Automatic Design Scaling

by Omid Rowhani

Contents

List of Figures

List of Tables

Acknowledgments

First, my special thanks to Bob Brodersen, my research advisor, for his continuous advice and guidance throughout this project.

I also wish to thank Jan Rabaey for his input, Tom Burd for all his help as well as the other members of the group, students and staff.

To my dear parents, Abdul-Raouf and Bahereh, I dedicate this work. No words of mine can adequately thank them for their never ending love and support, for standing by me while thousands of miles away, for their confidence in me and their encouragement. Thank you.

1

Introduction

The fast progress towards ever smaller feature sizes renders many designs practically obsolete in a short period of time. Thus technology's rapid pace increases the importance and attractiveness of design reuse, but having to re-design the circuits each time the process changes is expensive. While the main challenge in the past was creating a chip with a single functionality and an acceptable performance, the emphasis is shifting towards entire systems on one chip which will be based on reusable cores. With this trend, the challenges and requirements of today's designer have changed too. A method, therefore, that will quickly and efficiently provide the designer with these blocks in the latest technology, without sacrificing performance and requiring large amounts of time and effort is highly desirable.

The goal of this project was to create such a methodology and to provide a tool which the designer can use to convert existing designs for use in the latest technology. Our

method is based on a program written in Cadence's Skill code, which performs a series of shrink and expand operations on the existing layout. These operations create a new layout that conforms to the new target design rules.

To demonstrate the operation of our program, and to study its advantages and shortfalls, we used an ARM8 microprocessor designed in a 0.6 micron technology and translated it to a 0.25 technology. This scaled down ARM8 will act as a core processor in the Berkeley Pleiades project [1]. The ARM8 is a low-power 32-bit RISC microprocessor fully implemented in static CMOS. It consists of a 5-stage pipelined Core and a branch predicting Prefetch Unit. The Prefetch Unit together with a double-bandwidth memory interface act to reduce the power consumption and the overall CPI. Since the memory interface's bandwidth is greater than the bandwidth requirement of the Core, the Prefetch Unit takes advantage of that by buffering the instructions in a prefetch buffer. Branch prediction is then used to remove some instructions before they are presented to the core. Thus, fewer instructions are passed to the core for processing. Pipelining of the instructions and the data ensure a continuous operation of the processor and the memory system.

The following chapters will describe in detail our scaling process, chapter 2 will present the details of our program and the steps needed to complete the scaling process. Chapter 3 will provide the results obtained from our experiment and we will end this work with our conclusions in chapter 4.

2

Scaling Methodology

In an effort to provide the designer with the capability to reuse existing blocks and to allow the main effort to be concentrated on the task of system design we present our method of design scaling. This simple method makes it possible to scale down existing designs in a quick and almost error free fashion for use in newer technologies. As with any other engineering task, we are faced with a number of trade-offs. The main trade-off facing us was that of simplicity versus performance as measured mainly by the final design area as well as by the number of acceptable violations which must be handled manually.

After a discussion that explains how we determined our scaling factor we will look more closely at the scaling method itself. Section 2.2.1 will explain the layout scaling process while section 2.2.2 will do the same for the schematic. The final sections will describe the process

of moving the design to the new technology and the verification steps needed to ensure that the design is functional.

2.1 Determining the scaling factor

Had the different technologies adhered to the lambda design rules or some other form of linear scalable rules, our task of scaling the design would have been extremely simple. This would have consisted of a simple program that linearly shrinks every geometry in the layout. Since this is not the case, and different rules scale differently between technologies, we need to determine a maximum scaling factor. This scaling factor will determine how much we can shrink the design and still comply with all the new design rules. Table 2.1 below shows how, for certain design rules, violations begin to occur as we start scaling down from 0.6 microns towards 0.25.

Rule Name	**Scaling To:**					
	0.25	**0.3**	**0.35**	**0.4**	**0.45**	**0.5**
Active Spacing	*	*				
Poly-Active Spacing	*	*	*			
N+/P+ Implant Spacing	*	*	*	*	*	
Contact Width	*					
Contact Spacing	*	*				
Contact-Active Spacing	*					
Via1 Width	*	*	*			
Via1 Spacing	*	*	*			
Via2 Width	*	*	*			
Via2 Spacing	*	*	*			

Table 2.1: Design Rule Violations For Different Scaling Factors.
(Violations are denoted by an asterisk and a shaded box)

As seen from the table, scaling down to 0.5 microns results in no design violations. Therefore, this can be achieved by a straight forward linear scaling of all the shapes by a factor of 1.2 ($= \frac{0.6}{0.5}$). However, attempting to scale beyond 0.5 microns results in the introduction of design rule violations with an increasing number as we scale to smaller and smaller sizes. This surely does not mean that 0.5 microns is the limit for scaling between these two technologies. A careful study of the design rules shows that by performing some selective expand and shrink operations on certain layers, we can safely scale the entire design to 0.35 microns and still comply with all design rules. For example, it is possible to overcome the poly to active spacing violation by further shrinking the poly traces. This is shown in figure 2.1. The initial step scaled the 0.6 micron wide poly to 0.35 microns. Since the minimum poly width is 0.25 microns, we can further shrink the poly line thus increasing the spacing to the active layer.

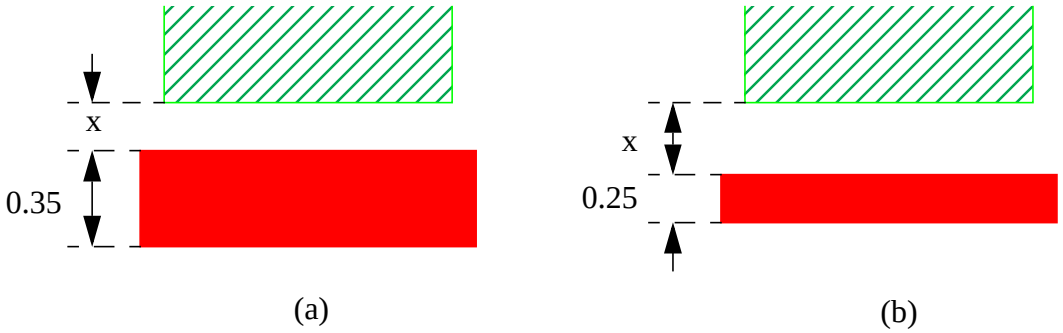


Figure 2.1: Poly-Active Spacing. *(a) x < minimum spacing (b) After resizing poly, x now satisfies minimum spacing requirements*

Using similar techniques the remaining violations shown for the 0.35 micron case can be resolved. This however is not the case if we scale down to 0.3 microns. In addition to the

larger number of violations that need to be addressed, no quick and easy technique for resolving them was found.

An important factor that can not be overlooked is the minimum design grid which defines the smallest dimension that can be used. Thus a minimum grid of 0.1, for example, will not accept dimensions that are not multiples of 0.1 such as 0.05 and the like. This limits the factor by which we can scale down the design. In our case, the minimum design grid allows scaling to 0.4, 0.35 or even 0.3 microns. However, scaling to 0.375 or 0.325 is practically impossible even if the design rules allow it. Such scaling will require shifting and expanding geometries and layers in order to align them onto the grid. This process is involved and computationally complex as we need to check for new violations that can be introduced as a result of such movements. We also must ensure that the minimum design grid is not violated for each of the scaled down geometries. Thus, there are cases where certain layers with certain dimensions need to be expanded or shifted somewhat to accommodate the minimum design grid.

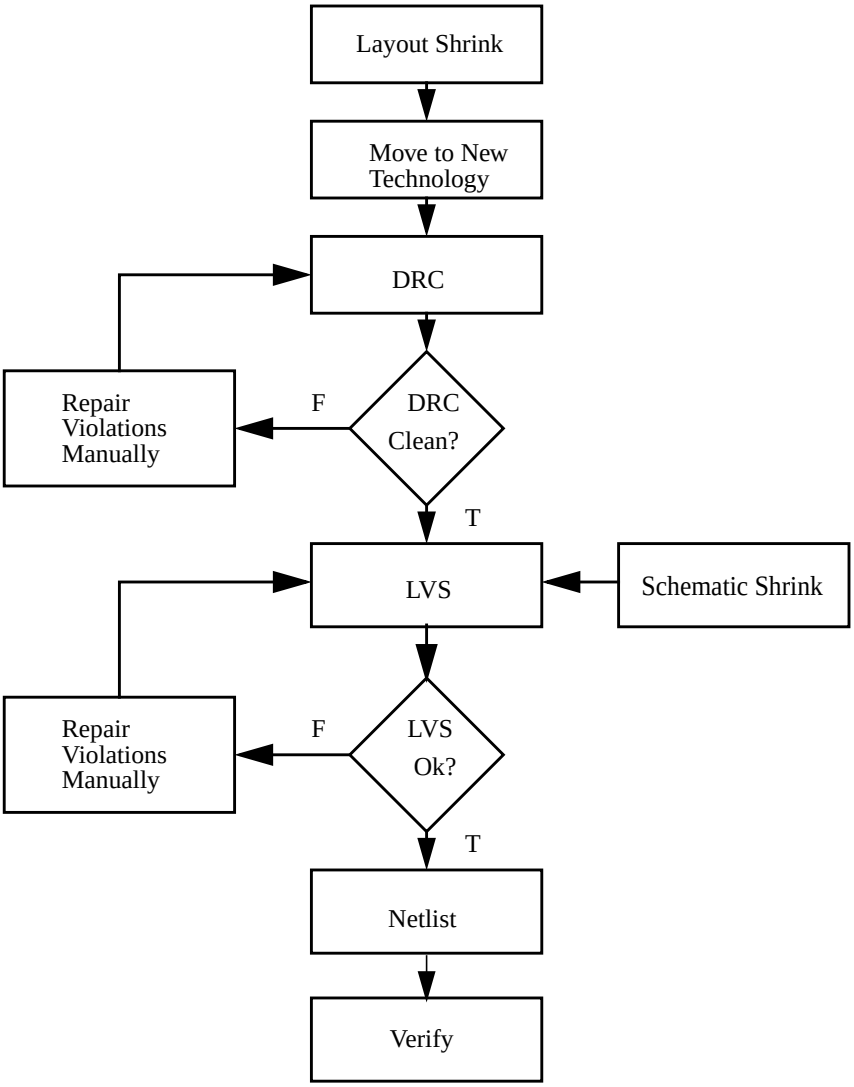


Figure 2.2: The Scaling Process

2.2 The scaling process

The scaling process created in this project is illustrated above in figure 2.2. Although the main objective is to scale down and convert the physical layout to the new technology, the schematic can not be ignored as it is needed for the verification stage. We therefore scale

down and convert both representations to the new technology as described in the following sections.

2.2.1 Layout scaling

By far the most challenging and time consuming part of this work was scaling the layout. The starting point was to study and compare the two technologies and determine the best scaling factor possible. This was described in section 2.1 above. The rest of the process is described in figure 2.3 and consists entirely of a program, written in Cadence's SKILL code. This program accesses the database, where the original layout resides, and through a series of shrink and expand operations transforms the existing geometries to their new dimensions.

2.2.1.1 Constant Geometrical Shrink

The constant geometrical shrink consists of the following steps:

1. Merge all overlapping shapes from same layer.
2. Shrink all rectangles.
3. Shrink all polygons.
4. Shrink all labels.
5. Report any other shape encountered.

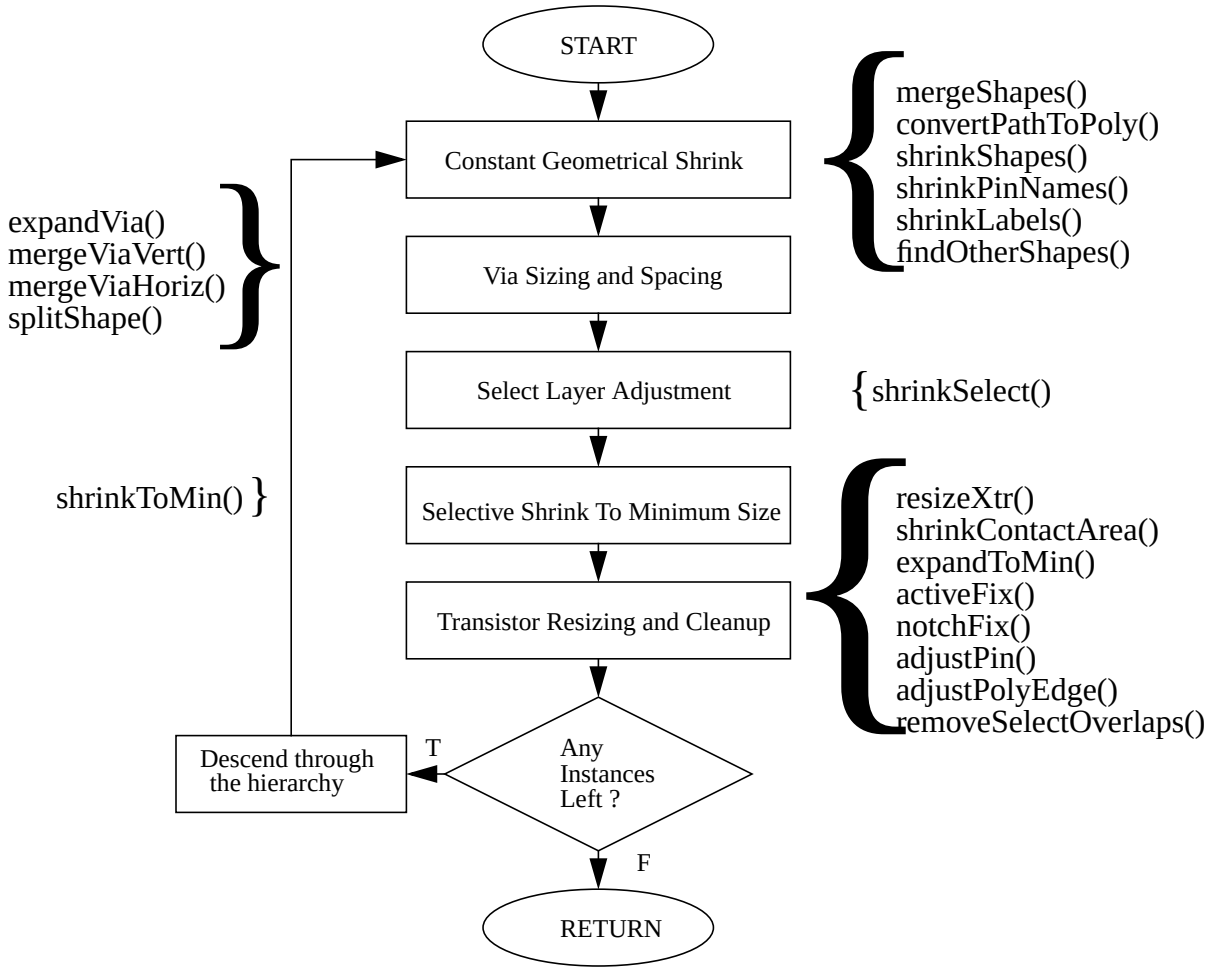


Figure 2.3: Layout Scaling Process

A merge of all overlapping shapes step is repeated several times throughout the program. As the name implies, this function merges all the different, overlapping shapes of one layer into one polygon. This is done to avoid a situation where adjacent shapes are scaled or expanded differently by the program causing them to become separated or create other violations.

This program is made to work with rectangles and polygons only. Any other shape (such as arcs, donuts, ellipses, etc.) are reported in the log file but are left untouched.

2.2.1.2 Via Sizing and Spacing

The constant scaling performed in the step above created via sizing and spacing violations. Direct scaling to 0.35 results in vias that are smaller than the required minimum size. Since less metal overlap of vias is required in the 0.25 technology, this issue is resolved by expanding the vias to their correct size.

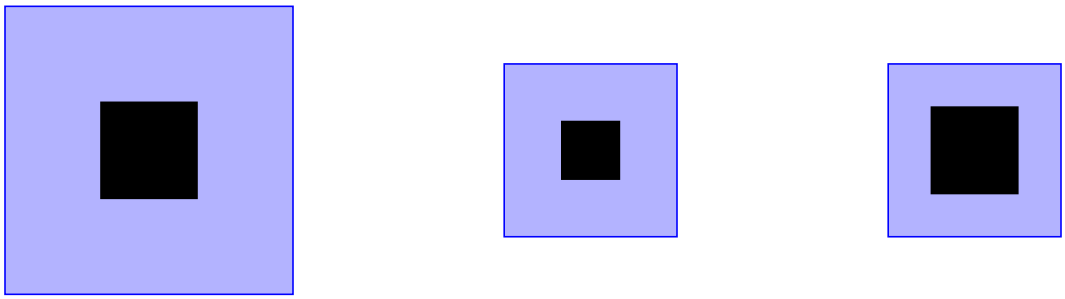


Figure 2.4: Via Scaling. *(a) Original 0.6 micron via (b) After linearly scaling to 0.35 and (c) after expanding the via to it's correct size*

This expansion, however, creates a spacing violation where vias are clustered together. To overcome the spacing problem, figure 2.5 shows how the vias are merged vertically into one long rectangle, then properly spaced individual vias are formed. This process is then repeated horizontally. As the figure shows, some of the vias are discarded since the additional spacing between them pushes one column and one row too far out thus violating the minimum overlap requirements.

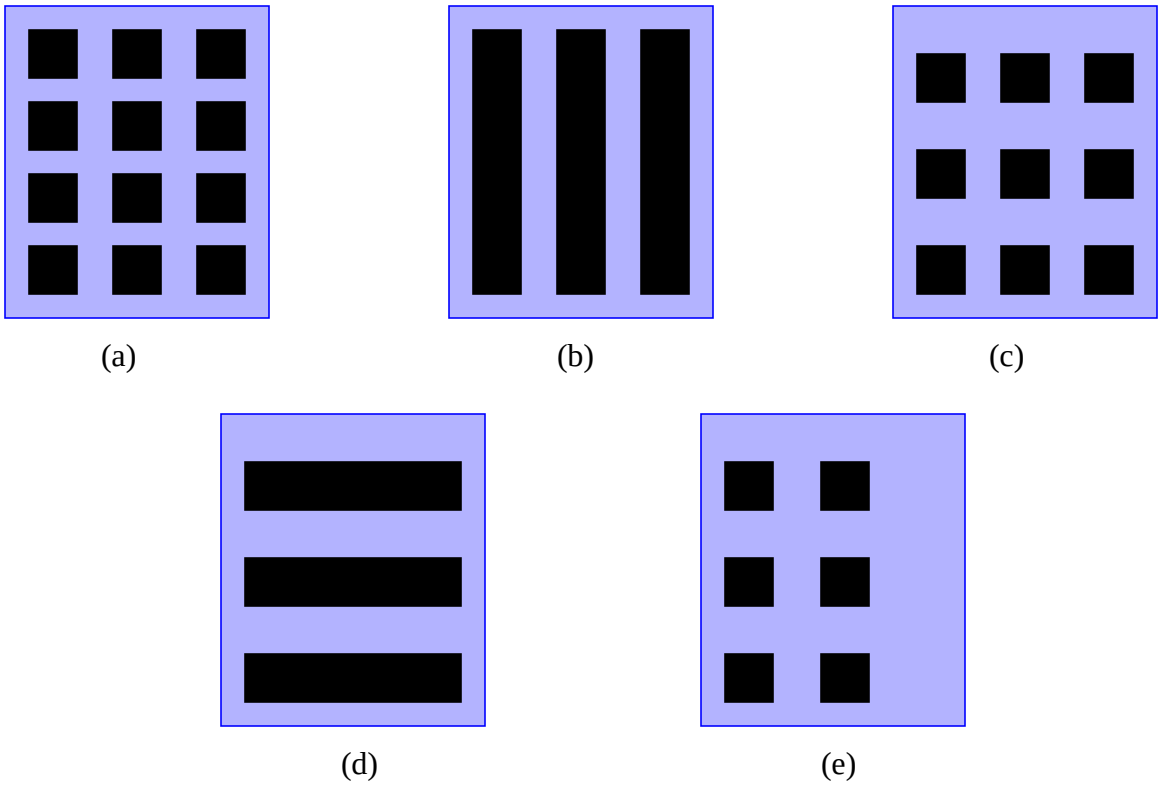


Figure 2.5: Via Array Spacing. *(a) Original via array with the vias spaced too close to each other (b) Vias merged vertically (c) vias spaced vertically (d) Vias merged horizontally (e) Final horizontal spacing*

2.2.1.3 Select Layer Adjustment

Table 2.1 shows how we begin to encounter select layer violations at 0.45 microns. The violation is a result of the relative increase in the spacing requirements in the new technology. However, we observe that at 0.35 microns the active overlap by the select layer is more than the minimum required. This enables us to move all the select layers inwards to create the adequate spacing and thus scale beyond 0.5 microns.

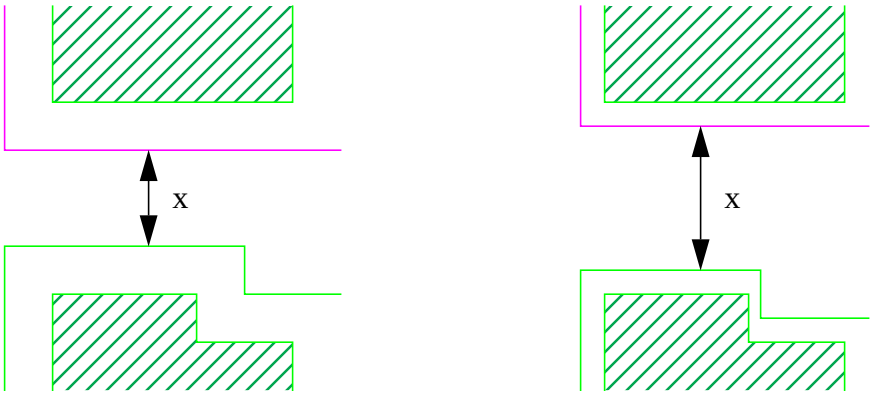


Figure 2.6: Select Layer Adjustment. *(a) Before the adjustment, x < minimum spacing (b) The adjustment removes the spacing violation*

2.2.1.4 Selective shrink to Minimum Size

The design we have at this point is a complete 0.35 micron equivalent layout. Since our target technology is 0.25 microns, we can further selectively shrink different layers and the transistors to their absolute minimums. Although the overall area of the layout will remain unchanged, we expect some reduction in the device and interconnect capacitance leading to better overall performance.

In this step certain poly and metal traces are reduced to their new minimum size. This is done only for those traces that where minimum size in the original 0.6 micron layout. The program always assures that the minimum design grid is not violated. In the case of round off errors it snaps all points and coordinates to the nearest design grid. This is true for all the scaling steps. It also ensures that no geometry is scaled in such a way that will place it off the grid. This case was observed when the metal layers were further scaled to their 0.25 micron minimums.

Due to the way this scaling is done (see figure 2.7), a minimum size metal trace can be placed halfway between the design grids on both sides. The program therefore increases the width of these lines and places them on the grid. Note however, that this expansion can not create violations since it is not expanded past its original size.

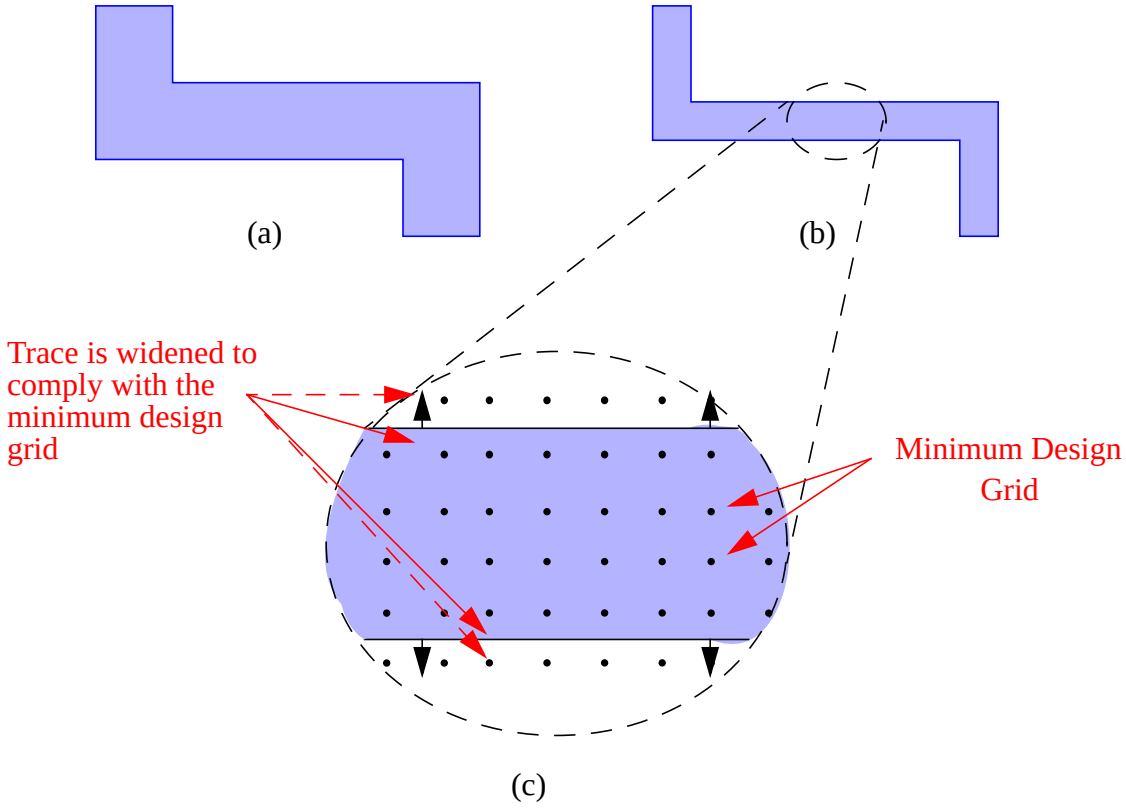


Figure 2.7: Grid Adjustment. *(a) Sample metal trace scaled to 0.35 (b) Further scaling of the trace to it's minimum size (c) The resulting grid violation caused by the further scaling and the widening to the nearest grid point*

A final step in this stage of the process was to repair any violations around contacts and vias. As figure 2.8 shows, the process of minimizing the widths of certain traces can result in changes around the vias and the contacts with less than the required minimum size overlap remaining.

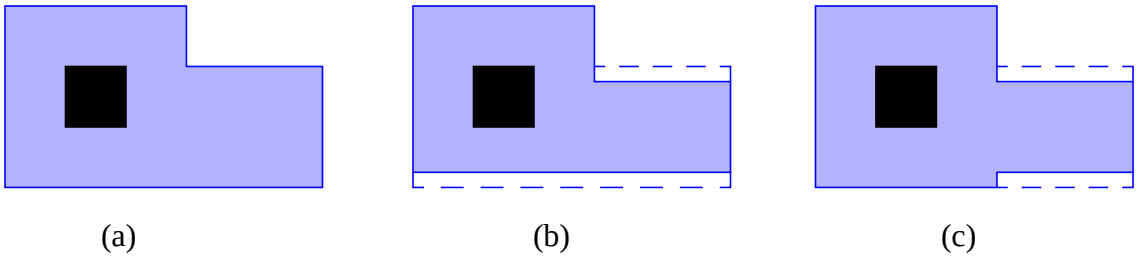


Figure 2.8: Via and Contact Repair (a) Before shrinking the line to minimum size (b) after the line shrink operation the metal no longer overlaps the contact as required (c) fixing the overlap violation

2.2.1.5 Transistor Resizing

The resizing done on the poly lines in the previous step has changed the length of all the minimum sized transistors. The width of all these transistors should therefore be adjusted accordingly in order to keep the design unchanged. This is done by identifying all the transistors and establishing their orientation. Once that is determined, the active is chopped and the width to length ratio is restored.

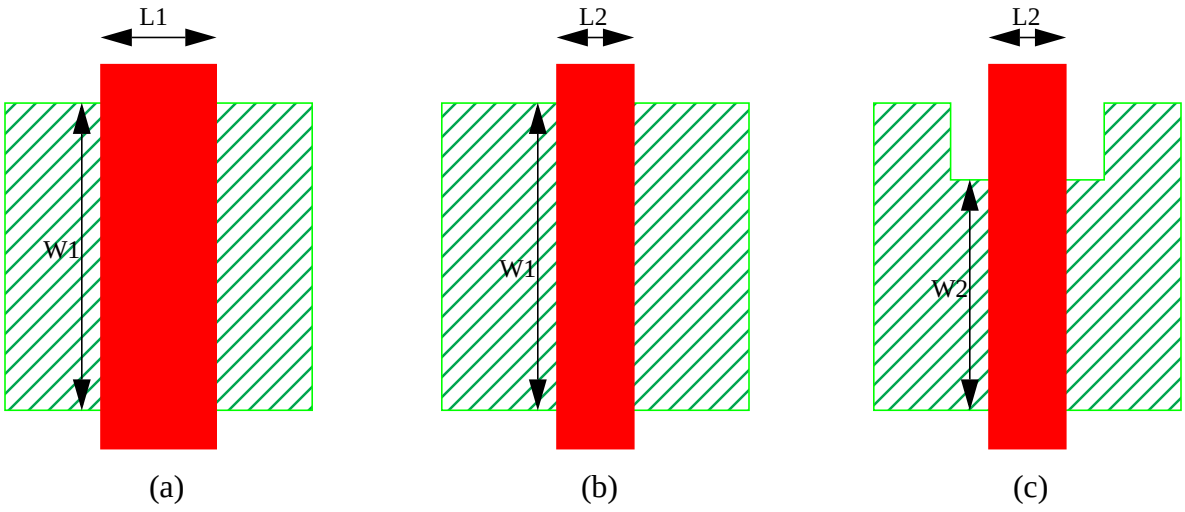


Figure 2.9: Adjusting Minimum Sized transistors. *(a) Transistors scaled to 0.35 microns (b) Poly width reduced to minimum size (c) Width adjusted appropriately*

The remaining, non-minimum sized transistors are treated in a similar way. First, the length is adjusted by trimming the poly layer. Next, the underlying active layer that determines the transistor's width is chopped as was done for the minimum sized devices.

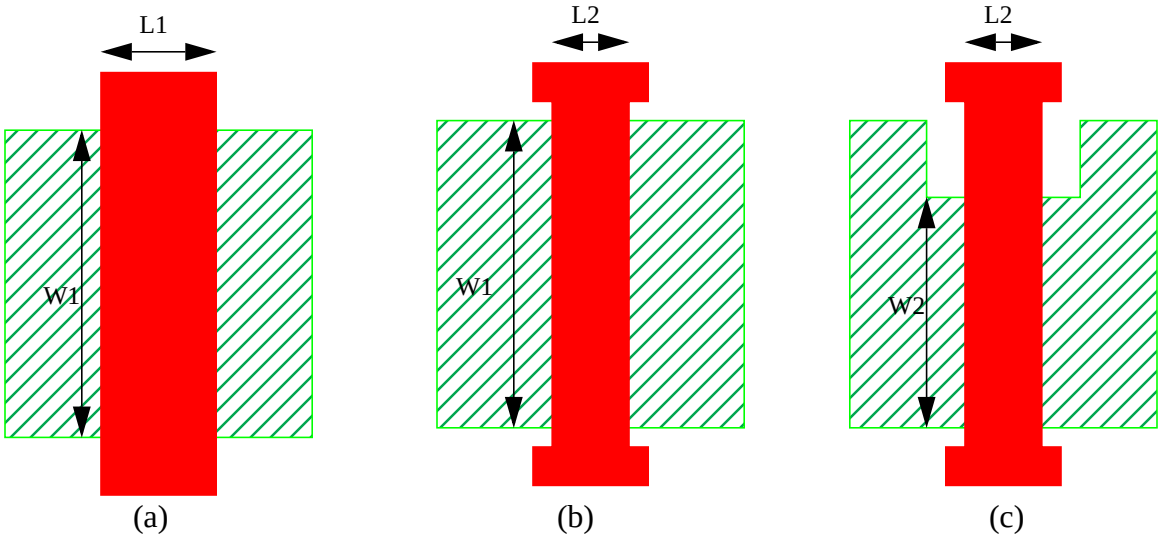


Figure 2.10: Non-Minimum Sized transistor Adjustment. *(a) Original transistor (b) Length adjustment (c) Width adjustment*

2.2.1.6 Cleanup

Although the bulk of the scaling is now complete, few fixes and adjustments are still necessary before we can continue down the design hierarchy. The first step in this stage is to cleanup all the contacts and the vias and remove the unnecessary overlaps. The many shrink and expand actions performed on the different layers resulted in many notched figures. The program therefore goes through each cell and fixes any notches by either filling them in or trimming them. Furthermore, the pin layers were left untouched throughout the entire process whereas the underlying drawing layers were altered. We therefore adjust the pin layers to correspond with the underlying drawing layer in this stage. Since the pin information is not saved when the design is

transferred to the new technology, we save all this information in a text file and later restore it directly to the new design.

2.2.1.7 Recursive descend through the hierarchy

Since the advantages of keeping the design hierarchical far outweigh the disadvantages it introduces, the program was written to preserve the design's hierarchy. Scaling a flattened design is less involved and simpler to perform. However, doing so proves to be computationally inefficient, time consuming, and may result in a design that is at best difficult to modify or fix at a later stage. Many cells are repeatedly used in a large design, such as this ARM processor, particularly in a datapath. This repetition can not be recognized in a flattened design, resulting in the same cell being scaled each and every time it appears. Moreover, since no process is perfect and error free, it is highly desirable to facilitate easy repair and modifications. Such fixes need only be applied once to the basic cell in a hierarchical design whereas it may prove to be impossible to perform in the flattened case. This will be clearly demonstrated when we discuss the verification process and the results obtained in the following sections and chapter.

Using a recursive algorithm, the steps described in the subsections above are repeated for each cell in the design. As each cell is processed, the program checks for any instances it may contain and proceeds to descend into them and scale them.

One of the difficulties brought about by the hierarchical nature of the design is violations across the hierarchy or across instances. Since each instance is scaled individually, it's respective placement in the overall design is not considered. This may create situations where two

adjacent cells, or cells nested within each other, produce spacing or other design rule violations. This problem is addressed for the case of adjacent cells and cells that are nested up to one level deep. This is accomplished by making a copy of each layer from the child cell in the parent cell. Next, patches are created in the parent cell where violations occur. For example, if a small notch appears on the boundary of two cells, the patch will be a small rectangle that fills in that notch. Once all the patches are created the copied layers are discarded. Since these patches are most likely small rectangles the design should not be checked in the hierarchical mode. In a hierarchical mode these rectangles may stand out as violations themselves since they do not always obey the minimum design rules.

2.2.2 Schematic scaling

The schematic scaling process is shown in figure 2.11 below. This process is much simpler than the layout scaling one due to obvious reasons. Whereas the layout is a physical design which places constraints on the connection between the devices no such limitation exists in the schematic. Our only concern is the device dimensions which are represented as properties of each transistor in the design.

The program traverses the design hierarchy modifying the length and width of each transistor it encounters. These dimensions are scaled by a constant factor of 2.4 ($=\frac{0.6}{0.25}$). Since the two technologies are separate and come with their own device libraries it is important to reference the new devices. This is the stage labeled as “point to new master” in figure 2.11 below. All instances that are transistors, I/O pins, vdd, gnd and any parametrized cells (termed “list A” in the figure) are replaced with the equivalent instance from the new libraries.

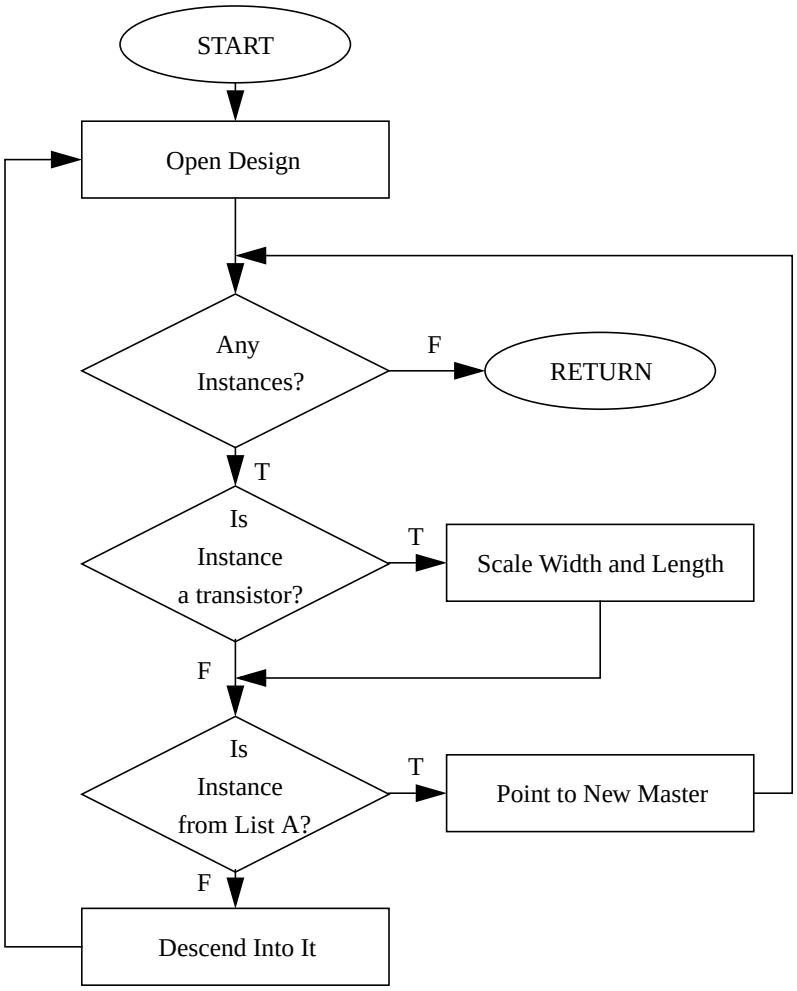


Figure 2.11: Schematic Scaling Process. *(List A refers to all transistors, I/O pins, V_{DD} , Gnd, and any parametrized instance)*

2.2.3 Moving To The New Technology

Although both the layout and the schematic have been scaled to the new dimensions, the layout still requires a few more adjustments before its scaling is complete. As we described in the previous section, the schematic was altered to point to the new devices that correspond to the new technology. This was not done yet for the layout. We therefore stream out the layout to an intermediate GDSII stream format and then stream it back into a Cadence database using the new technology layer numbers. As a final step we restore all pin information. A separate program

looks for the pin information that was saved during the scaling process in text files. These pins are restored and the scaling process is complete.

2.2.4 Design Verification and Testing

The verification and testing process is carried out as it is for any other design for which a layout and a schematic exist. For the layout, a DRC (Design Rule Check) is run and all violations are corrected. To maintain program simplicity no attempt was made to guarantee an error free scaling. Although the program produces a correct layout under most circumstances, certain cases may introduce rule violations. These violations are easier to correct manually thus avoiding program complexity. Further justification for this approach is given in the following chapter where the results obtained and the DRC violations encountered are explained.

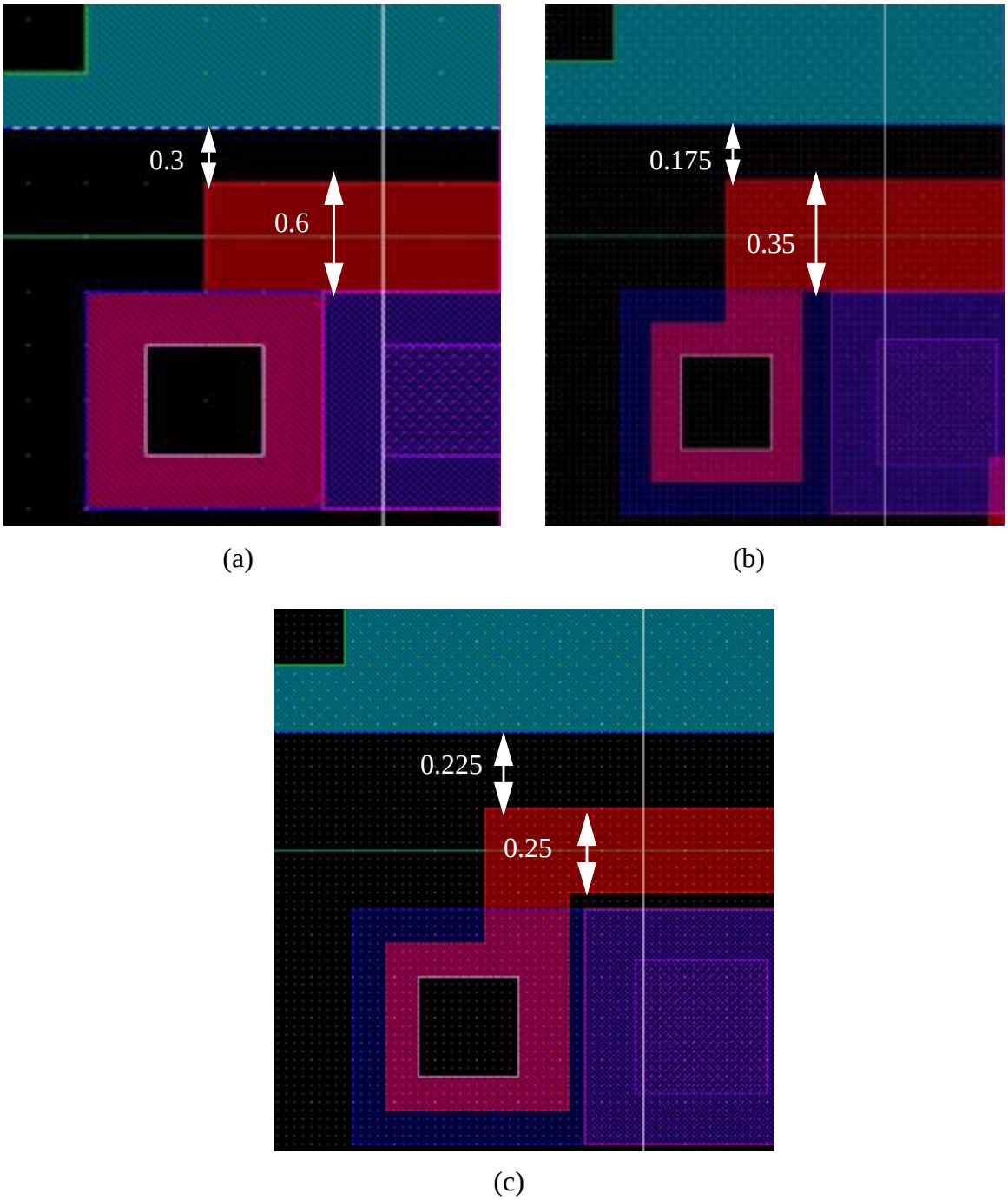
A final check before we go on to netlist our design is an LVS (Layout Versus Schematic) check. Once that is complete and a netlist is produced we proceed to verify the circuit's operation using a circuit simulation tool.

2.3 Scaling Example

This section will demonstrate through examples some of the problems that were mentioned earlier in this chapter. These scenarios are generated by disabling some of the functions in the program. In this way we can see the actual errors and how they are corrected.

2.3.1 Poly Active Spacing

Figure 2.12(a) shows a layout section where a poly line runs at a minimum distance to the active layer.



(a) (b)

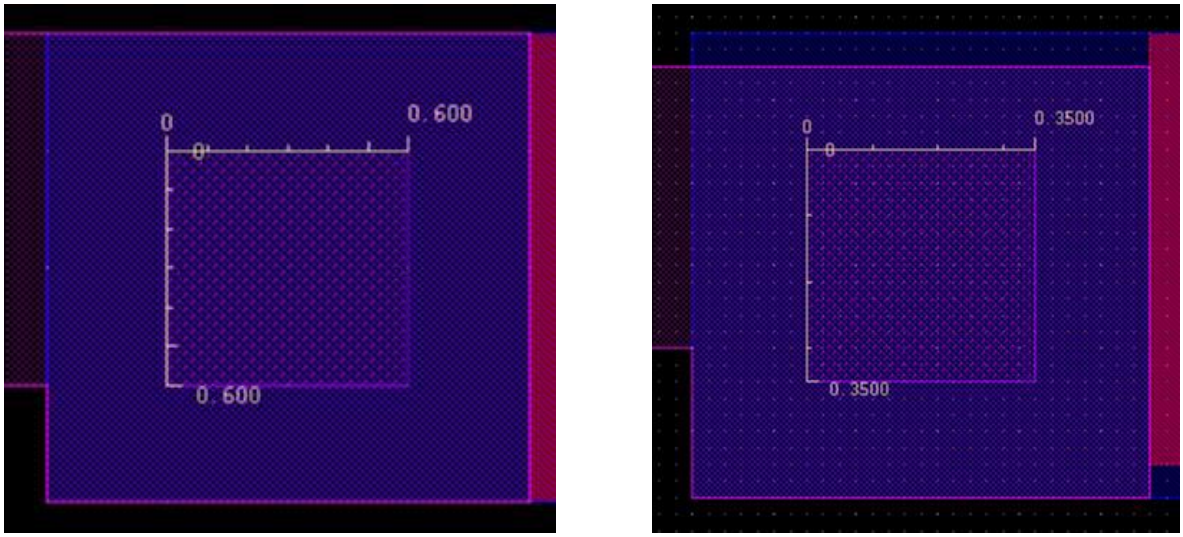
(c)

Figure 2.12: Scaling of the Poly-Active Spacing

Part b of the same figure shows what happens if we disable the shrinking of the poly to the minimum 0.25 micron size. In this case the 0.175 micron spacing to the active layer is not adequate to satisfy the new design rules. Part c shows how the spacing increases to 0.225 microns once the poly line is further scaled to 0.25 microns. The poly to active spacing now meets the design rule minimums.

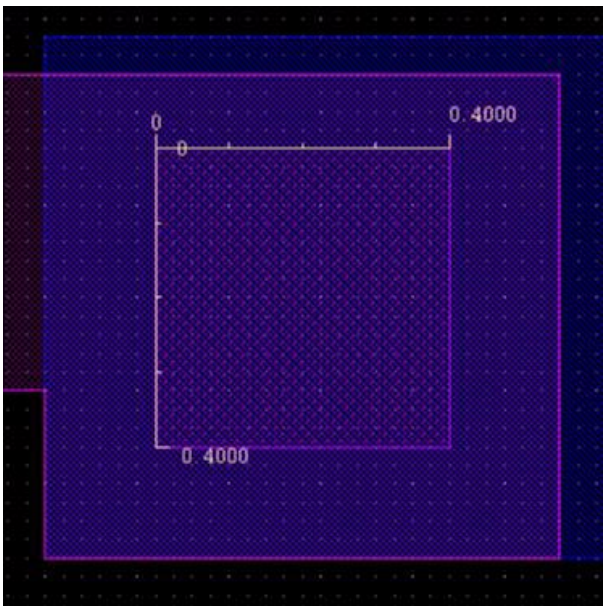
3.3.2 Via sizing

Section 2.2.1.2 described how the vias must be selectively expanded to their correct size. As figure 2.13 shows, the original 0.6 x 0.6 via is scaled to 0.35 x 0.35. Since the new via must have dimensions of 0.4 x 0.4, it is selectively expanded. This is possible since the new design rules require less metal overlap around the vias.



(a) (b)

Figure 2.13: Via Scaling and Expanding. *(a) the original 0.6 micron x 0.6 micron via scaled to (b) 0.35 x 0.35 microns*

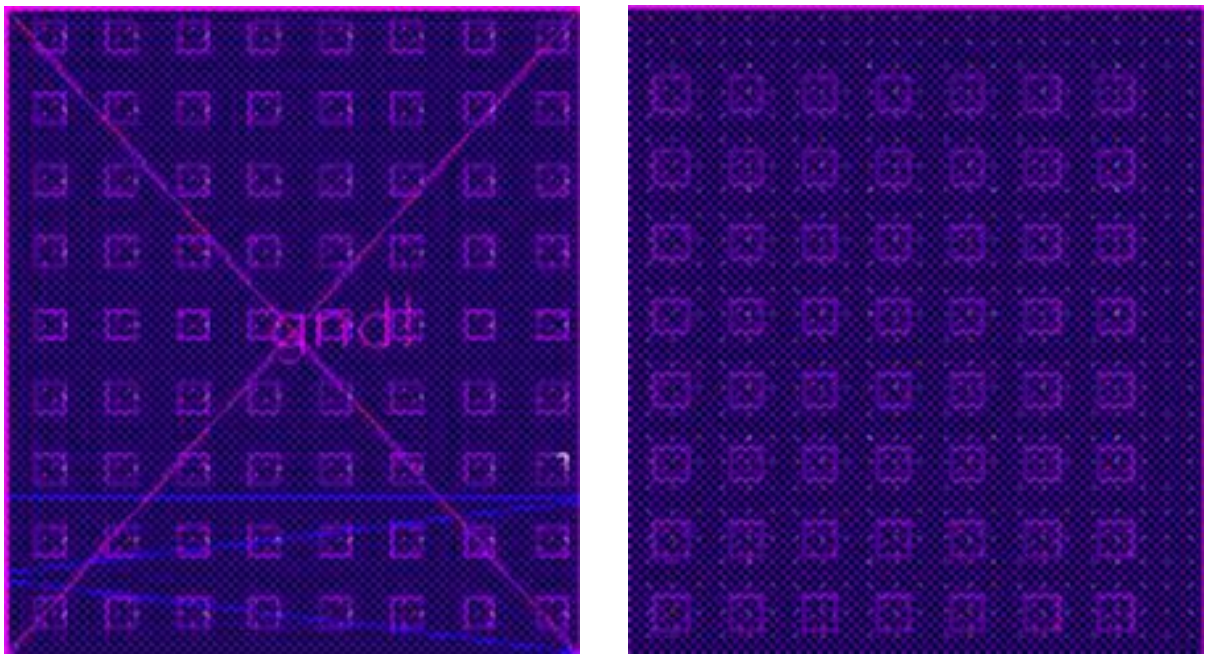


(c)

Figure 2.13 (contd.): Scaling and Expanding the Vias. *(c) After the expansion, the via size is now 0.4 x 0.4 microns*

2.3.3 Via Array Spacing

The increase in the relative size of the vias creates a situation where tight via arrays are spaced too close and need to be modified. The way this is resolved was described in 2.2.1.2 above and figure 2.14 below shows an array of vias before and after the scaling is complete. As part (a) shows the original array consists of 8 columns and 9 rows of vias. Part (b) shows how one row and one column are removed to facilitate for the increased spacing between the vias.

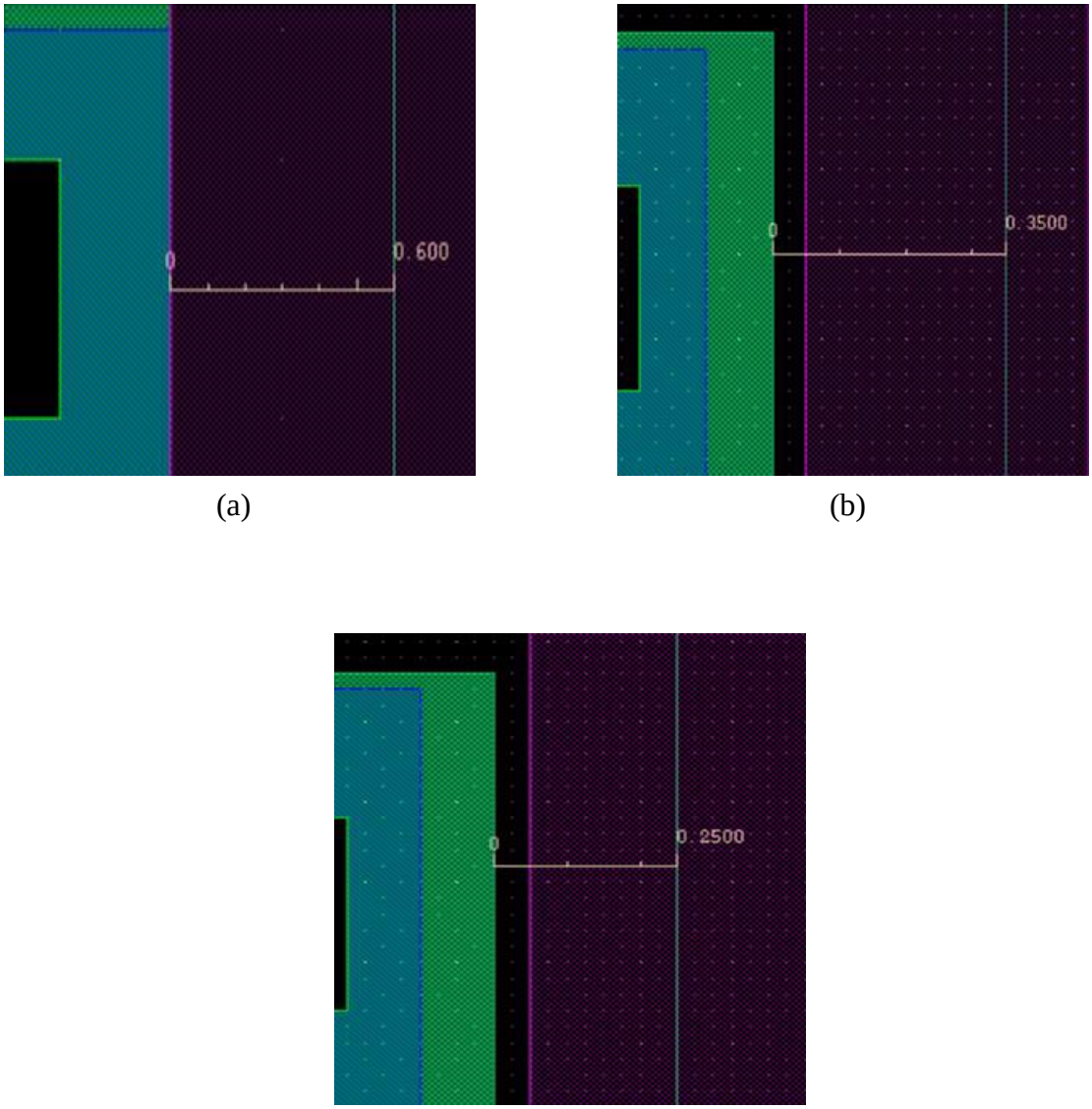


(a) (b)

Figure 2.14: Via Array modification. *(a) the original array with 9 rows and 8 columns (b) the new array with one row and one column removed and the spacing increased.*

2.3.4 Select Layer Adjustment

Figure 2.15 shows how the select layer is brought closer to the active layer to allow for enough spacing between the N and the P selects. Part (a) is from the original design showing the 0.6 micron overlap of the active by the select layer. After the original scaling, the new overlap is reduced to 0.35 microns. This is shown in (b). Since the minimum overlap allowed is 0.25 microns, the select layer is shrunk inwards resulting in the minimum overlap as shown in part (c).



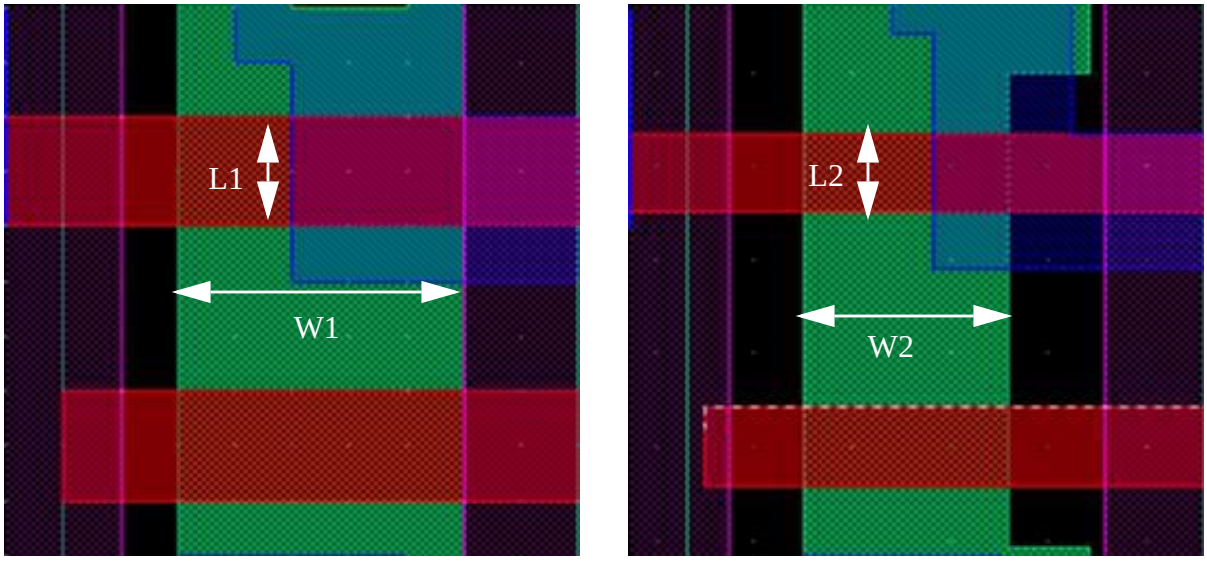
(a) (b)

(c)

Figure 2.15: Reducing the Select Layer Overlap. *(a) the original design with 0.6 microns overlap (b) after the linear scaling the overlap is reduced to 0.35 microns and then (c) further reduced to 0.25 microns.*

2.3.5 Transistor resizing

Figure 2.16 below shows how the transistors are resized to their new width and length. Since the poly was further reduced from 0.35 to 0.25 microns, the active layer is chopped to reduce the width correspondingly.



(a) (b)

Figure 2.16: Transistor Resizing.

2.3.6 Cleanup

The cleanup process involved filling in notches and removing excess “material”. Figure 2.17 shows how the transistor resizing process leaves the active layer somewhat notched and irregular. Figure 2.16 (b) shows the final transistors after the cleanup process.

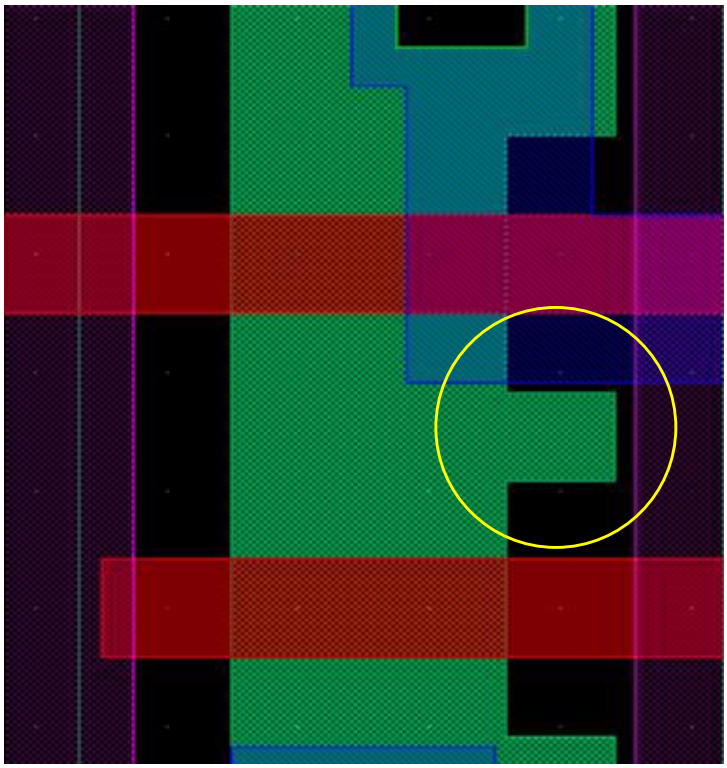


Figure 2.17: Notch Removal

3

Analysis and Results

Aside from functionality and performance, power and area are two of the most important measures of a successful design. Great efforts are made in minimizing these two parameters. The design area has a direct influence on the overall power consumption. Larger area means more transistors and longer distances which translate directly into power. More power, in return, may limit the size of the chip as the consumption and the resulting heat may prove to be intolerable.

This chapter attempts to explore the impact of the scaling on the overall area, power dissipation, and performance. After a brief description of the area savings achieved, section 3.2 will look more closely at the power which can be determined from the circuit capacitance and the supply voltage. We will also look at the performance and how it compares to the original design's

performance. The final sections will discuss and provide some statistics from the verification process.

3.1 Area

One of the biggest advantages brought about by smaller feature sizes is the decrease in the overall area the design occupies. This savings in real estate makes it possible to realize more complex circuits on the same die, a trend that is gaining importance as we are moving towards System On a Chip.

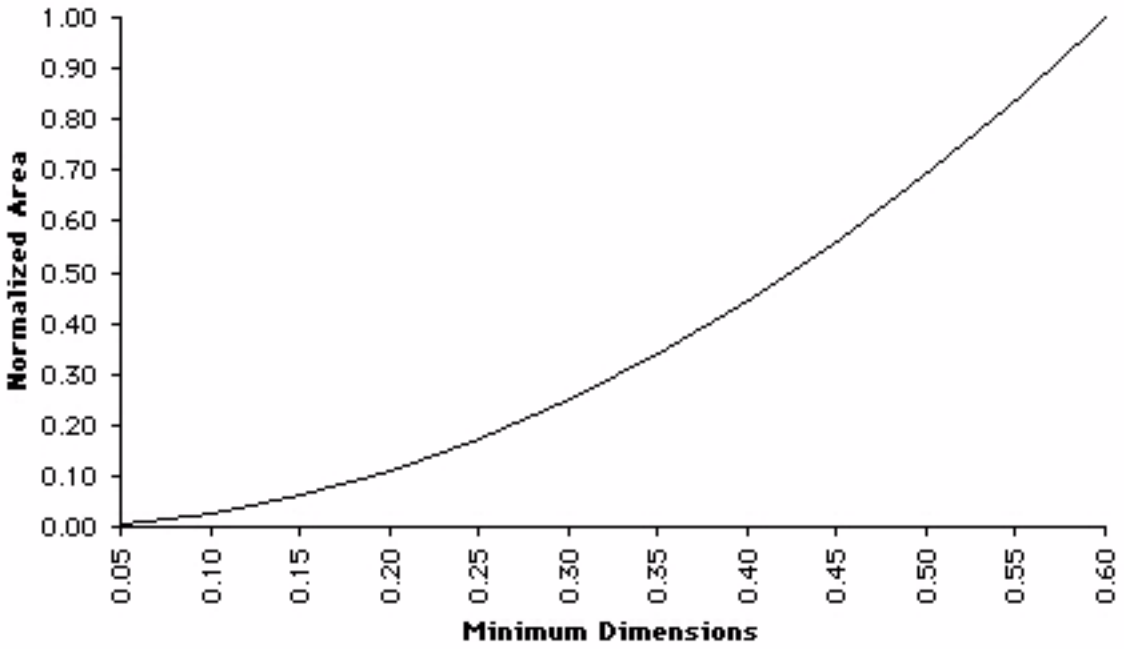


Figure 3.1: Design Area

Figure 3.1 shows how the area scales as the square of the design dimensions. As the figure shows, the ARM scaling to 0.35 microns has produced a three fold decrease in its area. Had

it been possible to scale it all the way down to 0.25 microns we would have realized a five fold area savings.

3.2 Power

As our current ARM design is a scaled down version of the original 0.6 micron one, our ability to introduce power saving measures is practically non-existent. Of course, nothing stops us from making changes if these are deemed necessary, however our approach is that of complete automation with minimum or no need for redesign. One key advantage in this case is that the original ARM was designed with low power operation in mind, employing specially designed low power cell libraries and many power saving techniques [3]. We therefore expect the scaled down version to be equivalently power efficient. A main source of possible difference is the physical characteristics of the new technology. The sections that follow will look at how the capacitance (a main contributor to power consumption) of the devices themselves as well as the scaling of the interconnect between the two technologies.

The main source of power dissipation in digital circuits is dynamic switching power which is given by the formula:

$$P = C_L \times V_{DD}^{\ 2} \times f$$

Where C_L is the load capacitance, V_{DD} the supply voltage, and f the switching frequency. As this equation suggests, a savings in power occurs if any of these terms are reduced. However, fast performance requires a high operating frequency, which although is not the same as the switching frequency it directly affects it. At the circuit level, techniques to reduce glitching, turn off certain

blocks that are not in use, or to eliminate the clock (asynchronous and self-timed) can and are sometimes used in an attempt to reduce the switching frequency and the power consumption due to it. Substantial effort is dedicated to reducing the parasitic capacitance, both device and interconnect, and to operating at ever lower supply voltages.

3.2.1 Capacitance

Circuit capacitance can be broken down into two types, device and interconnect. Small, local circuits, are typically dominated by device capacitance. As the chip size increases the distance a signal travels also increases and interconnect capacitance can begin to dominate. The scaling process had two affects on the circuit capacitance. On the one hand all the devices and the distances were scaled and became smaller and shorter. On the other hand, we are now dealing with a different technology having different physical characteristics. These characteristics cause the device capacitance to scale differently from the interconnect capacitance, making our task of comparing and analyzing the results somewhat more complex.

A careful study of the capacitance scaling is, therefore, necessary if we want to predict the performance and the power dissipation of the new design. We hence compare the increase or decrease in capacitance between the original design and the new one as we perform the scaling process. We look at the two types of capacitance, namely device and interconnect, first separately and then together with the actual data obtained from simulations. We distinguish between the different metal layers when we look at the interconnect capacitance as their physical properties vary.

Figure 3.2 attempts to explain the behavior of the device capacitance (gate capacitance) as the design is scaled down from the original 0.6 microns. The figure includes the effects of both the scaling as well as the new physical properties of the 0.25 micron technology. Assuming a minimum width transistor, the data is derived as follows and is presented in table 3.1 below:

$$C_G = W \times L \times C_{OX} = L \times 1.5L \times \frac{\varepsilon_{OX}}{T_{OX}}$$

$$C_{RATIO} = \frac{C_{Gnew}}{C_{Gold}} = \frac{L_{new}^2 \times T_{OXold}}{L_{old}^2 \times T_{OXnew}} = \frac{L_{new}^2 \times 94}{0.6^2 \times 50}$$

Gate Length [μm]	**C_{RATIO}**
0.25	0.33
0.30	0.47
0.35	0.64
0.40	0.84
0.45	1.06
0.50	1.31
0.55	1.58
0.60	1.88

Table 3.1: Gate Capacitance Ratio for Different Gate Lengths

As the plot shows, we should experience a relative improvement in the transistor's gate capacitance as we move below approximately 0.45 microns. This is seen by a capacitive ratio smaller than one.

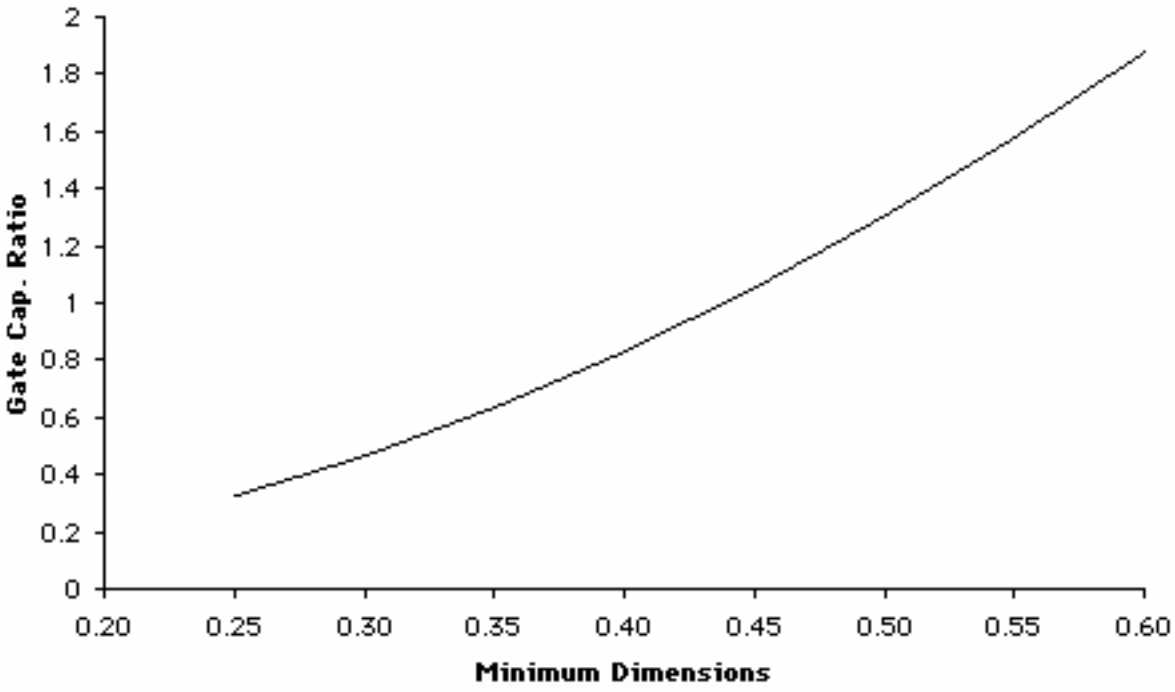


Figure 3.2: Gate Capacitance Ratio (assuming 0.25 micron process technology)

Figure 3.3 tries to show the same information for the case of interconnect dominated capacitance. It is the ratio of the new interconnect capacitance (at different lengths) to the old one at a 0.6 micron length. The figure distinguishes between the different metal layers for reasons we mentioned earlier. Table 3.2 on the next page shows the interconnect capacitance per unit length for the three metal layers in the two technologies. The numbers indicate that the interconnect capacitance has worsened in the new technology. However, as the length of the interconnect decreases so does its capacitance. The curve in figure 3.3 is normalized for the different lenghts.

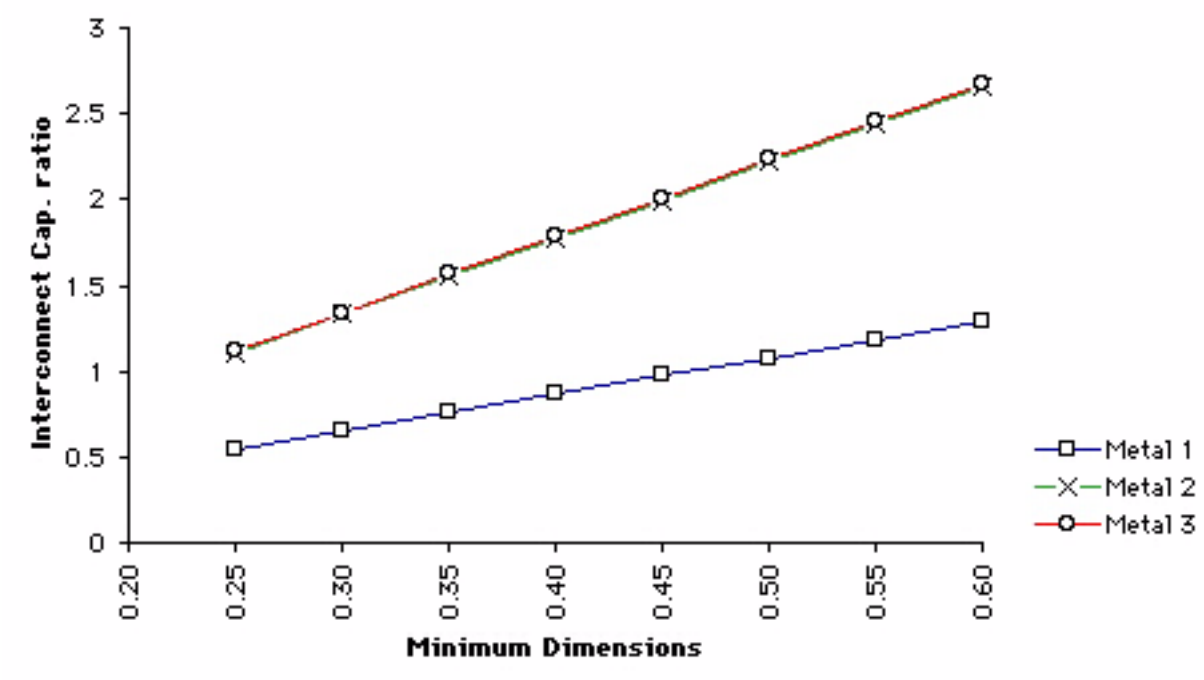


Figure 3.3: Interconnect Capacitance Ratio

We clearly see that whereas a significant improvement is realized in the metal 1 interconnect capacitance when moving to the 0.25 micron technology, signals travelling along metal 2 or metal 3 lines now face a larger capacitive load.

Layer	**Capacitance per Length [fF/mm]**	
	0.6 micron technology	**0.25 micron technology**
Metal 1	72.3	93.7
Metal 2	28.1	74.7
Metal 3	24.6	65.9

Table 3.2: Interconnect Capacitance per Unit Length (area and fringing capacitance for single line)

Figure 3.4 plots the actual capacitive ratio as was measured from our simulations. These results were obtained by performing our scaling process multiple times on only one block, each time to a different dimension. We chose the a8Fwd block and looked at 8 different nodes throughout the design. We did not use the entire ARM design for this study since scaling it multiple times is a non-trivial, time consuming task. This however means that the results thus obtained are not a true representation of the scaling process. In particular, the effect of the long buses, which are major contributors to the interconnect capacitance, is left out. Furthermore, it was difficult to distinguish between device dominated and interconnect dominated nodes within the local block. We, therefore, make no such distinction in our measured results and present only one curve. This curve is the average capacitance as measured over several nodes in the design.

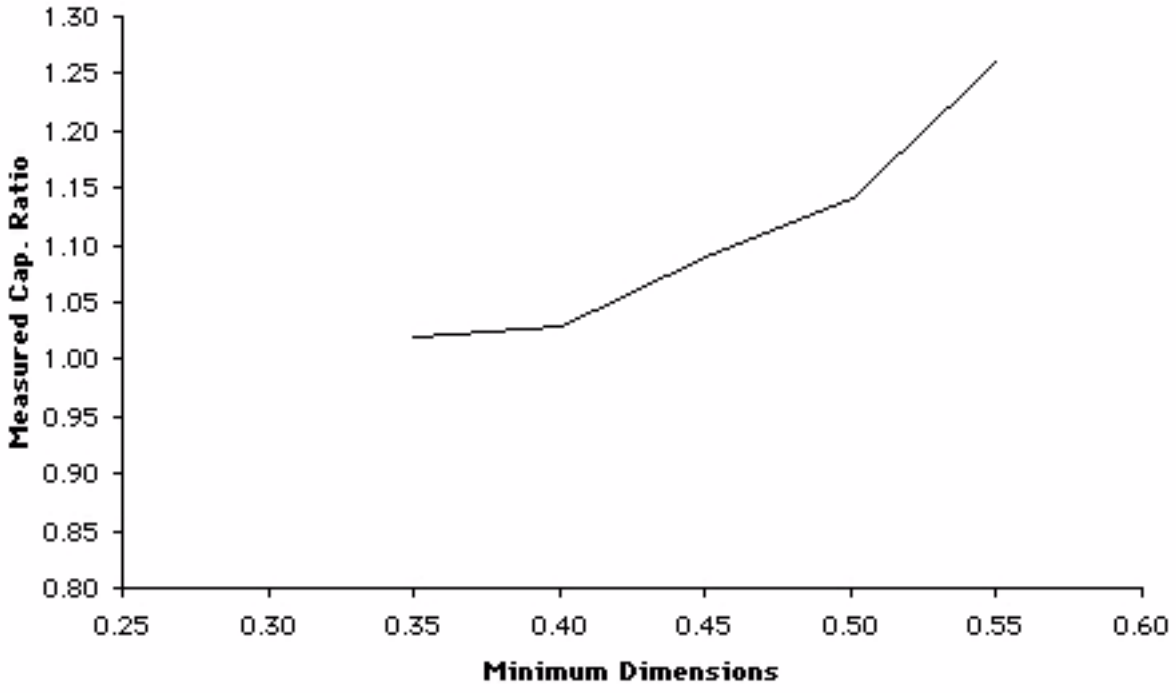


Figure 3.4: Measured Capacitance Ratio

3.3 Delay

To determine the speed at which we can operate the new ARM we study the circuit delay. As was done for the circuit capacitance, we distinguish between two types of delay. One is delay based on device loading only while the other is interconnect dominated delay. As figure 3.5 shows, since device capacitance improves significantly due to scaling and the new technology, transistor delays are expected to reflect that improvement. This is shown by the bottom curve labeled “Gate” in the figure. Similarly, it was noted that interconnect capacitance tends to worsen, particularly for metal2 and metal3 layers. We therefore see that interconnect based delay improves slower than the gate delay as shown by the top curve of figure 3.5 which is the metal3 based interconnect delay.

The calculated data was obtained by comparing the propagation delay of a transistor in each of the two technologies. In one case we used a gate capacitance as the load while in another we used an interconnect capacitance.

$$t_P = \frac{V_{DD} \times L \times T_{OX}}{2 \times W \times \varepsilon_{OX} \times \mu_P} \times \left\langle \frac{1}{2.5 \times \langle V_{DD} - V_{TN} \rangle^2} + \frac{1}{\langle V_{DD} - V_{TP} \rangle^2} \right\rangle \times C_L$$

$$Speedup = \frac{t_{Pold}}{t_{Pnew}} = \frac{V_{DDold}}{V_{DDnew}} \times \frac{T_{OXold}}{T_{OXnew}} \times \frac{\left\langle \frac{1}{2.5 \times \langle V_{DDold} - V_{TNold} \rangle^2} + \frac{1}{\langle V_{DDold} - V_{TPold} \rangle^2} \right\rangle}{\left\langle \frac{1}{2.5 \times \langle V_{DDnew} - V_{TNnew} \rangle^2} + \frac{1}{\langle V_{DDnew} - V_{TPnew} \rangle^2} \right\rangle} \times \frac{C_{Lold}}{C_{Lnew}}$$

$$Speedup = 3.277 \times \frac{C_{Lold}}{C_{Lnew}}$$

To measure the actual delay, we use the same block we scaled to the different dimensions in order to examine the capacitance scaling. We choose a certain path and use Hspice to determine the delay. The results are shown by the curve labeled “Measured” in figure 3.5. As expected, the actual delay lies somewhere between the pure interconnect delay curve and the gate delay curve since the layout consists of both. We, however, expect the actual ARM delay to be somewhat longer (worse performance) than that depicted in the plot. This is due to the fact that the information for this plot was gathered by looking at only one block and not the entire ARM. Therefore, the plot does not take into account all those delays caused by the global interconnect of the ARM.

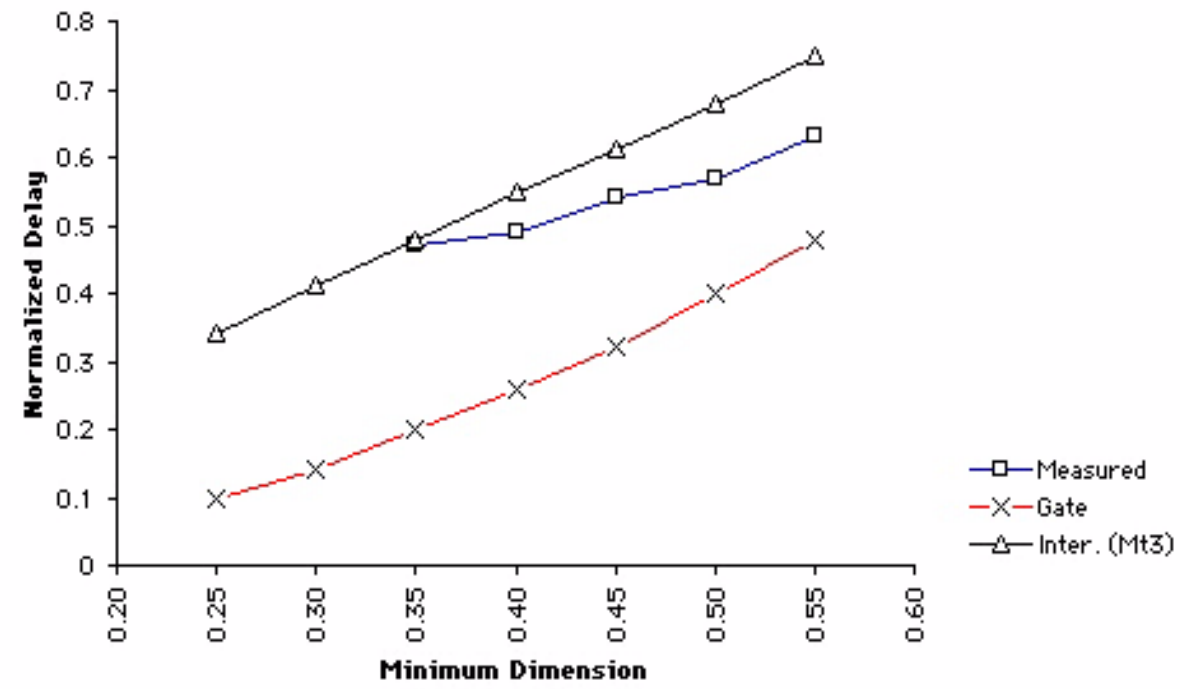


Figure 3.5: Circuit Delay

3.4 Supply Voltage

From the previous analysis and the results obtained we see that the new ARM should operate about twice as fast as the original one. These results assume a 1V target supply voltage. In this section we examine the effect of V_{DD} on performance. Figure 3.6 shows the speed up obtained as we vary the supply voltage from 1V to 2.5V. A significant improvement in performance can be obtained just by increasing V_{DD} slightly above 1V.

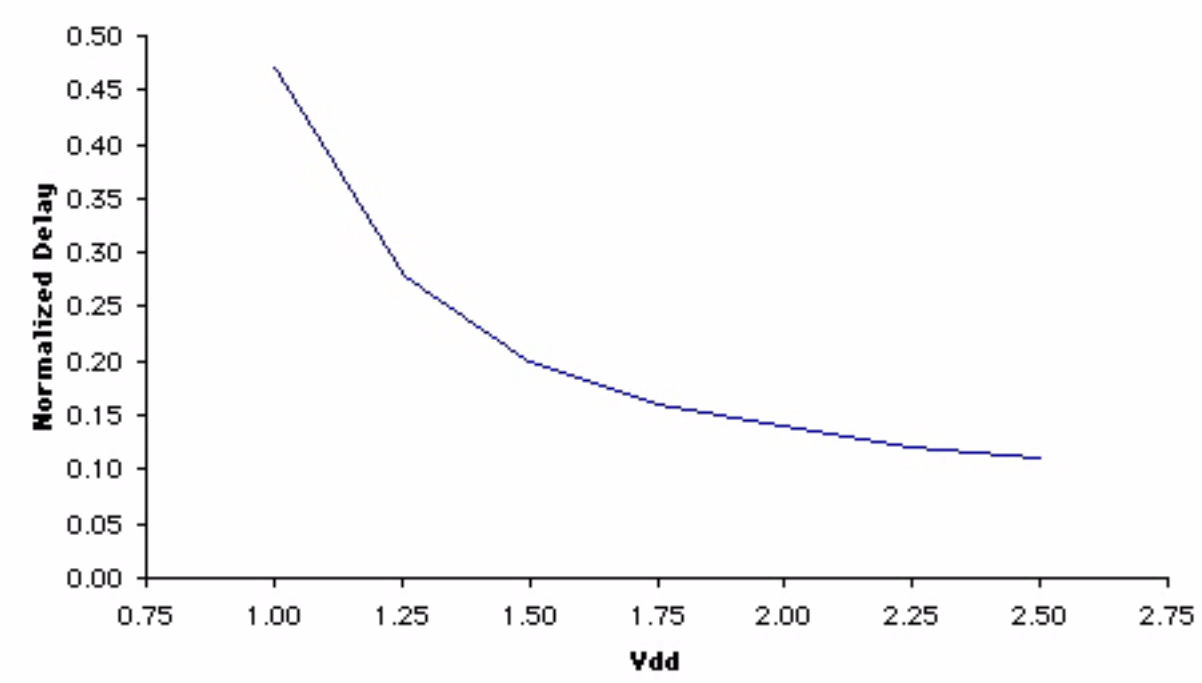


Figure 3.6: Effects of Vdd on Delay

3.5 Verification

The verification process consisted of two main parts. First we had to ensure that the design contained no design rule violations and that it agreed with the new schematic. The second step consisted of a study to ensure that no race conditions existed.

3.5.1 DRC and LVS

To ensure that the final design was correct both physically and logically, we first ran DRC and corrected all the errors manually. Appendix B contains a comprehensive table showing all the different violations reported and the number of fixes needed to eliminate them. Although some blocks contained a large number of violations most of them stemmed from few or one sub-block. Due to the hierarchical nature of the design, a fix in the lower level block which was replicated many times was enough to remove most if not all the violations.

To verify that the design was logically correct we ran an LVS test. The very few cases where the transistor was not scaled correctly we fixed manually. All other connectivity information was not changed during the scaling process.

3.5.2 Clock Skew

Race conditions are a direct outcome of skew. The complex clock network, spreading across the entire chip, is susceptible to skew problems. To ensure that skew was within acceptable limits we followed the testing method performed for the 0.6 micron ARM design. This

process consisted of examining race conditions within local blocks as well as the design as a whole. Table 3.3 shows the entire skew picture.

Vdd	Process	Global Clock RC	Local Clock RC	Clock Driver	Local Enable RC	Total Skew	Max. Allow. Skew
2.5V	**Fast**	14	2.72	65.2	0.11	82.03	290
	Typ			74.7		91.53	381
	Slow			86.0		102.83	477
1V	**Fast**			138.5		155.33	1030
	Typ			197.3		214.13	1621
	Slow			269.1		285.93	2398

Table 3.3: Clock Skew [pS]

To examine the interconnect delay, the length and width of the clock line with all its branches was measured and using a pi 3 model (see figure 3.7) a spice netlist was created. This enabled us to obtain an approximate figure for the clock delay to each block of the ARM design.

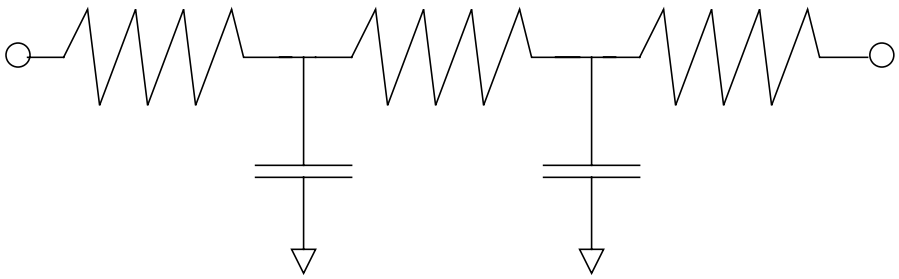


Figure 3.7: A Pi-3 Model

We were also able to use the model to test how the delay changes when we vary different parameters such as line width and metal layer. The conclusion we reached was that the delay can be significantly reduced by using a wider metal 5 global clock line. Figure 3.8 shows the delay for different width of metal 5 clock line. Although the delay will continue to decrease as we widen the clock line beyond 3 microns, we chose that value for the width of the clock line since it safely satisfies all the delay and skew requirements.

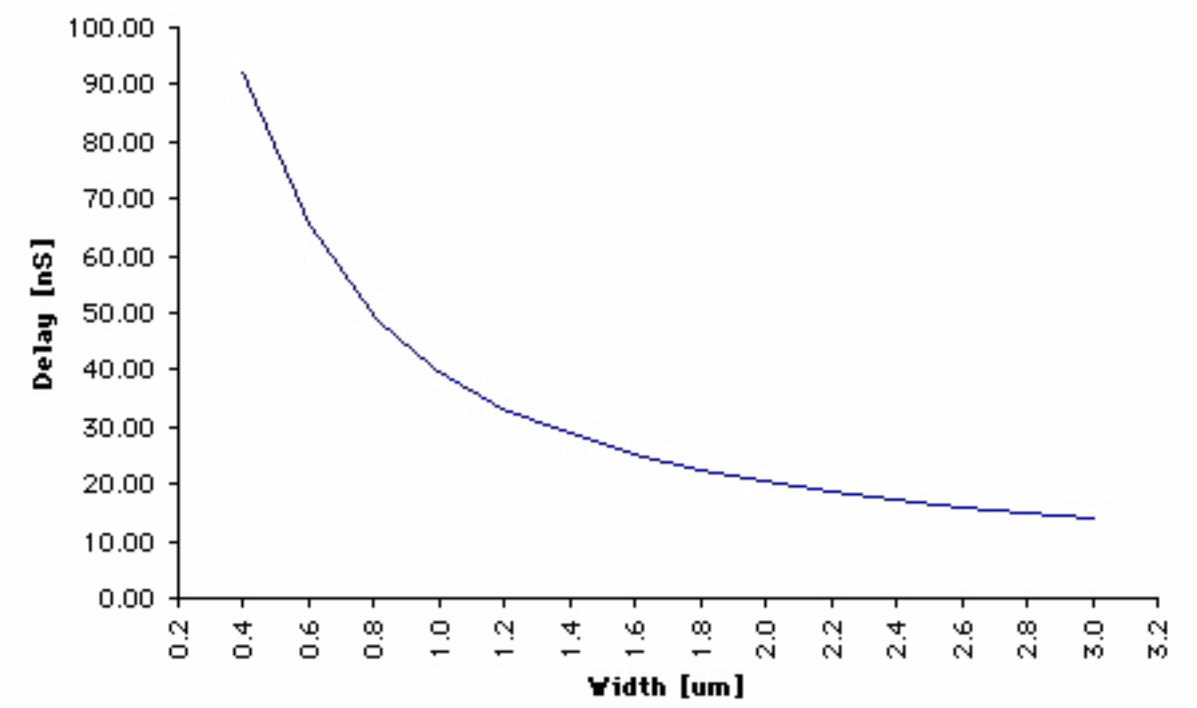


Figure 3.8: Clock Line Delay

Once it was determined that the design is physically as well as logically correct and that no race conditions can occur we generated a netlist and ran simulations to verify the design's

functionality. We used the same test vectors that were used to verify the 0.6 micron ARM while changing them slightly to match our desired speed of operation.

4

Conclusions

This work demonstrated the possibility for automatic methods for design scaling. Such methods can be used to quickly translate designs to new technologies and provide them as basic blocks to the system designer. The short amount of time needed to convert a design using this method and to verify its functionality is one of its greatest advantages. Improving the code efficiency by using better algorithms and ensuring a better compliance with the design rules can further add to its advantage.

This approach however, is not without shortcomings. The disadvantage of scaling to 0.35 microns (in our example) rather than to the absolute 0.25 micron minimums is clear. Not only do we pay a penalty in terms of increased area (only three fold reduction versus five fold for a 0.25 micron design) but, as discussed earlier, this increase brings about a higher power consumption and a decrease in performance. Another apparent disadvantage is the fact that the design is limited to the same number of layers as the original one since the program cannot arbi-

trarily change layers. Newer technologies tend to increase the number of available metal layers which can help decrease the design area and improve performance. This is especially true for those large, interconnect dominated designs but has little or no effect on the lower level blocks and standard library cells. We therefore observe that performance critical, interconnect dominated, designs may not enjoy the same benefits from this method as other designs do. One solution to this problem may be to use this method only as an initial step in the scaling process. Once the scaling is complete the designer can identify the different critical paths and modify those to improve the performance.

A good use for this method can be the scaling of standard cell libraries. Since these cells do not exhibit the same complexity as a large design such as the ARM8, it may be possible to scale them further down beyond 0.3 microns without increasing the program's complexity. Standard cells seldom use more than one layer of metal hence there is no disadvantage in not utilizing the additional metal layers.

Since technologies change quickly, this method of scaling is only as good as the ease with which it can be adapted to newer technologies. Although an attempt was made to keep the program parametrized with respect to the design rules some parts need to be modified to achieve that goal. The objective was to write a program that accepts two sets of design rules and scales the design appropriately.

Based on the two design rules we worked with and the comparisons we made, it is not difficult to come up with a set of "layout rules" which will facilitate the scaling process. Natu-

rally, such rules will call for stricter spacing and sizing to create more flexibility for future automatic scaling. Table 2.1 on page 4 can be the starting point for a modified set of design rules. Table 4.1 below shows the steps that if taken when laying out a design in the old technology can eliminate all violations and allow scaling down to 0.25 microns. If we create a design using these rules and then proceed to scale it to 0.25 microns, all the spacings shown in the table will be exactly minimum size. Although in many instances it is possible to accomodate these "new" rules in a 0.6 micron design without penalty, more often than not the designer will be forced to increase the spacing between the layers and thus the overall layout size. It is impossible to predict how large this increase will be since such a prediction depends on the layout itself and on the relative number of new adjustments. In a worst case scenario where all spacings must be increased, the area will increase by a factor equal to the square of the "new" rule over the old.

Rule Name	Modified 0.6 micron rule
Active Spacing	Minimum Spacing = 1.2 microns
Poly-Active Spacing	Minimum Spacing = 0.48 microns
N+/P+ Implant Spacing	Minimum Spacing = 1.2 microns
Contact Width	Minimum Active Overlap = 0.54 microns
Contact Spacing	Minimum Spacing = 1.2 microns
Contact-Active Spacing	Minimum Spacing = 0.72 microns
Via1 Width	Resolved by selective expand
Via1 Spacing	Minimum Spacing = 1.44 microns
Via2 Width	Resolved by selective expand
Via2 Spacing	Minimum Spacing = 1.44 microns

Table 4.1: New Design Rules. *Adapting these rules when creating a 0.6 micron layout will allow for a scaling to 0.25 microns.*

However, since the relation between any two sets of design rules is hardly the same this approach is not really practical. This would not be the case if all technologies adhered to the lambda design rules. Such a situation will eliminate any complexity associated with scaling as all scaling become a simple linear shrink.

It is somewhat difficult to compare the results we obtained in this work to a possible redesign of the ARM8 at 0.25 microns without actually redesigning it. A new design will clearly have an advantage when it comes to area. It is also likely that a new design will perform faster since all the interconnects are shorter and the parasitics are smaller. The most pronounced difference, however, is design time. Here, the scaled down version has no competition.

A

Program Listing

Since the program's source code is too long to be listed in this report, only the main routine is presented below. The interested reader can access the entire code via the world wide web at the following address: http://infopad.eecs.berkeley.edu/~orowhani/report
The other programs that scale the schematic and extract and save pin information can also be accessed at that location.

A.1 Program Structure

The program has two main modes of operation. These are controlled by a on/off overwrite button on the main menu when the program is called. When the overwrite option is selected, the program will scale each and every cell it encounters regardless of the fact that it may have already been scaled. Deselecting this feature is more efficient and will scale each cell only once. This feature was included to allow repeated scaling of the same design if a change was made. However, as was explained above, for complex designs it will result in slower execution. If

it becomes necessary to repeat the scaling for the same design, deleting the previous run and running the program with this mode not selected is recommended.

The code that shrinks the schematic will only work for mos devices (nmos, pmos, nmos4 and pmos4 transistors) If other devices are used in a schematic the code should be modified accordingly.

A.2 Main Routine

```
defun( shrink ()

; Set default form values
   if( (! boundp( ’libnamedef) ) then
      libnamedef = "" )
   if( (! boundp( ’cellnamedef) ) then
      cellnamedef = "" )
   if( (! boundp( ’destlibnamedef) ) then
      destlibnamedef = "" )
   if( (! boundp( ’descellnamedef) ) then
      descellnamedef = "" )
   if( (! boundp( ’overwritedef) ) then
      overwritedef = "" )

; Define fields for form
   libname = hiCreateStringField(
         ?name ’libnamep
         ?prompt "Source Library Name"
         ?defValue libnamedef )
   cellname = hiCreateStringField(
         ?name ’cellnamep
         ?prompt "Cell Name"
         ?defValue cellnamedef )
   destlibname = hiCreateStringField(
         ?name ’destlibnamep
         ?prompt "Destination Library Name"
```

```
            ?defValue destlibnamedef )
    descellname = hiCreateStringField(
            ?name 'descellnamep
            ?prompt "Destination Cell Name"
            ?defValue descellnamedef )
    overwrite = hiCreateToggleField(
            ?name 'overwritep
            ?prompt "Overwrite Existing Files?"
;           ?defValue list(overwritedef)
            ?choices list( list('ov "")) )

; Create and Display Form
    hiCreateAppForm(
             ?name 'shrinkForm
             ?formTitle "Shrink Layout"
             ?callback "shrinkCode( libname->value destlibname->value cellname->value
                                    descellname->value overwrite->value )"
             ?fields list( libname cellname destlibname descellname overwrite ) )
    status = hiDisplayForm( shrinkForm )

; Set defalut values for next call
    libnamedef = libname->value
    cellnamedef = cellname->value
    destlibnamedef = destlibname->value
    descellnamedef = descellname->value
    overwritedef = overwrite->value

    t
)

;;-------------------------------------------------------------------------

defun( shrinkCode ( libname destlibname cellname descellname overwrite )
  prog( ( libId factor trans act_factor libTable )


        overwrite = car(overwrite)

        if( libname == destlibname && cellname == descellname then
            printf("ERROR: Can't overwrite the original cell.  Please check your input.\n")
        else
            libId = dmOpenLib( destlibname )
            if( !libId then
                libId = dmCreateLib( destlibname "." )
                tcLoadTechFile( libId "/tools/cadence/tech/cmos14tb.tf")
                tcSaveTech( libId )
```

```
        else
           if( !tcGetLayerNum( "poly" libId ) then
              tcLoadTechFile( libId "/tools/cadence/tech/cmos14tb.tf")
              tcSaveTech( libId )
           )
        )

        SCALE_LOG = outfile( "./SCALE_LOG.TXT" )
        PIN_ERROR = outfile( "./PIN_ERROR.TXT" )
        SCALE_ERROR = outfile( "./SCALE_ERROR.TXT" )
        libTable = makeTable("libraryTable" nil)

        ; First, set the transformation list
        factor = 0.35/0.6
        trans=list( 0:0 "R0" factor)
        ;; Active shrink factor (for resizing transistor widths)
        act_factor = (old_poly_width*factor)/poly_width

        shrinkCodeRecurs( libname destlibname cellname descellname overwrite nil nil nil )
        dmCloseLib( libId )
        close( PIN_ERROR )
        close( SCALE_LOG )
        close( SCALE_ERROR )
        printf("Program completed, please check:\n
           SCALE_ERROR.TXT for any error messages.\n
           SCALE_LOG.TXT   for output log.\n
           PIN_ERROR.TXT   for pin errors.\n" )
     )
  )
)

;;-------------------------------------------------------------------------
;;
;; shrinkCode:  This is the main routine called when the menu is filled in
;;              and executed.
;;
;;-------------------------------------------------------------------------

defun( shrinkCodeRecurs ( libname2 destlibname2 cellname2 descellname2
                          overwrite2  parentDb parentXY parentOrient )
   prog( ( instanceList retval origDb destDb numfound done delta
           newXY newbBox object windowID libID libName filename closeWin )

  printf("Shrinking cell from %2.2f to %2.2f\n" 0.6 0.35)
  fprintf( SCALE_LOG "Shrinking cell from %2.2f to %2.2f\n" 0.6 0.35)
  printf("Cellname:  %s Source library:  %s  To:\n" cellname2 libname2)
```

```
fprintf( SCALE_LOG "Cellname:  %s Source library:  %s  To:\n" cellname2 libname2)
printf("Cellname:  %s Destination library:  %s\n" descellname2 destlibname2 )
fprintf( SCALE_LOG "Cellname:  %s Destination library:  %s\n" descellname2 destlibname2 )

    origDb = dbOpenCellView( libname2 cellname2 "layout")
    if( (!origDb) then
       printf("ERROR: Could not open cell '%s' in library '%s'!\n"
                        cellname2 libname2 )
       fprintf( SCALE_ERROR "ERROR: Could not open cell '%s' in library '%s'!\n"
                         cellname2 libname2 )
       fprintf( SCALE_LOG "ERROR: Could not open cell '%s' in library '%s'!\n"
                         cellname2 libname2 )
       ;return(nil)
       )

    retval = dbCopyCellView( origDb destlibname2 descellname2 "layout" "" nil t)
    if( (!retval) then
       printf("ERROR: Could not copy cell '%s' to library '%s'!\n"
                        cellname2 destlibname2)
       fprintf( SCALE_ERROR "ERROR: Could not open cell '%s' in library '%s'!\n"
                         cellname2 destlibname2)
       fprintf( SCALE_LOG "ERROR: Could not open cell '%s' in library '%s'!\n"
                         cellname2 destlibname2)
       ;return(nil)
       )

    destDb = dbOpenCellView( destlibname2 descellname2 "layout" nil "a")
    if( (!destDb) then
       printf("ERROR: Could not open cell '%s' in library '%s'!\n"
                        descellname2 destlibname2 )
       fprintf( SCALE_ERROR "ERROR: Could not open cell '%s' in library '%s'!\n"
                        descellname2 destlibname2 )
       fprintf( SCALE_LOG "ERROR: Could not open cell '%s' in library '%s'!\n"
                        descellname2 destlibname2 )
       ;return(nil)
       )

    closeWin = nil
    if( !libTable[destlibname2] then
       ;; Open a window for the new layout then iconify it so it is
       ;; out of view until the shrinking is done, then close it.
       windowID = hiCreateWindow('default "graphics" "layout")
       geOpen( ?window windowID ?lib destlibname2 ?cell descellname2
          ?view "layout" ?version "0.0" ?mode "a")
       hiIconifyWindow( windowID )
       libTable[destlibname2] = t
```

```
    closeWin = t
)

;; flatten all mosaics and instances
flattenAll( destDb )

; Merge all similar layers into a single polygon
mergeShapes( destDb list( metal3 metal2 metal1 poly active  nwell "pwell" pselect ) t )

;; convert all paths to polygons
convertPathToPoly( destDb )

;; Shrink all rectangles
shrinkShapes( destDb "rect" trans )

;; shrink all polygons
shrinkShapes( destDb "polygon" trans )

;; shrink all pin labels
shrinkPinNames( destDb )

;; shrink all labels
numfound = shrinkLabels( destDb factor trans )
printf("%d labels shrunk\n" numfound)
fprintf( SCALE_LOG "%d labels shrunk\n" numfound)

;; find any other shapes
findOtherShapes( destDb )

; Expand/Shrink all contacts and vias to correct size
expandVia( destDb list( contact "cp" via1 via2 ) )

;; merge and split all Vias that are vertically too close to each other
printf("Merging and creating properly spaced vias...\n")
fprintf( SCALE_LOG "Merging and creating properly spaced vias...\n")
mergeViaVert( destDb )
splitShape( destDb "vertical" )
;; merge all Vias that are Horizontally too close to each other
mergeViaHoriz( destDb )
splitShape( destDb "horizontal" )

;; Shrinking select layers
printf("Shrinking and adjusting select layers...\n")
fprintf( SCALE_LOG "Shrinking and adjusting select layers...\n")
shrinkSelect( destDb "nselect" "pselect" 0.1 )
shrinkSelect( destDb "pselect" "nselect" 0.1 )
```

```
;; Shrinking to minimum size
shrinkToMin( destDb list( poly metal1 metal2 metal3 ) )

;; Repairing the contact/via enclosure
repairContacts( destDb )

; Merge all similar layers into a single polygon
mergeShapes( destDb list( metal3 metal2 metal1 poly active ) t )

;; Resizing transistor widths
resizeXtr( destDb )

;; shrink area around contacts and vias
printf("Minimizing contact/via area...\n")
fprintf( SCALE_LOG "Minimizing contact/via area...\n")
numfound = 0
numfound = shrinkContactArea( destDb "cp" poly numfound )
numfound = shrinkContactArea( destDb contact active numfound )
numfound = shrinkContactArea( destDb contact metal1 numfound )
numfound = shrinkContactArea( destDb "cp" metal1 numfound )
printf("%d contact/via area minimized\n" numfound )
fprintf( SCALE_LOG "%d contact/via area minimized\n" numfound )


;; Fixing min size vilations after trimming around contacts
expandToMin( destDb list( poly metal1 metal2 metal3 ) )

; Merge all similar layers into a single polygon
mergeShapes( destDb list( metal3 metal2 metal1 poly pselect ) t )

printf("Repairing notched figures...\n")
fprintf( SCALE_LOG "Repairing notched figures...\n")
delta = contact_width + 2*active_overlap_contact - min_design_grid
activeFix( destDb active delta )

notchFix( destDb active active_space )
notchFix( destDb poly poly_width )
notchFix( destDb metal3 metal3_width )

printf("Adjusting and trimming pin layers...\n")
fprintf( SCALE_LOG "Adjusting and trimming pin layers...\n")
adjustPin( destDb metal1 metal1_width/2 old_metal1_width )
adjustPin( destDb metal2 metal2_width/2 old_metal2_width )
adjustPin( destDb metal3 metal3_width/2 old_metal3_width )
adjustPin( destDb poly poly_width/2 old_poly_width )
```

```
adjustPolyEdge( destDb )

removeSelectOverlaps( destDb )

;; Get a list of all the instances in this level
instanceList = dbProduceOverlapInst(destDb destDb->bBox)
foreach( obj instanceList
  ;; move each instance to new location
  newXY = car(obj->xy)*factor:cadr(obj->xy)*factor
  obj->xy = alignToGrid( newXY )
  ;; resize instance's bBox
  newbBox = list( caar(obj->bBox)*factor:cadar(obj->bBox)*factor
                  caadr(obj->bBox)*factor:cadadr(obj->bBox)*factor )
  obj->bBox = alignToGridBBox( newbBox )
  ;; shrink each instance recursively
  ;; Each cell is shrunk into "libname_SGS2"
  libName = strcat( obj->libName "_SGS2" )
  libId = dmOpenLib( libName )
  if( !libId then
      ;; If the library does not exist, then create it and load the
      ;; technology file
      libId = dmCreateLib( libName "." )
      tcLoadTechFile( libId "/tools/cadence/tech/cmos14tb.tf")
      tcSaveTech( libId )
  else
      ;; If the library exist, check that the technology file was
      ;; already loaded.  If not, load it.
      if( !tcGetLayerNum( "poly" libId ) then
         tcLoadTechFile( libId "/tools/cadence/tech/cmos14tb.tf")
         tcSaveTech( libId )
      )
  )
  if( !dmFindCellView( dmFindLib( libName ) obj->cellName "layout" ) ||
                                        overwrite2 then
     printf("\nDescending into %s\n" obj->cellName)
     fprintf( SCALE_LOG "\nDescending into %s\n" obj->cellName)
     temp = shrinkCodeRecurs( obj->libName libName obj->cellName obj->cellName
                              overwrite destDb obj->xy obj->orient)
     printf("\nReturned from  %s\n" obj->cellName )
     fprintf( SCALE_LOG "\nReturned from  %s\n" obj->cellName )
  else
     printf("Using %s from %s library.\n" obj->cellName libName )
     fprintf( SCALE_LOG "Using %s from %s library.\n" obj->cellName libName )
     temp = dbOpenCellView( dmFindLib( libName ) obj->cellName "layout" )
     copyFigToParent( temp destDb obj->xy obj->orient pselect "y0" )
```

```
            copyFigToParent( temp destDb obj->xy obj->orient metal1 "y1" )
            copyFigToParent( temp destDb obj->xy obj->orient metal2 "y2" )
            copyFigToParent( temp destDb obj->xy obj->orient metal3 "y3" )
            copyFigToParent( temp destDb obj->xy obj->orient poly "y4" )
            copyFigToParent( temp destDb obj->xy obj->orient active "y5" )
          )
          ;; make instance point to new master
          obj->master = temp
          dbClose( temp )
          dmCloseLib( libId )
        ); foreach

        adjustLayer( destDb pselect select_space/2 "y0" )
        adjustLayer( destDb metal1 metal1_width/2 "y1" )
        adjustLayer( destDb metal2 metal2_width/2 "y2" )
        adjustLayer( destDb metal3 metal3_width/2 "y3" )
        adjustLayer( destDb poly poly_width/2 "y4" )
        adjustLayer( destDb active active_space/2 "y5" )

        copyFigToParent( destDb parentDb parentXY parentOrient pselect "y0" )
        copyFigToParent( destDb parentDb parentXY parentOrient metal1 "y1" )
        copyFigToParent( destDb parentDb parentXY parentOrient metal2 "y2" )
        copyFigToParent( destDb parentDb parentXY parentOrient metal3 "y3" )
        copyFigToParent( destDb parentDb parentXY parentOrient poly "y4" )
        copyFigToParent( destDb parentDb parentXY parentOrient active "y5" )


        ;; Save pin information to a text file in the current directory.
        filename = strcat("./" destlibname2 "_" descellname2 ".txt")
        printOutPins( destDb filename )
        deleteAllPins( destDb )
        ;; save and close
        dbSave(destDb)
        dbClose(destDb)
        if( closeWin then hiCloseWindow( windowID ) )
        return( destDb )
        )
)
```

A.3 Function Description

Name: Shrink

Description: This is the top most function. It displays a menu for the user to enter the required information and then passes those to the main program.

Name: shrinkCode
Description: This function receives the source and destination cell and library names. It also gets the ‘overwrite’ request from the user. It initializes some log files and the scaling factor, opens the top most cell then calls the rest of the program which is executed recursively.

Name: shrinkCodeRecurs
Description: This is the actual main routine that is responsible for the shrinking of each cell. This function calls all the other functions that perform the scaling / expanding and other operations on all the different layers.

Name: flattenAll
Description: Flattens all mosaics. (commented out within this function is a section that can flatten the entire design. If the comments are removed the resulting scaled down layout will no longer be hierarchical)

Name: mergeShapes
Description: Merges all shapes of the same layer into one polygon.

Name: convertPathToPoly
Description: Converts all figures of type path to polygon for all layers.

Name: shrinkShapes
Description: Shrinks shapes of type rectangle and polygons by the scaling factor.

Name: shrinkPinNames
Description: Shrinks all pin names by the scaling factor.

Name: shrinkLabels
Description: Shrinks all labels by the scaling factor.

Name: findOtherShapes
Description: Reports if any other shapes of type line, ellipse, arc, donut, or dot exist in the layout.

Name: expandVia
Description: Expands all vias (via1 and via2) as well as contacts to their correct dimensions.

Name: mergeViaVert
Description: Merges all via arrays that are lined up in a vertical line into one long rectangle.

Name: splitShape
Description: Splits the long via rectangle into individual, properly sized and spaced, vias.

Name: mergeViaHoriz

Description: Similar to mergeViaVert. It merges all the horizontal vias in a via array.

Name: shrinkSelect
Description: Shifts the select layer closer to the active to allow for enough spacing between the select layers.

Name: shrinkToMin
Description: Shrinks all poly and metal lines to their respective minimum widths. Only those traces that were minimum sized in the original design are transformed.

Name: repairContacts
Description: Repairs the overlap area around contacts and vias by adding the necessary layers. For active contacts a rectangle of active and metal1 is added. For poly contact, metal1 and poly rectangles are added and so on.

Name: resizeXtr
Description: Resizes the transistors to their correct W/L ratio by chopping off a section of the active under the gate.

Name: shrinkContactArea
Description: Removes the excess overlap from around contacts and vias.

Name: expandToMin
Description: Expands those traces that were made smaller than minimum size due to some of the previous operations such as the shrinkContactArea function.

Name: activeFix
Description: Removes all notches from the active layer that are the result of the transistor resizing.

Name: notchFix
Description: Fixes all notches on the given layers.

Name: adjustPin
Description: Trims the pin layers so a drawing layer always exists under it.

Name: adjustPolyEdge
Description: Adjusts the poly endcaps and makes them shorter to avoid spacing violations with an adjacent active layer.

Name: removeSelectOverlaps
Description: Removes any overlaps created around donuts or when one of two abutting select layers is minimum size and therefore gets expanded resulting in an area of overlap.

Name: copyFigToParent

Description: Copies all shapes of a given layer to the parent cell. The reason for this was described earlier and is to resolve spacing and notches that are caused by the hierarchical nature of the design.

Name: adjustLayer
Description: Creates the necessary “pockets” to remove any spacing and sizing violations over two levels of the hierarchy.

Name: printOutPins
Description: Prints out the pin information for each cell to a text file.

B

DRC Results

The following table gives a comprehensive listing of all the violations and the required number of fixes in the DRC process.

Block	Cell Name	Total Errors	No. of Fixes	M2 Spacing	M2 Notch	M2 Width	Pplus spacing	Pplus Width	Pplus Notch	M1 Spacing	M1 Notch	M1 Width	Via1 Spacing	Poly Width	Poly Dist to active	Other	Pin W/O Draw
a8AbortPipe	a8AbortPipe	7	4				2					2	3				
a8AoutMux	a8AoutMux	1	1								1						
	la8aoutmux_con	7	4			2	1	1					3				
	la8aoutmux_dp	128	10											128			
a8ByteRot	a8ByteRot	9	1								1						8
	la8byterot_con	0	0														
	la8byterot_dp	0	0														
a8ConsGen	a8ConsGen	31	0														31
	la8consgen_con	9	2											6	3		
	la8consgen_dp	3	2		3												
a8DataPipe	a8DataPipe	1	1								1						
	la8datapipe_con	7	3						1				6				
	la8datapipe_dp	0	0														
a8DinMux	a8DinMux	0	0														
	la8dinmux_con	5	2										1		4		
	la8dinmux_dp	0	0														
a8ExGen	a8ExGen	0	0														

Block	Cell Name	Total Errors	No. of Fixes	M2 Spacing	M2 Notch	M2 Width	Pplus spacing	Pplus Width	Pplus Notch	M1 Spacing	M1 Notch	M1 Width	Via1 Spacing	Poly Width	Poly Dist to active	Other	Pin W/O Draw
a8Fwd	a8Fwd	4	3				2					2					
a8IMux	a8IMux	6	1									2					4
	la8imux_dp	0	0														
a8MAS	a8MAS	0	0														
a8MAadder	a8MAadder	17	0														17
	la8maadder_con	2	2						2								
	la8maadder_dp	4	1												4		
a8MainDec	8MainDec	8*	2				1			3		4					
a8Mem	a8Mem	3	2					1								2	
a8Mul	a8Mul	0	0														
	a8MulCtrl	6	5					2	4								
	a8Muldata	56	14		4		3						49				
	a8MulIER	0	0														
	a8MulRow	1	1					1									
	a8MulRow4	0	0														
	a8mAccSel	2	1					1	1								
	a8mBoothSel	0	0														

Block	Cell Name	Total Errors	No. of Fixes	M2 Spacing	M2 Notch	M2 Width	Pplus spacing	Pplus Width	Pplus Notch	M1 Spacing	M1 Notch	M1 Width	Via1 Spacing	Poly Width	Poly Dist to active	Other	Pin W/O Draw
a8Mul (contd.)	a8mBottomDecoder	3	3				3										
	a8mBottomLatch	0	0														
	a8mEarlyTerm	6	6													6	
	a8mTopMux	0	0														
	a8mWriteResult	260	9						2			102			156		
	a8mulAccHighInit	0	0														
	a8mulAccLowInit	0	0														
a8NextInst	a8NextInst	3	3				1	2									
a8PCin	a8PCin	72	29		58												14
	la8pcin_con	0	0														
	la8pcin_dp	0	0														
a8PSR	a8PSR	5	4					1	4								
a8PuAdd	a8PuAdd	23	23		23												
	la8puadd_30	47	30			15		32									
a8PuBus	a8PuBus	2	2				2										
a8PuCC	a8PuCC	2	1									2					
a8PuIin	a8PuIin	25	6			2											23
	la8puiin_con	0	0														

Block	Cell Name	Total Errors	No. of Fixes	M2 Spacing	M2 Notch	M2 Width	Pplus spacing	Pplus Width	Pplus Notch	M1 Spacing	M1 Notch	M1 Width	Via1 Spacing	Poly Width	Poly Dist to active	Other	Pin W/O Draw
a8PuIin	la8puiin_dp	3	2									2					1
a8PuPSM	a8PuPSM	4	3						2						2		
a8PuPcInc	a8PuPcInc	18	2			2					1						15
	la8pupcinc_con	4	4					2	2								
	la8pupcinc_dp	0	0														
a8PuRAM	a8PuRAM	2	2		1												1
	la8puram_con	3	3														3
	la8puram_dp	1047	3	31	8										1008		
a8PuRAMctl	a8PuRAMctl	19	6				3	9	1								6
a8ReadReg	a8ReadReg	84	8		1		1			3	1	78					
a8RegBank	a8RegBank	0	0														
	la8regbank_con	2	2						1								1
	la8regbank_dp	215	10		32							64					119
a8SALU	a8SALU	42	1		1												41
	la8salu_con	4	4				2	1	1								
	la8salu_dp	73	4		72			1									
a8Write	8Write	2	2				2										

Block	Cell Name	Total Errors	No. of Fixes	M2 Spacing	M2 Notch	M2 Width	Pplus spacing	Pplus Width	Pplus Notch	M1 Spacing	M1 Notch	M1 Width	Via1 Spacing	Poly Width	Poly Dist to active	Other	Pin W/O Draw
Arm8	Arm8_hier	0	0														
	a8Core	42	42		1												41
	pu	187	157		60	1											126

References

[1] *Plieades, Ultra Low Power reconfigurable Computing.* http://infopad.EECS.Berkeley.EDU/research/reconfigurable/

[2] Advanced RISC Machines, *ARM 8 Data Sheet*. July 1996.

[3] Tom Burd, *Low-Power CMOS Library Design Methodology.* Master's Thesis, University of california, Berkeley.

[4] Cadence Design Systems, *SKILL Language Reference*.

[5] Cadence Design Systems, *Design Framework II SKILL Functions Reference*.

[6] J. M. Rabaey, *Digital Integrated Circuits, A Design Perspective*. Prentice-Hall, Upper Saddle River, NJ, 1996.