial  state;
The most popular terminating condition is T ~ 0.
```

For applying simulated annealing we need an efficient and accurate cost function to evaluate a placement. The cost function should include estimated cell width, wirelength and congestion. The placement module should be able to give optimal placement for wider variation of cells within a finite time. We designed the cost function to take care of the above layout aspects. We have assigned different cost scales based on cell complexity and have defined normalized acceptance probability that can work irrespective of cell size. We have used iterative simulated annealing to avoid locally optimal solution. We could have achieved the same by reducing the temperature at a slower rate, which would mean that it would take a huge number of iterations to reach the minima. Hence we opted for the iterative solution which is illustrated in Fig 2. Finally we have adjusted all the cost scales for different complexity.

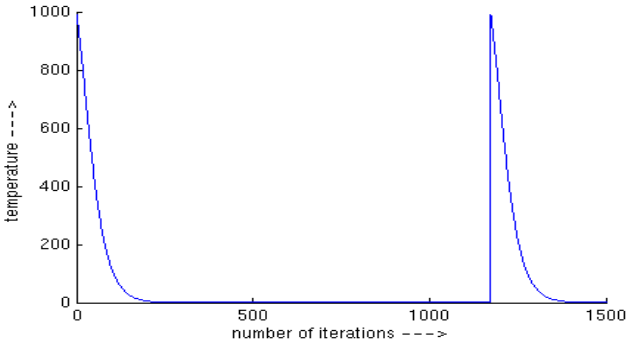

Fig2: annealing schedule

The cost function [7] that we have defined is the combination of four costs:

- *Area Cost*
- *Interconnect Cost*
- *Gate Alignment Cost*
- *Congestion Cost*

3.1 Area Cost

Area cost is a factor of estimated diffusion area. For a particular placement, diffusion area estimates what the cell width is going to be after compaction. Cell width is determined by Max[$PDIFF_{Width}$, $NDIFF_{Width}$](Fig3), assuming edge routings do not increase cell width. Apparently it may appear that Min[$PDIFF_{Width}$, $NDIFF_{Width}$] does not affect the layout quality. But by reducing it we can utilize the space for placing well/substrate contacts. We have defined area cost (A) as:

$A = N^* Max[PDIFF_{Width}, NDIFF_{Width}] +$
$Min[PDIFF_{Width}, NDIFF_{Width}] + Cost_{Tie}; \quad N>>1$

N is a relative scale to give higher preference to the first factor in the above equation.

$Cost_{Tie}$ is a factor to give higher preference to placement of well/substrate contacts that lead to lesser layout area.

The estimated layout width is:

Cell layout width = Max [$PDIFF_{Width}$, $NDIFF_{Width}$] + 2*g.

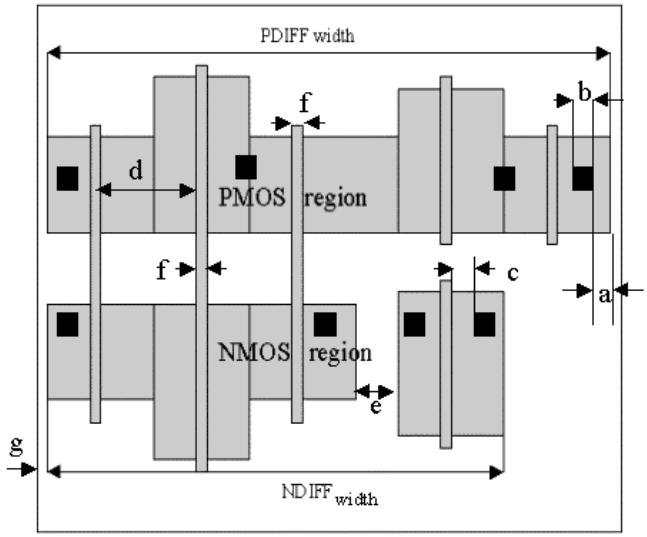

Fig 3: diffusion widths

For high-density libraries, abutment cost is the most important factor in placement evaluation to determine an area optimal placement.

3.2 Interconnect Cost

This estimates the length of wire needed to route the layout. This cost plays an important role to find an optimal placement for routing intensive cells. Routing cost estimates the Manhattan interconnect length of a placement. It is a linear function of

- Metal wire cost ($Length_{metal}$)
- Poly (polysilicon) wire cost ($Length_{poly}$)

Interconnect Cost= $Scale_{metal}$*$Length_{metal}$ + $Scale_{poly}$ *$Length_{poly}$

These two costs are considered because metal and poly differ in resistivity and run in different level. For high density libraries poly routing under power bus is preferred over metal ($Cost_{poly} < Cost_{metal}$) but for high performance library metal is preferred ($Cost_{metal} < Cost_{poly}$) because of lesser resistivity. Most of the gate to gate routing however

is done through poly in order to avoid use of contact (adds routing congestion in channel).

3.3 Gate Alignment Cost

Alignment of pmos and nmos having the same net for the gate increases routability of a layout [2]. For smaller cells misalignment leads to unroutable solution but larger cells can compromise misalignment for gaining in area.

To connect unaligned and far apart gates poly is less preferred for it's high resistivity. Again metal wire is also not preferred because it needs a poly contact which is bigger in size. To avoid misalignment, we have included alignment cost. This cost adds some penalty for each misalignment. For a particular placement if we consider the position of transistors as different indices of an array, then for n-th index if gate of the pmos transistor is not the same as the nmos transistor then we call this a misalignment. Experimentally we found a misalignment is having almost equal cost as K times of an additional diffusion gap i.e we can compromise a misalignment for removal of K diffusion gaps. K is an integer independent of technology.

Cost $_{alignment}$ = K*[(minimum distance between gates of two consecutive transistors with a diffusion gap in between) - (minimum distance between gates of two consecutive abutted transistors)]

i.e. Cost $_{alignment}$ = K*[{2* (a + b + c) +e} – d] (ref :Fig 2)

3.4 Congestion Cost

This cost represents amount of routing congestion in the layout. Congestion cost estimates average number of wires passing through vertical cross section of layout. Congestion cost plays a major role for routing intensive cells as cell height is fixed and a fixed number of wires can pass through a layout cross section. Source, gate and drain of each transistor are the points of cross section. If a wire (poly/metal) runs from the source of a transistor to the drain of other transistor then congestion at each point of all the transistor sitting below the wire is incremented by one.

Congestion cost is a combination of two factors:

- Average congestion (Cong$_{av)}$.
- Deviation from average congestion at each point.(Cost$_{devCong}$).

Average congestion takes care of less congestion in the layout. Less congestion deviation from the average helps to avoid locally congested layout. If the congestion is very high for a sample point it may lead to compaction failure. Cost$_{devCong}$ is the sum of deviations for each sample point. For example: If we consider 3 sample points with congestion 2,4 and 6 respectively and for another placement congestion are 4,4 and 4. For both cases Cong$_{av}$ is same(=4) but the later one can easily be compacted. For the first case Cost$_{devCong}$ is 2 where it is 0 for the next case.

.

3.5 Overall Cost Function:

$$cost\ (placement)\ \{return(Scale_{abutment} * Area\_Cost() + Scale_{interconnect} * Interconnect\_cost() + Scale_{alignment} * Alignment\_cost() + Scale_{congestion} * Congestion\_cost\ ());\}$$

For high-density libraries $Scale_{abutment} > Scale_{alignment} > Scale_{interconnect} > Scale_{congestion}$. These cost scales are depending on cell complexity. We calculate the cell complexity as the sum of the number of transistors connected to each net. For simple cells number of nets is less and for complex cells number of nets is large, which give an estimate of cell complexity. Relative weightage of abutment scale over all other cost scales is high for simple cells. We have preferred higher weightage of alignment cost for simple cells to avoid misaligned gates. We have set very high interconnect scale and congestion scale for routing intensive cells.

In this way we have taken care of the layout aspects related to placement of all kinds of combinational cells in the ASIC core cell library. Basic routing aspects of the cells are considered during the placement stage, like gate alignment etc. So the placer that we developed is in tune with the router to give better layout solutions. An example of a placement, which results in an un-routed solution, is shown in Fig 4.

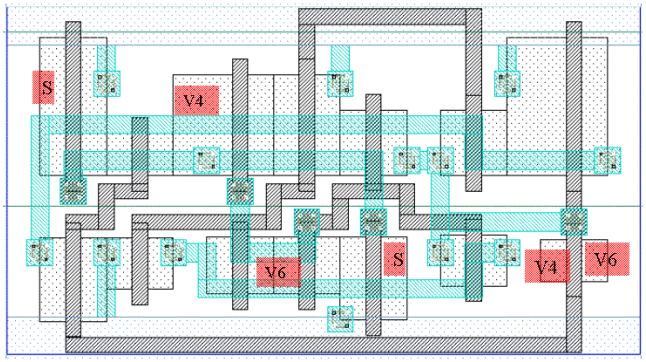

Fig 4: Sub optimal Placement with un-routable nets (V4, V6, and S)

With our placement solution the above placement changes to the one shown in Fig 5

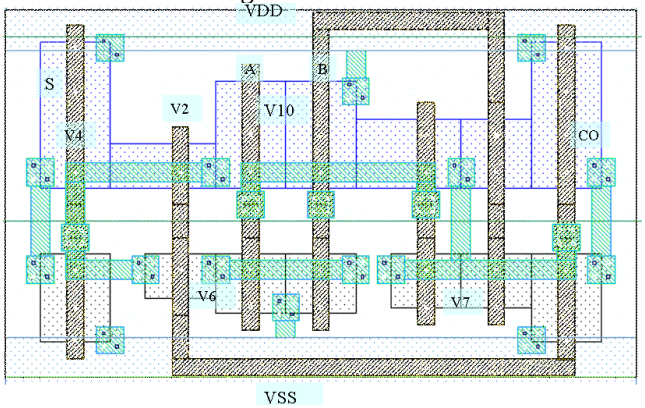

Fig 5: Optimal placement solution

4. THE MIGRATION FLOW

Migration is a process of porting the cell library from one technology to another when there is change in layout design rules but the topology remains more or less unchanged. The inputs to the migration tool are the cell layouts of any technology and the design rules and the standard cell architecture information for the target technology.

The constraint for doing automatic migration is that the tool cannot handle any change in the layout topology during migration. So if we want to automatically migrate the layouts, we will get good results in terms of layout quality only if the migration is possible without any changes in the positions and shapes of the geometries. However if we want to take advantage of any new design rules or cell architecture changes that will lead to a change in the way the transistors are placed or routed, then it will not be possible to do it through an automatic migration flow. This is even if the cells in any particular library can be reused for the next one. For example let us say we wanted to migrate layouts from a high-density library to a high performance library which is characterized by bigger transistors but a free routing track under each power bus which will be used for routing at chip level. This will not be possible through automatic migration.

Due to the above reason, we have observed that automatic migration is not best suited for ASIC standard cell library layouts. So we have tailored the migration flow to be used according to our requirements. The way we have done this is through the following:

1. Migration used as an incremental compaction engine
2. Migration used with target netlists
3. Migration used in the conventional way

4.1 INCREMENTAL COMPACTION

As we have already mentioned, migration indicates porting of layouts from one technology to another. This will be applicable if and only if the cells are existing in a particular library and are required for the next library. In many cases for a new library completely new cell functionalities may be defined or a cell that was existing in the previous library may be completely redesigned. In such cases, if the cell were very complex in nature, the layout in all likelihood would be done manually. Complexity is in terms of optimum placement of transistors, which requires a lot of manual interventions and also complex routing to get minimum area. This is where incremental compaction comes into the picture.

We have developed a methodology for incremental compaction which allows us to take advantage of the features of the compaction engine of a migration tool as well as get layouts of hand crafted quality at the same time. The migration tool that we have adopted for our flow has an interface to the layout editor that we use. We have exploited this feature to use the migration tool essentially as a compaction engine rather than a conventional migration tool. The way we have done this is described below.

Taking the netlist for the new cell we will place the transistors with the respective sizes as defined in the netlist as optimally as possible. Then an initial level of routing will be done manually also. However while doing this we need not worry about the design rules with respect to the various layout geometries – like the width of the metal wire, or the contact widths, or the metal overlap of contact etc. Having done the preliminary routing we migrate the layout to the target technology. So essentially we do an incremental migration of the layout, that is, we use the compaction engine of the migrator to enforce the design rules on the layout. Then we do the next step of routing and again compact through migration. In this way we can complete the whole layout – getting the best of both handcrafted quality for placement and routing of the transistors and automatic compaction through migration. This result in a significant reduction of cycle time for generating those layouts compared to what it would take to do those same layouts completely manually.

4.2 MIGRATION WITH TARGET NETLISTS

Let us consider a case where a cell is present in a particular library and we want to port it to the new library but only after changing the transistor sizes to improve the cell performance. For such a case, we devised our own flow of enforcing the new transistor sizes by feeding in the target netlist to the migration tool. The technique that we follow is generic in nature. For a particular transistor in the layout, we find it's equivalent from the target netlist based on connectivity information. Then we resize the transistor accordingly. The way we automated this is shown in Fig6 below.

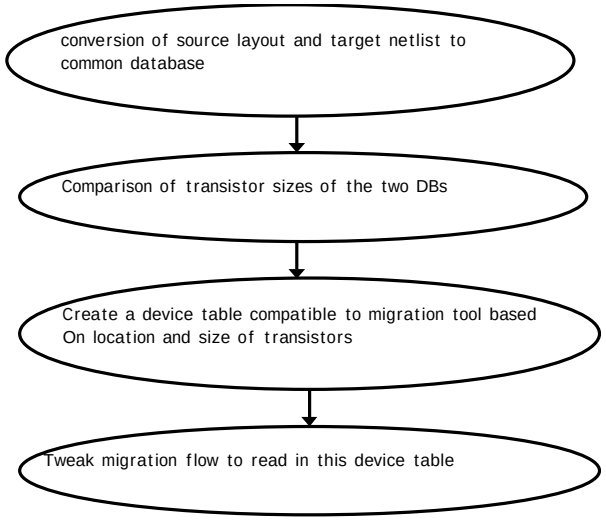

Fig 6: automatic target netlist reader

4.3 CONVENTIONAL MIGRATION FLOW

For large blocks or custom cells like datapath that have to be ported from one technology to another without any change in design topology are still most effectively done through the conventional migration flow. The output however may not be of the best quality and in many cases the area constraints may also not be met through this automatic migration flow. However the output that is obtained may be used as a starting point and may be manually edited sufficiently to get the quality of layouts that we want. This would still result in a significant reduction of cycle time as compared to doing the layout from scratch for the new technology.

5. IMPACT

The impact of our layout create flow is highlighted below:

1. Library development cycle time:
During several ASIC core cell library developments, we have used an optimum combination of both synthesis and migration as described in the previous sections to generate layouts for both high density and high performance libraries. Fig 7 illustrates the impact of our layout create flow on several ASIC libraries. The layout development cycle time was reduced by ~5X in LIB3 compared to LIB1. This reduction was after using the placement and routing techniques described in section 2.1 and 2.2. We deployed the complete layout create flow, using a combination of synthesis and different migration techniques wherever applicable for incremental releases of the same libraries which actually resulted in a further 2X improvement in the layout generation cycle time for LIB4.

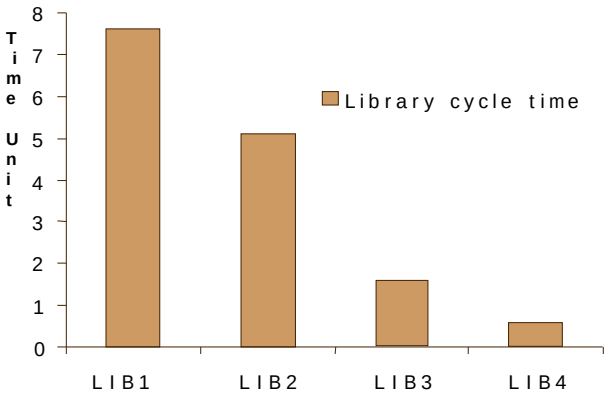

Fig 7: Reduction in library cycle time

2. Library definition and evaluation:
One of the most important aspects of library definition is the architecture evaluation phase. In order to do this and come up with the best possible architecture for a particular technology, number of experiments are performed with regard to layouts. To enable gathering of complete data for the above, automation in the layout generation process is absolutely essential. All architecture evaluations with respect to layouts are done using our layout create flow. In addition to architecture evaluation, all layout evaluations which help in library definition are also done using our flow. Some examples are given below:

- Evaluation to decide the optimum cell height as well as N-Well location.
- Critical path analysis to give feedback to fab on layout design rules in terms of impact on area.
- Determining area impact on usage of different transistors (characterized by different lengths) for the same technology node.
- Evaluation of using antenna protected cells in terms of impact on chip area.

6. FUTURE WORK

We plan to enhance the cell layout create flow even further by generating placement solutions for sequential cells which involve stacking and rotation of transistors. The migration flow will also be enhanced to make it more customizable.

References:

[1] Sadiq M Sait & Habib Youssef," VLSI PHYSICAL DESIGN AUTOMATION, Theory and Practice". Chapters 4 - 7. McGraw-Hill Companies - 1994

[2] Antun Domic, Digital Equipment Corporation, Hudson,"Layout Synthesis of MOS Digitial Cells", DAC 1990

[3] Chong-Leong, Ong Jeong-Tyng Li, Chi-Yuan Lo, 26th ACM/IEEE Design Automation Conference,"GENAC: An Automatic Cell Synthesis Tool"

[4] Sabyasachi S., Sornavalli R., Dibyendu G., Biswadeep C., "Minimizing Area and Maximizing Porosity for Cell Layouts Using Innovative Routing Strategies", VLSI Conference, 2001 Bangalore.

[5] Kirkpatrik, C.D Gelatt Jr and M.P. Vechhi, May 1983;"Optimization by Simulated Annealing", Science, Vol.220, No.4598, pp.671-680

[6] Carl Sechen, "VLSI Placement and Global Routing Using Simulated Annealing",Kluwer Academic Publishers, 1988

[7] Qinghong Wu and Thomas H. Sloane "CMOS Leaf-Cell Design Using Simulated Annealing", IEEE 1992

[8] Chao C. Chen, Shau-Lim Chow,"The Layout Synthesizer: An Automatic Netlist-to-Layout System", 26th ACM/IEEE Design Automation Conference

[9] Jon Levi,"Dreaming Up a New Methodology for Physical Migration of Hard IP", ISD Mag, May 1999