Strategies For Microarchitecture and Logic Optimization

by

Nels Vander Zanden
Daniel Gajski

Information and Computer Science Department
University of California, Irvine
Irvine, CA. 92717

TR No. 88-04

January 29, 1988

Abstract: This report describes different strategies for area, power, and time optimization for designs on microarchitectural and logic levels as implemented in MILO system.

TABLE OF CONTENTS

1. Optimization Strategies

Strategies apply intelligence to the optimization process, deciding which techniques to use and when. A strategy is a plan that selects different techniques to be applied depending upon the current state of optimization. One can view the state of optimization through a graph, such as that in Figure 1a. On the x axis are paths that have been given time constraints. The y axis shows time. A horizontal line drawn through the y-axis represents the "critical" point. The distance a path is from this line is its slack (difference between actual and required times). In constraint driven optimization we would ideally like to produce the circuit represented in Figure 1b. In this figure critical paths A and B have had their delays brought down to the critical point (possibly at the expense of area) and non-critical paths C, D, and E have been raised to the critical point by trading off time for area and power. In this manner, timing constraints are met by reducing critical paths and area/power constraints are met by manipulating non-critical paths.

Time represents a local constraint while both area and power are global constraints. Time constraints may be entered for each path while those for area and power are entered for the entire design. Thus timing optimizations make local improvements, only to selected paths. Area and power optimizations improve the constraint for the entire design, having a global effect. Generally

techniques that decrease area also decrease power and vise versa.

In the next two sections we examine strategies for time and area optimization. Strategies are required to select: 1) what subsection of the design to optimize, 2) what order to use the various optimization techniques, and 3) what rule in a rule set to apply.

1.1. Optimization Strategies for Time

When time is the most important constraint to solve, all critical paths should be made non-critical -- even if it means exceeding the area constraint. One plan to do this is shown in Figure 2. A timing analyzer determines which paths in the design are critical. The optimizer can then proceed to optimize on a path by path basis. For example, one could choose the worst critical path -- the one furthest from its constraints -- then select the next worst path, etc. Alternatively, the optimizer can operate on multiple paths at the same time. This can be done by finding a cover for all critical paths -- that is, finding a set of intervals that cover all of the critical paths. This idea is illustrated in Figure 3. It shows one possible covering for four critical paths.

Once a path is chosen, the optimizer must decide where to optimize the critical path. One can apply techniques over the entire path or choose a subsection of the critical path to optimize.

1.1.1. Strategies for Interval Selection

Optimizing along the entire critical path is the simplest strategy. It can also be time consuming as rules/algorithms can be applied to any components along the path. If the critical path is long, many different rules must be examined to determine which to apply.

More complex strategies examine the topology of the critical path and divide it into intervals. They look for sections of the critical path that offer better optimization opportunities than others. For example, interval boundaries can be drawn where one or more critical paths join or leave the path being optimized as shown in Figure 4(a). In this manner, optimizations in different intervals have effects on different critical paths. Thus the optimizer has greater control as it can select various goals, then choose the interval that can best meet them.

Four strategies for selecting an interval are shown in Figure 4. The first strategy chooses the interval containing the most critical paths. By optimizing along this interval, the most critical paths are reduced with a single transformation. This is especially important when trading off area for time improvements. The increase in area will usually be less from a single transformation than from multiple ones.

The second strategy, shown in Figure 4c, selects the interval closest to the external inputs. This strategy hopes to reduce the most paths overall (both critical and non-critical). Generally improvements made closer to an input result in reductions for more paths than those made on points closer to external outputs. Reductions in the delay of non-critical paths moves them even further below the "critical" point shown in Figure 1. This allows more time-for-area tradeoffs along these paths, producing a design with less area.

The third strategy, depicted in Figure 4d, selects the interval with the most components. Such an interval will most likely contain more possible transformations as well as a larger variety of possible transformations. Thus it may be possible to use more techniques that either increase area only slightly for time improvements or result in major time improvements.

The final strategy, displayed in Figure 4e, chooses the interval having the longest delay. Since this interval represents the bottleneck of the critical path, it is an ideal spot to use techniques that can result in significant improvements. Such intervals may have been poorly designed and can be greatly improved. Alternatively, a well-designed interval may be able to be speeded up through logic duplication techniques.

The four strategies can be used in some combination. For example if strategy 1 selects two intervals (both containing the same number of critical

paths), strategy 2 can be used to break the deadlock.

Another method for interval selection is shown in Figure 5. With this method, interval boundaries are set only when the path being optimized is split or reconverges. Figure 5 shows two paths from *I0* to *Fout*. Both of them fail to meet the timing constraint of 12ns. Intervals *a* and *d* are the best intervals for optimization. Any improvements along these subsections improve both *I0->Fout* paths. However, only a time improvement of 1 can be made on interval *b* without also improving interval *c*. For example, even if *b* is reduced to delay 2 the worst case delay from *I0* to *Fout* has been improved by only 1 unit. Interval *c* is now the bottleneck. Hence a reduction in time between interval *a* and *d* may require optimization on two paths.

1.1.2. Applying Techniques to Intervals

The optimization strategy must also decide how to apply its techniques to intervals. This strategy may choose to operate in three different fashions. The first would be to apply a technique to an interval, then select another interval and apply the same technique. After all intervals have been examined, a new technique can be selected. In this manner, a technique that trades off only a small amount of area for time can be used throughout the critical path before a technique with a greater area cost is applied.

An alternative method is to select an interval, then apply all techniques to it before choosing a new interval. This method takes full advantage of those intervals that provide "better" optimization opportunities. For example, consider a single application of a large area-for-time tradeoff technique along an interval containing many critical paths. It may result in less of an area increase than a number of small area-for-time tradeoffs along other intervals.

The final method uses a combination of the the first two approaches. A number of technique subsets can be created. With this arrangement, an interval can be chosen and a subset of techniques applied. After all techniques in the subset are exhausted, a new interval can be chosen. Finally, after all intervals are examined, a new subset can be selected.

1.1.3. Rule Selection

For rule-based techniques a strategy must be selected to choose a single rule from the set of rules that can be applied along an interval. While lookahead is quite useful, it is also extremely time consuming. The use of effective rule-selection strategies can greatly reduce the need for lookahead. Four such strategies are to select the rule: 1) with the most complex components, 2) with the most components, 3) that replaces the pattern having longest delay, or 4) with the highest gain.

As there are fewer rules incorporating complex components, rules containing such components should be applied first. Often improvements made with these rules produce superior improvements than those rules involving less complex components. The second rule selection strategy is of a similar philosophy. A rule with the most components is seen as a special case of other rules and is given preference. Rule selection strategy three chooses the rule whose pattern has the longest delay. This pattern can be replaced by a design with a much shorter delay. The fourth strategy uses a greedy method. The rule resulting in the highest gain is chosen. Strictly following this strategy typically produces sub-optimal results. However, it can be combined effectively with the above strategies. For example, if strategy 1 selects more than one rule, strategy 2 could be applied. Finally, if strategy 2 produces more than one rule, the greedy method can be used.

1.2. Strategies for Area

When the area constraint is most important the optimizer attempts to achieve it, possibly at the cost of leaving some paths critical. One plan for area optimization is shown in Figure 6. In order to reduce critical paths, the optimizer must first know how much area can be traded off for delay reduction. Thus the first step is to decrease area along non-critical paths, bringing their delay up to the "critical" point. Figure 7(a) shows the results of applying this

strategy to the circuit of Figure 1(a). If after this step the area falls below the constraint, the optimizer will reduce critical paths and can tradeoff area for time until the area constraint is reached. At that time only strategies that do not increase area will be considered.

If the first step fails to meet the area constraint, the optimizer must continue making time-for-area tradeoffs -- thereby pulling these paths above the "critical" point as in Figure 7(b). This increase in delay can be applied to a single path or spread over multiple paths.

1.2.1. Strategies for Interval Selection

As in the optimization of timing constraints, one can optimize over an entire non-critical path or break it up into intervals. The same methods can be be used to create intervals. For example, the interval creation method of Figure 4 can be used to eliminate intervals containing part of the critical path. Figure 8 shows three intervals created using this method. Interval *b* can be eliminated since it is part of a critical path.

Similarly, the interval creation method of Figure 5 can be used to find intervals where a time for area tradeoff does not increase the worst case delay of a non-critical path. For example in Figure 9 the interval *c* is reduced in area at a cost in time. However, the worst case delay from *I0->Fout* did not increase.

Interval selection tends to be inverse of that for timing optimization. An interval closest to an external output is preferred in area optimization as fewer paths will have their delay increased. Also, intervals with short delays are of particular interest since they can often be factored to save area. As in timing optimization, those intervals containing the the most components provide more opportunities for improvement. This strategy can be combined with the shortest delay strategy to select the interval having the most components and the shortest delay.

1.2.2. Rule Selection

Rule selection strategies used for timing optimization are similar to those for area. In general, rules that reduce delay look for patterns showing long depth with a long delay. Transformations are used that decrease the depth and hence the delay. Rules that reduce area look for patterns having wide breadth and a short delay. They make transformations that increase depth. Thus rules having complex and/or many components are preferred in both cases. However, rules matching patterns with the smallest delays are preferred for area reduction and rules matching patterns with the largest delays are preferred for delay reduction.

1.3. An Algorithm for Constraint Optimization

Having examined a number of strategies for time and area optimization, we present an algorithm for incorporating these strategies.

Optimize_Constraints:

```
If Time Constraint is Most Important
    Reduce_Delay
    Reduce_Area
Else
    Reduce_Area
    Reduce_Delay
```

Reduce_Delay: (Version I)

```
-- This version applies multiple techniques over the same interval
-- before selecting a new interval.
Use Time Analyzer to find Critical Paths
While More Critical Paths [ AND Below Area Constraint (if area is most important) ]
    Choose Path Selection Strategy
    Select a Path not Yet Optimized
    Select Interval Creation Strategy
    Break Up Path into Intervals
    For Each Interval on Selected Path
        Choose Rule Selection Strategy
        Apply Logic Critic Rules
    End For
    While Path is Still Critical [ AND Below Area Constraint (if area is most important) ]
        Choose Time Interval Selection Strategy
        Select Interval not yet optimized
        Choose Rule Selection Strategy
        Apply Time Critic Rules
        If Path is Still Critical
            Use Boolean minimization & refactorization strategy
        Mark interval as optimized
    End While
    Mark Critical Path as optimized
End While
```

Reduce_Area:

```
Use Time Analyzer to find Critical Paths
While More Non-Critical Paths
    Choose Path Selection Strategy
    Select a Path not Yet Optimized
    Break Up Selected Path into Intervals
    For Each Interval on Selected Path
        Choose Rule Selection Strategy
        Apply Logic Critic Rules
    End For
    While Path is Still Non-Critical AND Rule Set Not Exhausted
        Choose Area Interval Selection Strategy
        Select Interval not yet optimized
        Choose Rule Selection Strategy
        Apply Area Critic Rules
        Mark interval as optimized
    End While
    Mark Non-Critical Path as optimized
End While
```

Path Selection Strategies For Time:

1. Worst Critical Path

2. Select Cover

Path Selection Strategies For Area:

1. Most Non-Critical Path

2. Select Cover

Rule Selection Strategies For Time:

1. Most Complex Component
   Most Components
   Highest Gain

2. Longest Delay
   Most Components
   Highest Gain

3. Highest Gain

Rule Selection Strategies For Area:

1. Most Complex Component
   Most Components
   Highest Gain

2. Most Components
   Shortest Delay
   Highest Gain

3. Highest Gain

Time Interval Selection Strategies:

1. Most Critical Paths
   Closest to External Input

2. Longest Delay
   Most Critical Paths
   Closest to External Input

3. Most Components
   Closest to External Input

Area Interval Selection Strategies

1. Interval Closest to External Output

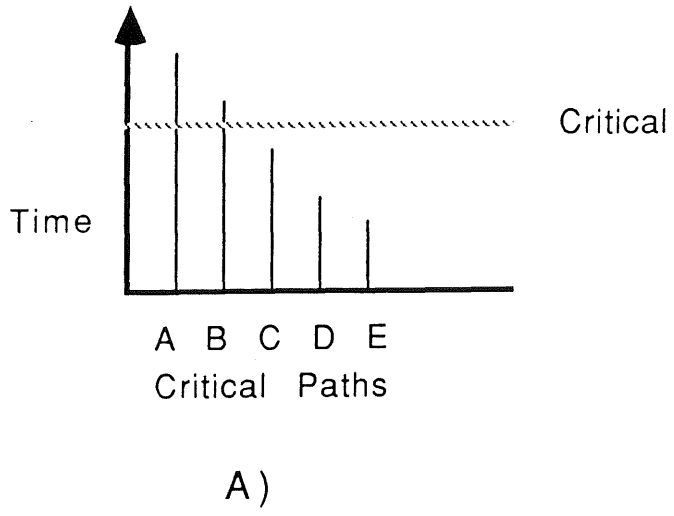
2. Most Components
   Shortest Delay
   Interval Closest to External Output

Reduce_Delay: (Version II)

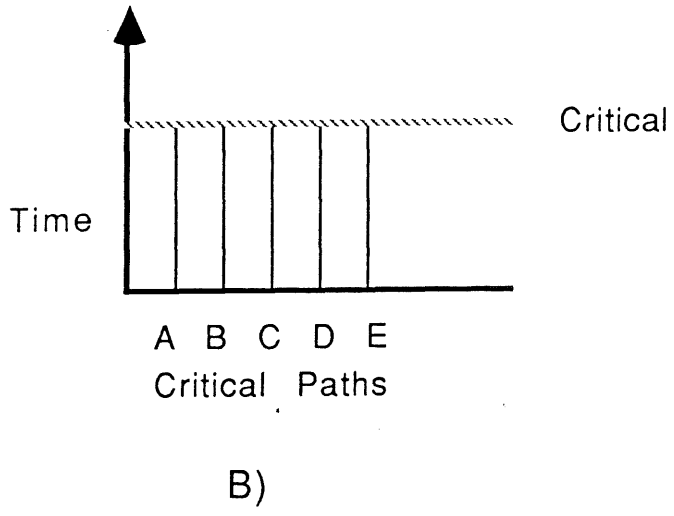
```
-- This version applies the same technique over multiple intervals
-- before selecting a new technique.
Use Time Analyzer to find Critical Paths
While More Critical Paths
    Choose Path Selection Strategy
    Select a Path not Yet Optimized
    Break Up Path into Intervals
    For Each Interval on Selected Path
        Choose Rule Selection Strategy
        Apply Logic Critic
    While Path is Still Critical AND More Optimization Techniques
    [ AND Below Area Constraint (if area is most important) ]
        Choose Time Interval Selection Strategy
        Select Interval not yet optimized
        Choose Time Technique
        Select an Optimization Technique that has not been used yet
        Apply Technique on Interval
        Mark Technique as used
    End While
    Mark Critical Path as optimized
End While
```

Time Technique Selection

1. Swap Pins
2. Time Critic
3. Boolean Minimization & Refactorization



A)



B)

Figure 1: Critical Path Analysis

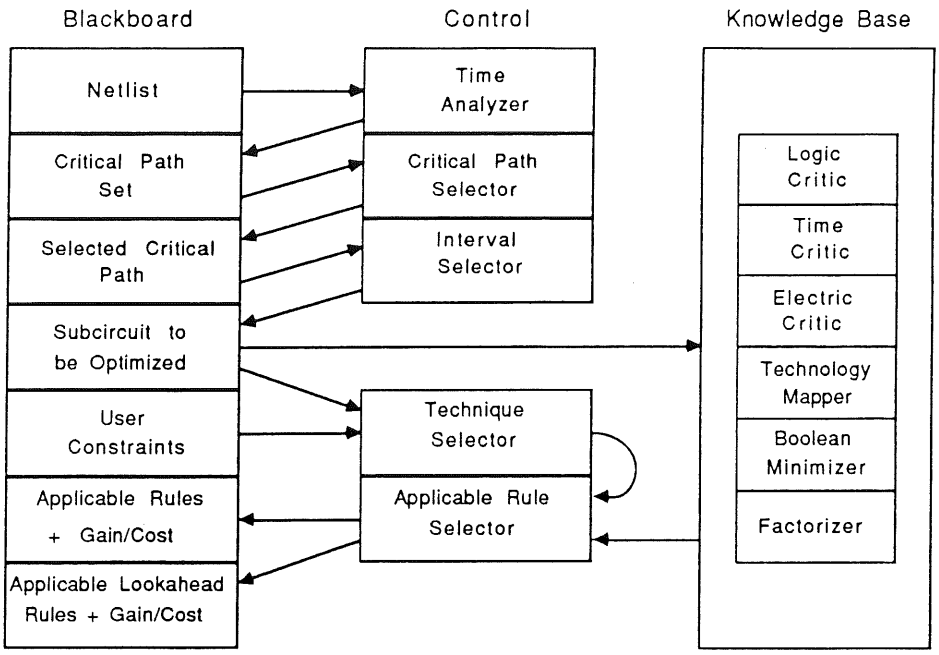
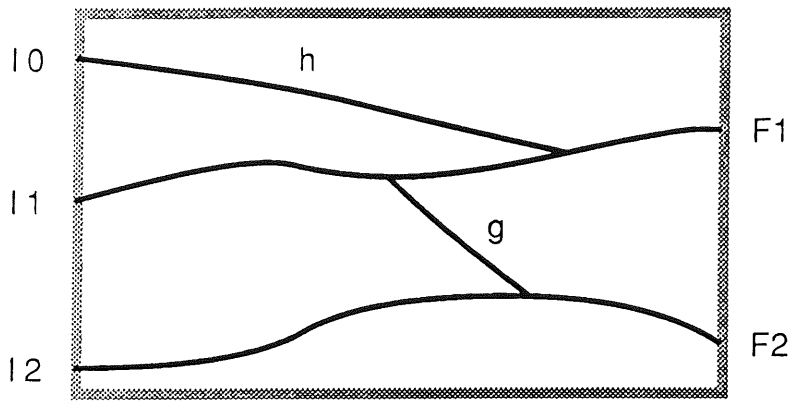
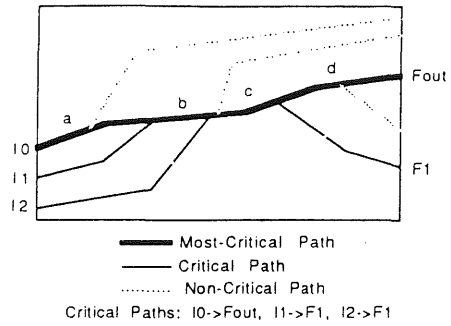


Figure 2: Time Optimizer

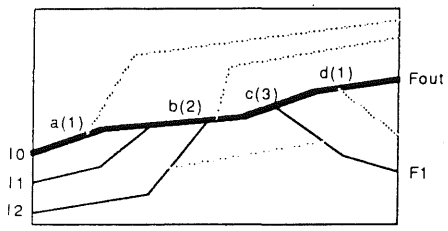


Strategy 1: Optimize All Paths	Strategy 2: Optimize Using Covers
I0->F1	I1->F1
I1->F1	I2->F2
I1->F2	g
I2->F2	h

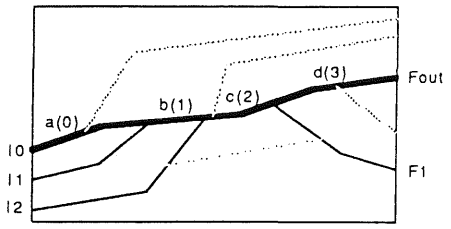
Figure 3: Selection of Covers for Critical Paths



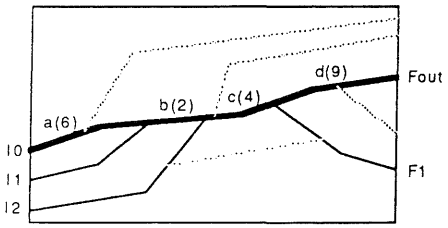
A) Creation of intervals



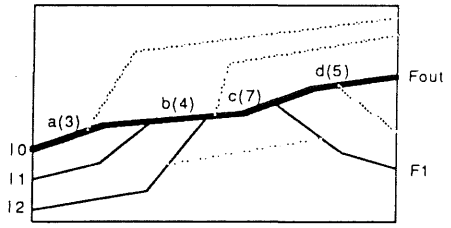
B) Choose interval containing the most critical paths



C) Choose interval closest to external input

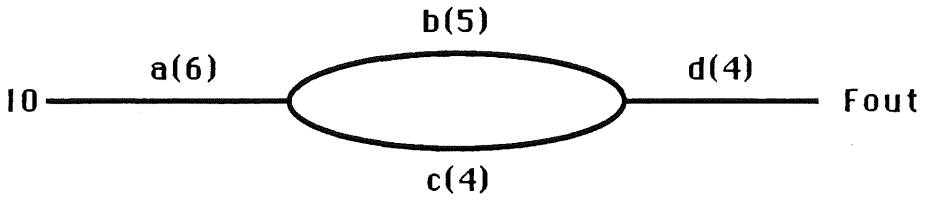


D) Choose interval with most components



E) Choose interval with longest delay

Figure 4: Strategies for Interval Selection



Worst Case Delay from I0->Fout: 15ns
Required Delay: 12ns

Figure 5: Diverging Critical Path

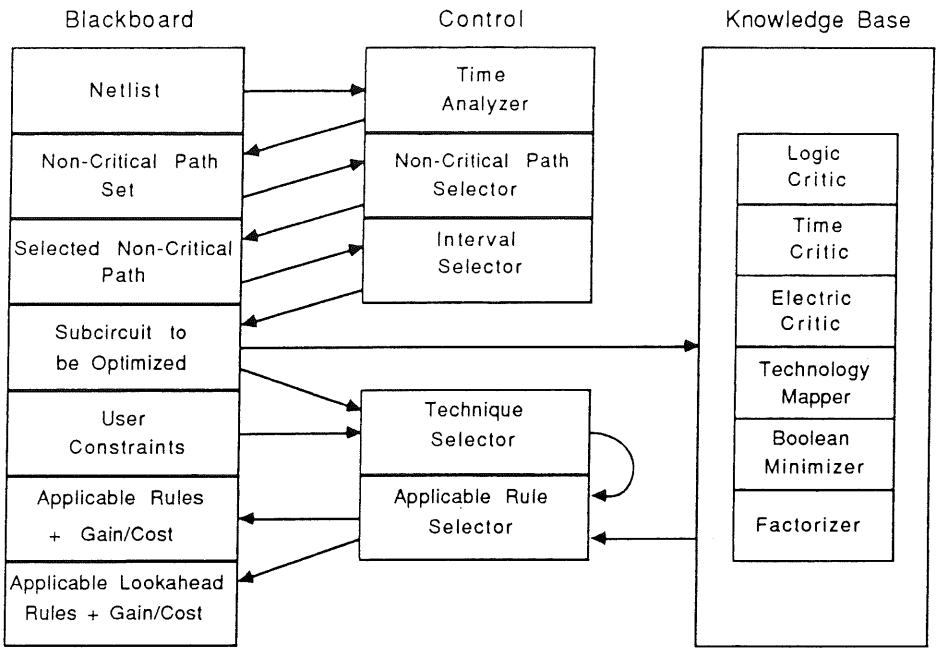


Figure 6: Area Optimizer

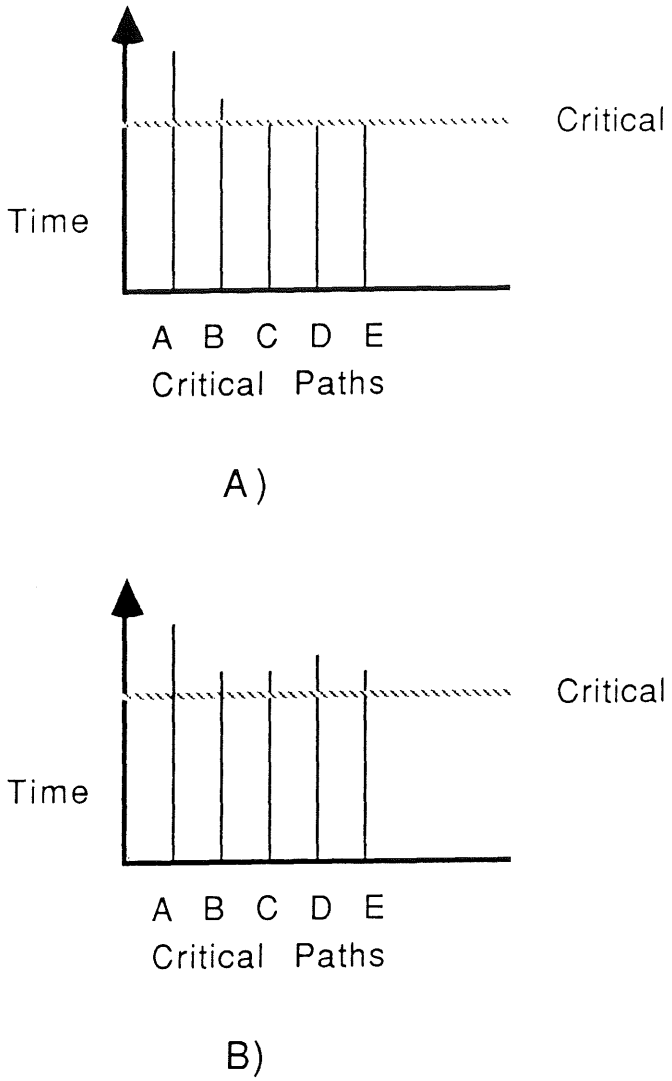
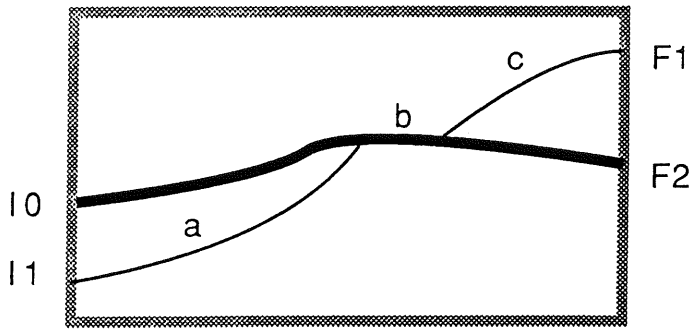
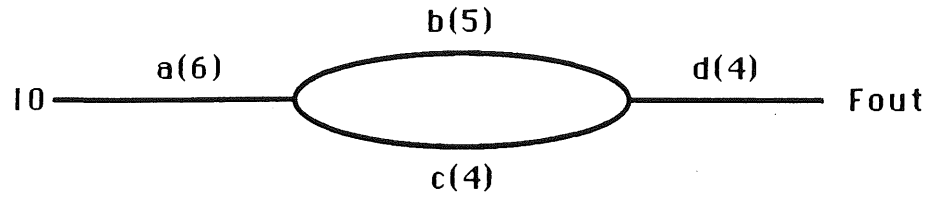


Figure 7: Critical Path Analysis After First Step of Area Optimization

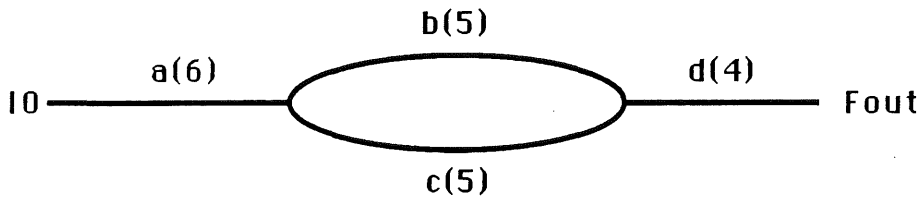


Critical Path: I0->F2
Non-Critical Paths: I1->F1, I1->F2
Intervals: a, b, c
Intervals to be Optimized: a, c

Figure 8: Interval Selection Strategy for Area Optimization



Worst Case Delay from I0->Fout: 15ns
Required Delay: 18ns
Area: 12 cells



Worst Case Delay from I0->Fout: 15ns
Required Delay: 18ns
Area: 10 cells

Figure 9: Optimization of a Diverging Non-Critical Path