Evaluating Optimization Transformations of Behavorial Descriptions

BY

Rajiv Jain and Alice Parker

Technical Report CENG 90-07
February 1990

Electrical Engineering - Systems Department
University of Southern California
Los Angeles, CA. 90089-0781

Evaluating Optimizing Transformations of Behavioral Descriptions[1]

Rajiv Jain † and Alice Parker *

†Department of Electrical and Computer Engineering
1415 Johnson Drive
University of Wisconsin
Madison, WI 53706
(608)262-3610

*Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-0781

November 3, 1989

[1]This research was supported by the Semiconductor Research Corporation (Contract 86-01-075), and the Departments of Air Force, Army and Navy (Contract N00039-87-C-0194).

Evaluating Optimizing Transformations of Behavioral Descriptions

Abstract

To enhance performance or reduce the cost of digital designs, RTL synthesis systems apply high-level graph transformations. Many of these transformations are compiler optimizations and their merits are well known. Benefits obtained from other transformations are not very well understood and it is not clear when and where these data flow graph transformations should be applied. In this paper we use models of cost and performance to predict the effects of three data flow graph transformations on both pipelined and non-pipelined designs. Results obtained using the models are supported by experimental results.

1 Introduction

As a front end to high-level synthesis designers often apply optimizations which, without altering the intended behavior, modify the input specification for a more area-delay efficient implementation. These optimizations include several types of data flow graph transformations [1] [2] [5] [10] [11] [12] [13] [14] including tree height reduction, dead code elimination, loop winding and loop unwinding. Other front-end tasks such as choice of operation bit-width, hierarchical decomposition of operations into sub-operations, design style selection, choice of arithmetic (for example, 2's complement vs. signed magnitude), choice of number system, and algorithm selection also constrain the design. These front-end data preparation tasks are *high-level* design decisions and must be done before the actual synthesis can take place.

Making correct high-level design decisions is important, as an erroneous decision may result in a poor design despite lower-level efforts. The benefits of high-level decisions are obvious for some transformations like dead code elimination, but are not so obvious for most other design decisions. Performing synthesis after every transformation to obtain the actual design space is computationally expensive [8]. In this paper we propose an analytical method for analyzing the impact of these transformations on the area-delay tradeoff curve of the final design and apply the method to three transformation types. This method would greatly help in reducing the design time and in guiding the designer towards a better design. The method uses the area-delay prediction models developed for pipelined and non-pipelined designs [3] [4]. These models predict the lower-bound area-delay tradeoff curve for each design style. The difference in the area-delay tradeoff curves for the original and the transformed data flow graphs can then be computed easily. Thus, in our approach the designer can select or reject the transformation based on the impact of the transformation on the entire area-delay tradeoff curve and not just the impact on a single design. The models used in this paper are restricted in that only functional area of the designs is considered. In practice, register, multiplexer, and wiring area and delay estimates need to be incorporated into the model. Estimators for register and multiplexer area [6] and PLA based control area [7] exist and can be used along with the models used in this paper.

The paper is organized as follows. First we review the lower-bound area-delay prediction models for pipelined and non-pipelined designs (Section 1.1). In Section 2, we will identify three transformations which will be analyzed in this paper, and then using the area-delay models we will study these transformations and predict their impacts on the final designs. In Section 3 we detail some experiments which were conducted

to verify the theory and conclude in Section 4 with future research problems.

1.1 The Lower-Bound Area-Delay Prediction Models

The basic model for pipelined designs [4] predicts the lower-bound clock cycle to be

$$c_p = maximum(d_i) \tag{1.1.1}$$

and the area-delay product to be

$$A \times T_p = c_p \sum_{i=0}^{m-1} (a_i \times n_i) = constant \tag{1.1.2}$$

Here c_p is the clock cycle of the design, d_i (a_i) is the delay (area) of the module which implements operation i, n_i is the effective number of operations of type i in the data flow graph, m is the number of different types of operations in the data flow graph, A is the functional area of the design, and T_p is the delay between two successive initiations of the input data (also a measure of throughput). The above equations state that the lower-bound clock cycle is the delay of the slowest module in the module set, and the lower-bound area-delay product of the pipelined designs is a constant.

A factor which affects the throughput of pipelined designs is resynchronization. Performance degradation due to resynchronization depends on the number of time steps the data flow graph is partitioned into. From [8], we know that with resynchronization rate of ρ, $0 \leq \rho \leq 1$, the average performance of a pipelined design is given by

$$T_{p-avg} = (1 + \rho(\lceil \frac{P}{l} \rceil - 1))lc_p \tag{1.1.3}$$

where P is the number of stages of the pipeline, and l is the initiation interval of the pipeline. In this case, the area-delay product of the pipeline is given by

$$A \times T_{p-avg} = (1 + \rho(\lceil \frac{P}{l} \rceil - 1))lc_p \times \sum_{i=0}^{m-1} (a_i \times o_i) \tag{1.1.4}$$

Since n_i/l is a lower-bound for o_i we can substitute for o_i to obtain

$$A \times T_{p-avg} = (1 + \rho(\lceil \frac{P}{l} \rceil - 1))c_p \times \sum_{i=0}^{m-1} (a_i \times n_i)$$

The lower-bound clock cycle for designs with resynchronization cannot be computed using Equation 1.1.1. However, we can bound c_{pr} (clock period with resynchronization)

$$c_{pr} = maximum(\frac{C}{p}, d_i)$$

where C is the critical path delay. We can enumerate p from one to the number of operations in the data flow graph and can enumerate l from one to p for each value of p, in order to generate the AT curves.

For non-pipelined designs the model [3] predicts the clock cycle to be

$$c_{np} = maximum(C/N, maximum(d_i)) \tag{1.1.5}$$

and the area-delay to be

$$A \times T_{np} = c_{np} \sum_{i=0}^{m-1} (a_i \times n_i) = c \times constant \tag{1.1.6}$$

where T_{np} is the circuit delay, C is the critical path delay, and N is the number of time steps that the data flow graph is partitioned into. Computation of critical path can be performed in two ways.

1. Summation of delays along the path having maximum delay in the data flow graph.
2. Decomposition of the data flow graph operations into bits and summation of bit operations along the path having maximum delay. This method is desirable in special cases where composure delays are less than the sums of individual delays. An example of this is the chaining of two additions into one time step. The second addition can start as soon as the lowest-order bit of the first addition becomes available, making the total delay slightly longer than the delay of single addition, but smaller than delay of two consecutive additions.

If any module is pipelined d_i includes delays of individual stages in the module. If modules can be carried over several time steps, the clock cycle can be arbitrarily small, and in this case the clock cycle is usually selected a priori and the effective number of modules is $o_i = \left\lceil \frac{n_i \times d_i}{k \times c} \right\rceil$,where $k = l$ for pipelined designs and $k = N$ for non-pipelined designs.

2 Evaluating Transformation Impact

2.1 Transformations Under Consideration

The evaluation technique presented in this paper is general enough to be applied to many transformations and design style. The following three transformation types are analyzed in this paper.

1. Transformations which alter the number of operations in data flow graph (for example, common subexpression elimination, strength reduction and dead code elimination [1]),
2. transformations affecting critical path delay (for example, a tree height reduction transformation [13]), and
3. hierarchical decomposition of operations into sub-operations.

We will now characterize the effects of the above mentioned transformations on the area-delay curves for each design style.

2.2 Altering the Number of Operations

First, we analyze the transformations which alter the number of operations in the data flow graph as their impact is easiest to quantify. These transforms always improve the best-case cost and/or performance if the number of operations is decreased. We use our models to verify this intuitive result. From Equations 1.1.2 and 1.1.6 we observe that the area-delay product of the design is proportional to the number of nodes in the data flow graph. Any increase (decrease) in the number of nodes of a given type in the data flow graph increases (decreases) the area-delay product of the design. That is, for the same performance requirement, more area will be required if the number of operations is increased. This is true for both pipelined and non-pipelined design styles (assuming there is no change in the critical path delay).

Analyzing the impact of a transformation on the area-delay tradeoff curve is the same as performing a sensitivity analysis of Equations 1.1.2 and 1.1.6. For example, for

non-pipelined designs varying n_i while keeping other parameters of Equation 1.1.6 constant implies that nodes of the existing operation type are deleted or added to the data flow graph without changing the critical path delay. Here we perform the analysis only on the non-pipelined design style as the pipelined design style can be similarly analyzed. We have seen that the non-pipelined area-delay tradeoff curve consists of two parts, namely, $C/N > maximum(d_i)$ which is given by a vertical line and $C/N < maximum(d_i)$ which is the sloping line. If we fix all parameters in Equation 1.1.6 and vary n_i alone, we note that the sloping part of the AT curve shifts away from the origin if n_i is increased or moves towards the origin if n_i is decreased. The vertical line does not appear to move although individual design points shift up the curve. Figure 1a shows the lower-bound area-delay tradeoff curves before and after an increase in n_i. In this figure we observe that increasing n_i has moved the sloping part of the area-delay tradeoff curve away from the origin towards the inferior part of the design space. Further, all design points of the transformed data flow graph are inferior to the design points of the untransformed data flow graph. Similar results are obtained when n_i and m are increased, keeping other parameters constant, is shown in Figure 1b. Figure 1c shows the results of increasing n_i, m and $maximum(d_i)$ simultaneously.

In general, one can state that any transformations which reduce the number of operations in the data flow graph without changing the critical path are favorable, and ones which increase the number of operations with no change in critical path are unfavorable.

2.3 Reducing Critical Path Length

Reducing critical path delay has interesting effects on the predicted area-delay curves. In order to analyze the critical path reduction transformation we first assume that the number of nodes of each operation type remains unchanged before and after the transformation. Evaluating the effects of varying n_i, m, $maximum(d_i)$ and critical path delay C in the area-delay equations simultaneously can be analyzed similarly. Figure 2 shows an example of reducing critical path delay in which the number of nodes in the data flow graph remains unaltered. We show in this section that tree height reduction has no impact on the pipelined design style unless resynchronization is considered. For pipelined designs, the lower-bound area-delay product is dependent only on the area and delay (as clock cycle is dependent on the delay) of the modules used for implementation, and the number of nodes of each type in the data flow graph, and is independent of the critical path delay. This implies that a change in the critical

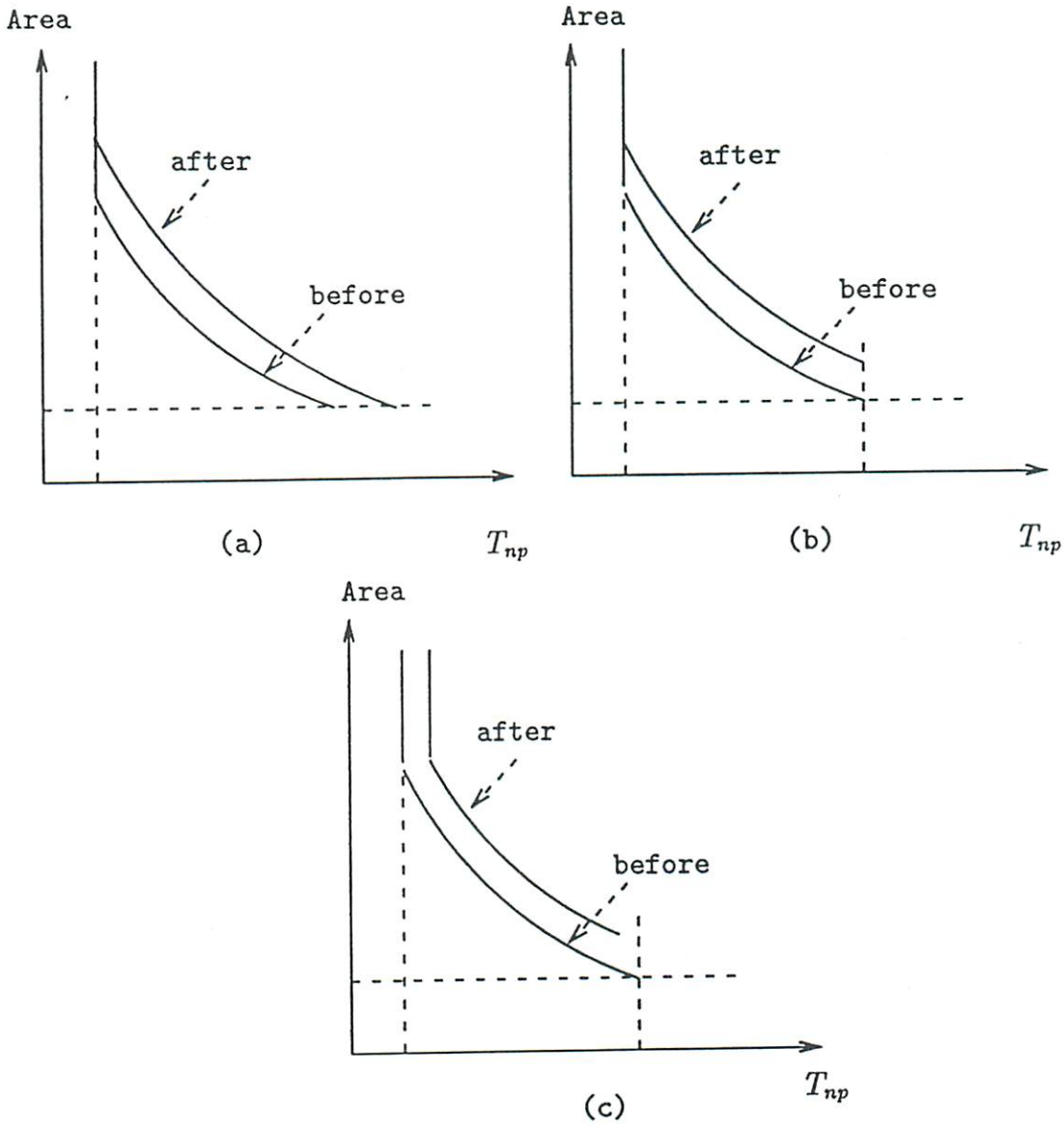


Figure 1: Effect of Increase in Node Count for Non-Pipelined Designs

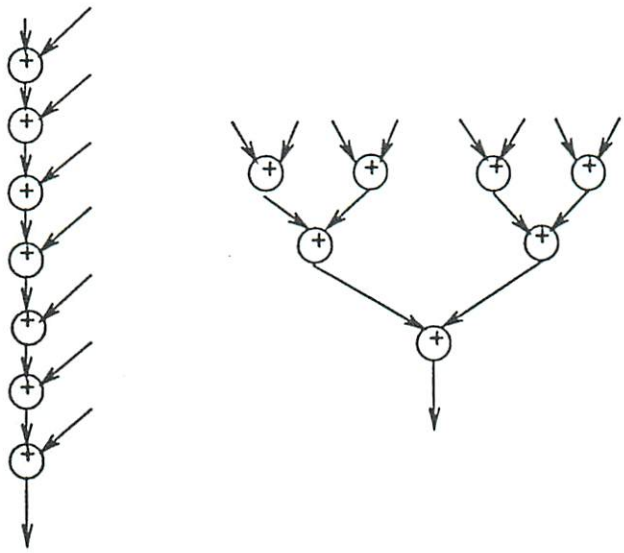
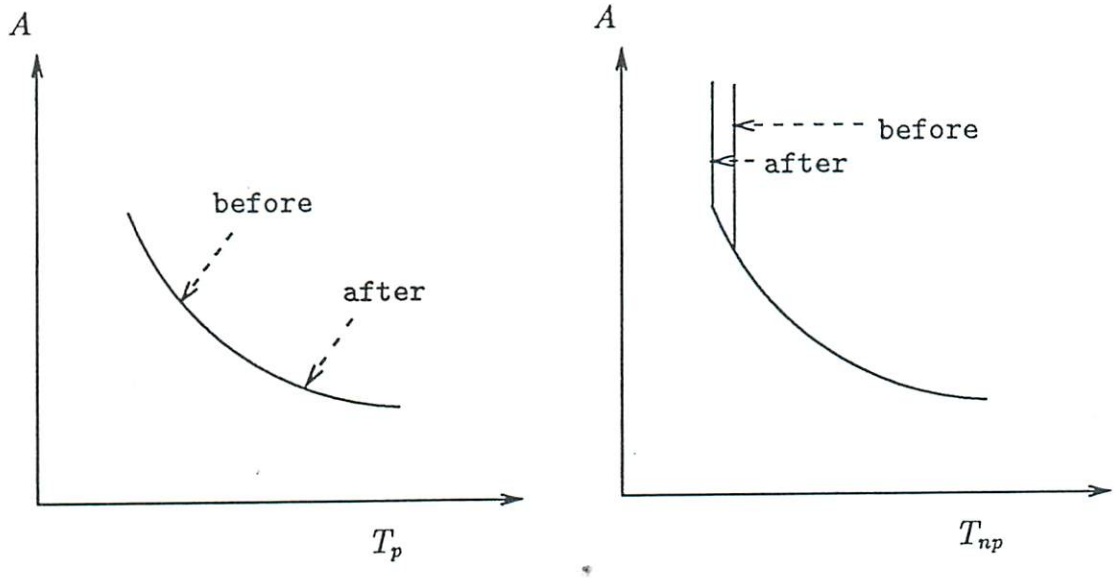


Figure 2: A Critical Path Reduction Transformation



(a) Pipelined (b) Non-Pipelined

Figure 3: Predicted effect of Critical Path Reduction

path delay does not impact the area-delay tradeoff curve for pipelined design. Figure 3a shows the impact of critical path reduction on the pipelined design style, i.e. there is no difference. Changes to the critical path will only affect the area-delay curve of the pipelined design if resynchronization occurs. Figure 2 was synthesized by Sehwa for different resynchronization rates. These results are tabulated in Table 1 and show that by reducing the critical path delay, performance of a pipelined design can be improved as the resynchronization rate increases.

For non-pipelined designs, the clock cycle depends on the critical path delay. From Equation 1.1.5 we know that $c_{np} = maximum(C/N, maximum(d_i))$. By reducing the critical path delay C, C/N will approach $maximum(d_i)$ for a smaller number of partitions N. Also, reducing the length of the critical path increases parallelism in the data flow graph allowing more high-performance designs to be generated [13]. Reducing critical path delay and increasing parallelism in the data flow graph increases the number of possible designs which can be generated and a larger design space can be explored for a faster design. Figure 3b shows the impact of critical path reduction on the non-pipelined design style. However, whereas the scope for parallelism in the data flow graph has increased, the lower-bound $A \times T_{np}$ remains same (assuming there is no change in the number of nodes in the data flow graph).

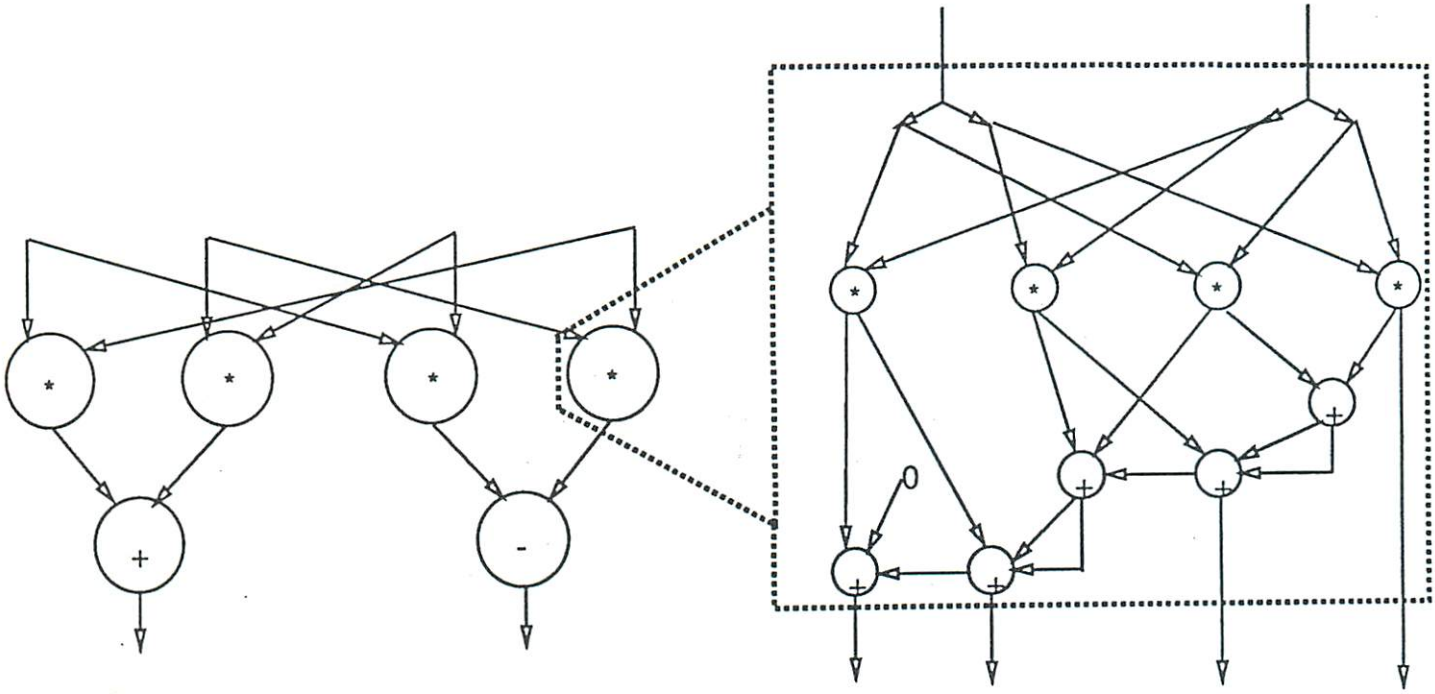
It has been observed in the two examples from Trickey [13] (Figures 27 and 35) that reduction in tree height increased the number of nodes of the data flow graph, which moves the lower part of the area-delay tradeoff curve away from the origin towards the inferior design space. It is not clear if tree height reduction will always increase the number of nodes in the data flow graph.

2.4 Hierarchical Decomposition

The effect of transformations which decompose operations into sub-operations is more complicated. Figure 4 shows an example hierarchical decomposition where a 16-bit multiplication node can be replaced by a subgraph composed of 8-bit multiplication and addition nodes.

Resynchronization (%)	Area (A)	Delay (T_p)	
		Before	After
0	29400	340	340
	16800	680	680
	12600	1020	1020
	8400	1360	1360
	4200	2380	2380
10	29400	544	408
	16800	884	748
	12600	1224	1122
	8400	1496	1360
	4200	2380	2380
20	29400	748	476
	16800	1088	748
	12600	1428	1224
	8400	1632	1360
	4200	2380	2380
30	29400	952	544
	16800	1292	884
	12600	1632	1326
	8400	1768	1360
	4200	2380	2380
40	29400	1156	612
	16800	1497	952
	12600	1837	No Result
	8400	1905	1360
	4200	2380	2380

Table 1: Effect of tree height reduction on an example with resynchronization



(a) Complex multiplication

(b) 16-bit multiplication using 8-bit multipliers

Figure 4: Hierarchical Decomposition

2.4.1 Pipelined Designs

Let us examine the pipelined case first. Pipelined designs are characterized by Equations 1.1.1 and 1.1.2. Let the data flow graph before and after the transformation be represented by the following two equations respectively:

$$A_1 \times T_{p1} = c_{p1} \sum_{i=0}^{m1-1} (a_i \times n_i) \tag{2.4.7}$$

$$A_2 \times T_{p2} = c_{p2} \sum_{i=0}^{m2-1} (a_i \times n_i) \tag{2.4.8}$$

where $m1$ is the number of operation types before decomposition and $m2$ is the number of operation types after decomposition. Decomposition of an operation *may* produce a change in the clock cycle, in the number of different types of operations in the data flow graph and in the area of the design. The lower-bound clock cycle is equal to the delay of the slowest operation in the data flow graph. A change in the clock cycle will occur if the slowest operation in the data flow graph is decomposed and the sub-operations produced by the decomposition have a smaller delay than the original operation. Further, as each new sub-operation will have a delay less than or equal to the delay of the original operation, the new clock cycle c_2 will be less than or equal to the old one, c_1. Whenever $c_2 < c_1$, then the new data flow graph might allow designs with higher throughput. The area-delay characteristic of the new data flow graph will depend on the amount of increase in the summation term on the right hand side. If

$$\sum_{i=0}^{m2-1} (a_i \times n_i) < \frac{c_{p1}}{c_{p2}} \sum_{i=0}^{m1-1} (a_i \times n_i) \tag{2.4.9}$$

then the new design will be more area-delay efficient than the original data flow graph. Further, if $\sum_{i=0}^{m1-1} a_i > \sum_{i=0}^{m2-1} a_i$, then cheaper designs might be generated for the new data flow graph. For each hierarchical decomposition employed, the effects can be computed using Equation 2.4.9.

The design characteristics can be improved by decreasing the clock cycle if the areas of the designs before and after the transformation are related by Equation 2.4.9. For pipelined designs, we know that the clock cycle is equal to the delay of the slowest module in the module set (Equation 1.1.1). Decreasing the clock cycle can be achieved either by using a faster module for the slowest operation or decomposing the slowest node of the data flow graph into faster sub-operations. For example, in a data flow graph with 16-bit multiplications and 16-bit additions, the 16-bit multiplication nodes may be decomposed into 8-bit or 4-bit multiplication and addition nodes. As

our example does not consider the register and multiplexer area/delay the example suggests an indiscriminate decomposition of the current biggest operation till the data flow graphs contains nothing but basic primitives. This, of course, is not necessarily true in practice.

The main advantage of decomposing operations into sub-operations is the increase in potential parallelism and resource sharing which helps the designer in searching a larger design space for a higher performance design or a cheaper design.

2.4.2 Non-Pipelined Designs

The only difference in the analysis for pipelined and non-pipelined designs is the critical path delay factor which occurs in the non-pipelined model. If the delay of the operation being decomposed is the same as the critical path delay of the decomposed sub-graph then the non-pipelined design is similar to the pipelined case. Otherwise, the effect of the change in critical path delay can be easily understood by considering the area-delay equations for the two data flow graphs:

$$A_1 \times T_{np1} = maximum(C_1/N, maximum(d_i)) \sum_{i=0}^{m1-1} (a_i \times n_i)$$

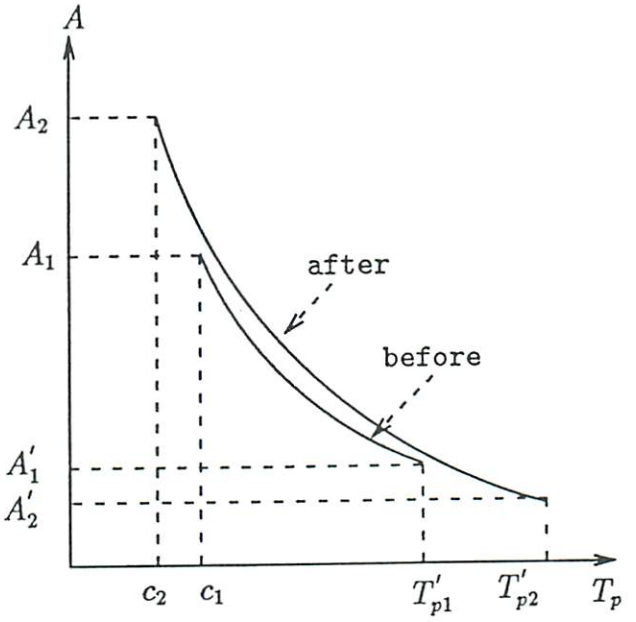
$$A_2 \times T_{np2} = maximum(C_2/N, maximum(d_i)) \sum_{i=0}^{m2-1} (a_i \times n_i)$$

An analysis similar to that for pipelined design can be easily done and the break point for the improvement in the area-delay characteristic occurs when

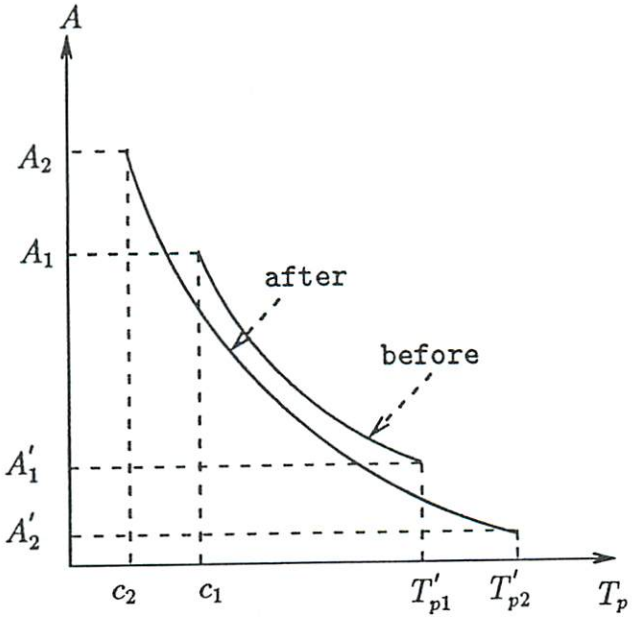
$$\sum_{i=0}^{m2-1} (a_i \times n_i) < \frac{maximum(C_1/N, maximum(d_i))}{maximum(C_2/N, maximum(d_i))} \sum_{i=0}^{m1-1} (a_i \times n_i) \qquad (2.4.10)$$

Similar to the pipelined case, decomposing operations helps in exploring a larger design space for faster and cheaper designs. An example analysis for bit-width hierarchical decomposition (Figure 4) for pipelined and non-pipelined designs is given in the next section.

Figure 5a (Figure 5b) shows the predicted effect of hierarchical decomposition on the pipelined design style where the slowest operation has been decomposed into smaller suboperations with an increase or decrease in area-delay product of the design. In Figure 5, $A_1' = \sum_{i=0}^{m1-1} a_i$, and $A_2' = \sum_{i=0}^{m2-1} a_i$.



(a) Increase in area



(b) Decrease in area

Figure 5: Effect of Hierarchical Decomposition on Pipelined Designs

Function	Area mil^2	Delay nS	Bit Width
addition	2000	225	8
addition	4200	340	16
subtraction	4200	340	16
multiplication	13800	250	8
multiplication	49000	375	16

Table 2: Design Library

3 Experiments and Results

Experiments were conducted to verify the ideas given in the preceding section. For every transformation type a data flow graph was selected and synthesized before applying the transformation and again after applying the transformation. For every such pair of data flow graphs pipelined and non-pipelined designs were synthesized. Sehwa [8] and MAHA [9] were the pipelined and non-pipelined synthesis tools used in the experiments. All experiments were conducted using 16-bit modules given in Table 2, except in hierarchical decomposition experiments where 16-bit and 8-bit components were used.

To demonstrate the effect of increase in number of nodes in the data flow graph we used the *presum* computation and the data flow graphs given in Figures 32 and 35 of [13]. The number of addition nodes in the data flow graph prior to applying the transformation was seven and after the transformation it was twelve. The critical path had seven addition nodes before the transformation and three addition nodes after the transformation. The results for pipelined and non-pipelined synthesis are given in Figures 6 and 7 and the effects are as predicted.

To evaluate the effect of reducing the critical path delay we used the simple data flow graphs given in Figure 2. The results produced by Sehwa, MAHA and prediction models are shown in Figures 8 and 9 respectively. The results are in consonance with the prediction.

The 16-bit complex multiplication data flow graph shown in Figure 4a is used to demonstrate the effect of hierarchical decomposition. Figure 4b shows the decomposition of a 16-bit multiplication node into 8-bit multiplications. The original data

flow graph has four 16-bit multiplication, one 16-bit addition and one 16-bit subtraction node. After substituting Figure 4b for all 16-bit multiplication nodes, the data flow graph has 16 8-bit multiplications, 20 8-bit additions, one 16-bit addition and one 16-bit subtraction node. The critical path delay of the original data flow graph consists of one 16-bit multiplication and one 16-bit addition delay (assuming a 16-bit addition delay is equal to a 16-bit subtraction delay). The transformed data flow graph has a critical path delay of one 8-bit multiplication, five 8-bit additions and one 16-bit addition delay.

Using Equations 2.4.7 and 2.4.8 for pipelined designs the area-delay characteristic of the two data flow graphs can be easily computed. For the original data flow graph given in Figure 4a,

$$A_1 \times T_{p1} = c_{p1}(4 \times a_{16-bitmultiplier} + a_{16-bitadder} + a_{16-bitsubtractor})$$

Using the area and delay of modules (a_1, m_1) given in Table 2, $c_{p1} = maximum\ (340, 375) = 375$ and

$$A_1 \times T_{p1} = 375(4 \times 49000 + 4200 + 4200) = 2.15 \times 10^7 mil^2 ns$$

Similarly, for the data flow graph with the decomposed operations, we have

$$A_2 \times T_{p2} = c_{p2}(16 \times a_{8-bitmultiplier} + 20 \times a_{8-bitadder} + a_{16-bitadder} + a_{16-bitsubtractor})$$

$$= 340(16 \times 13800 + 20 \times 2000 + 4200 + 4200) = 9.15 \times 10^7 mil^2 ns$$

From the above analysis of the two data flow graphs we see that the original design has a better area-delay characteristic than the second. However, the second design has the potential for a higher throughput than the first and it allows to user to explore a larger design space, especially in the spectrum of designs with cheap area, for a design satisfying the area constraint. The results of the pipeline synthesis program Sehwa are given in Figure 10 and support the analysis.

Figure 11 shows the results produced for the hierarchical decomposition transformation by MAHA. The original graph had a critical path delay of $C_1 = 715nS$, and $c_{np1} = maximum\ (715/N, maximum(375, 340))$. The critical path delay of the new data flow graph is $C_2 = 1815nS$, and $c_{np2} = maximum\ (1815/N,\ maximum(225, 250, 340))$. Thus,

$$A_1 \times T_{np1} = c_{np1}(4 \times 49000 + 4200 + 4200) = 204400 \times c_{np1}$$
$$A_2 \times T_{np2} = c_{np2}(16 \times 13800 + 20 \times 2000 + 4200 + 4200) = 269200 \times c_{np2}$$

For any $N < 5$, $c_{np1} < c_{np2}$ and $A_1 \times T_{np1} < A_2 \times T_{np2}$. For $p > 4$, $c_{np1} = 375nS$ and $c_{np2} = 340nS$. Substituting these values we again get the result $A_1 \times T_{np1} < A_2 \times T_{np2}$. Thus, the original design has a better lower-bound area-delay product than the design for the new data flow graph. However, it is possible that the decomposition would allow more actual module-optimal designs to be synthesized.

4 Summary

In this paper we have demonstrated a technique for evaluating the impact of data flow graph transformations on pipelined and non-pipelined design styles. We have analyzed three transformations and verified the theoretical predictions with experimental results. The method is general enough to be applied to several other data flow graph transformations as well.

References

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[2] E. Girczyc. Loop Winding - A Data Flow Approach to Functional Programming. In *Proceedings of the IEEE International Symposium on Circuits and Systems.* IEEE, May 1987.

[3] R. Jain, M. J. Mlinar, and A. C. Parker. Area-Time Model for Synthesis of Non-Pipelined Designs. In *Proceedings of the International Conference on Computer-Aided-Design.* ACM/IEEE, November 1988.

[4] R. Jain, A. C. Parker, and N. Park. Predicting Area-Time Tradeoffs for Pipelined Designs. In *Proceedings of the 24th Design Automation Conference.* ACM/IEEE, June 1987.

[5] D. J. Kuck. *The Structure of Computers and Computations - Volume 1.* John Wiley & Sons, New York, 1978.

[6] M. J. Mlinar and A. C. Parker. Estimating Register and Multiplexer Costs in VLSI Design. Technical report, Department of Electrical Engineering, University of Southern California, 1988.

[7] M. J. Mlinar and A. C. Parker. PASTA: A Model for Estimating Control Area. Technical report, Department of Electrical Engineering, University of Southern California, 1988.

[8] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on Computer-Aided-Design*, 7(3), March 1988.

[9] A. C. Parker, J. Pizarro, and M. J. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proceedings of the 23rd Design Automation Conference.* ACM/IEEE, June 1986.

[10] J. Rabaey and H. De Man. Computer Aided Design of Digital Signal Processing Systems: The IMEC View. In *Proceedings of the International Conference on Computer Design.* ACM/IEEE, October 1987.

[11] E. A. Snow, D. P. Siewiorek, and D. E. Thomas. A Technology-Relative Computer-Aided Design System: Abstract Representations, Transformations, and Design Tradeoffs. In *Proceedings of the 15th Design Automation Conference.* ACM/IEEE, 1978.

[12] D. Thomas. *The Design and Analysis of an Automated Design Style Selector.* PhD thesis, Department of Electrical Engineering, Carnegie-Mellon University, April 1977.

[13] H. Trickey. *Compiling Pascal Programs into Silicon.* PhD thesis, Department of Computer Science, Stanford University, July 1985.

[14] R. A. Walker and D. E. Thomas. Behavioral Transformation for Algorithmic Level IC Design. *IEEE Transactions on Computer-Aided-Design*, 8(10), October 1989.

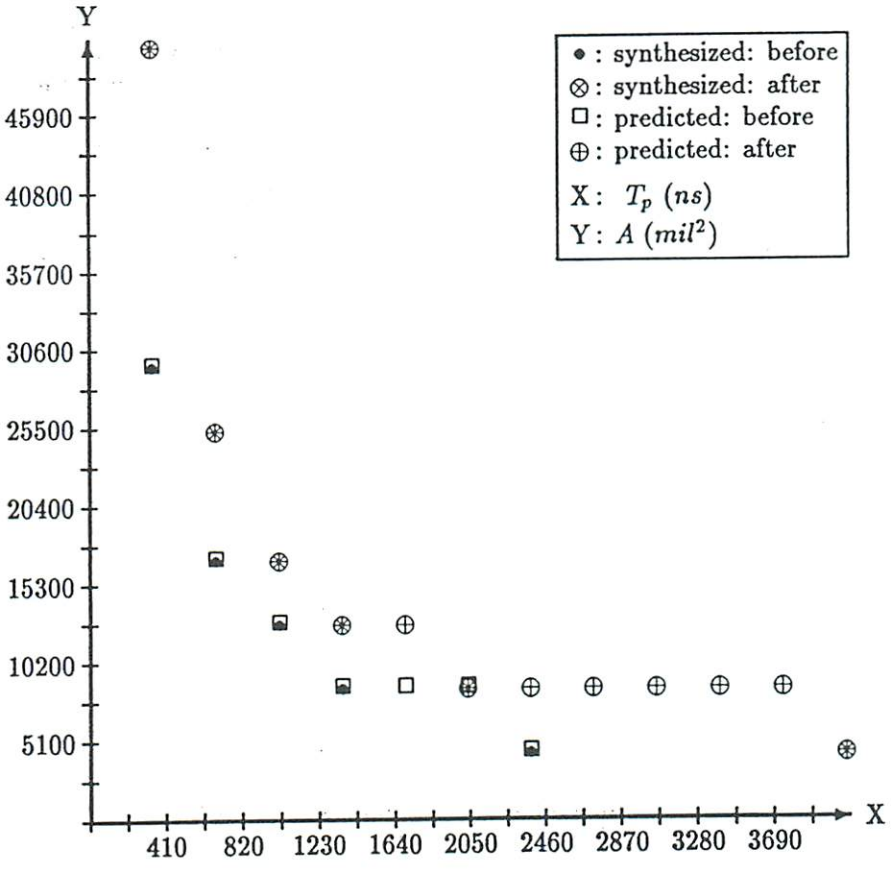


Figure 6: Pipelined Designs: Increase in Node Count

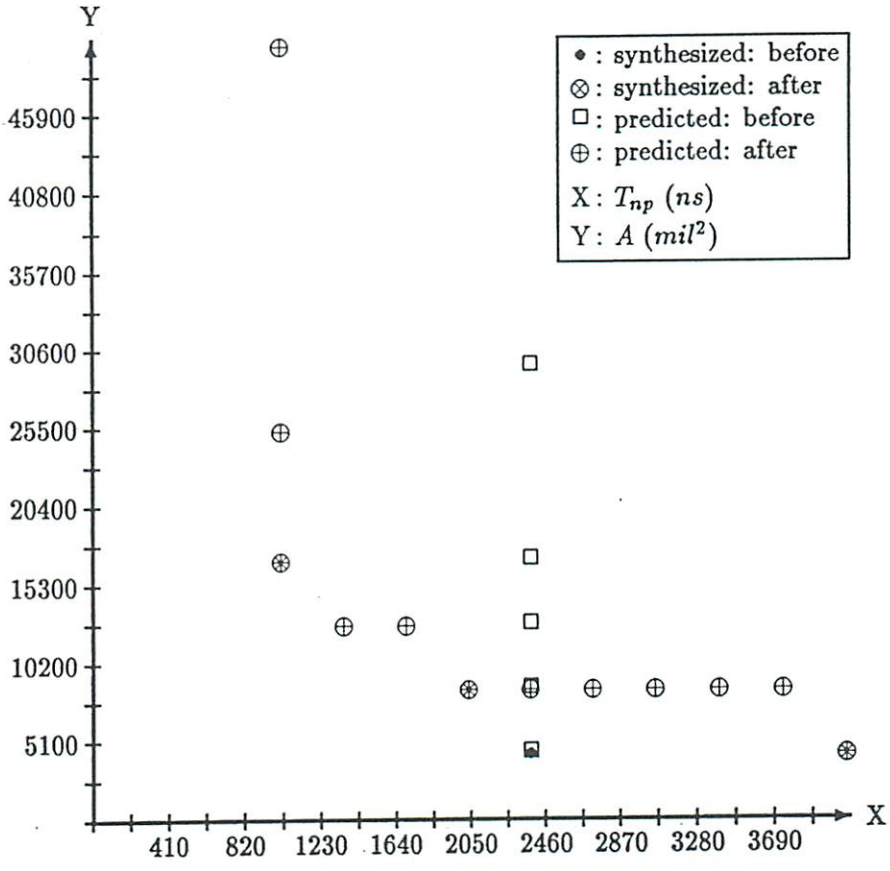


Figure 7: Non-Pipelined Designs: Increase in Node Count

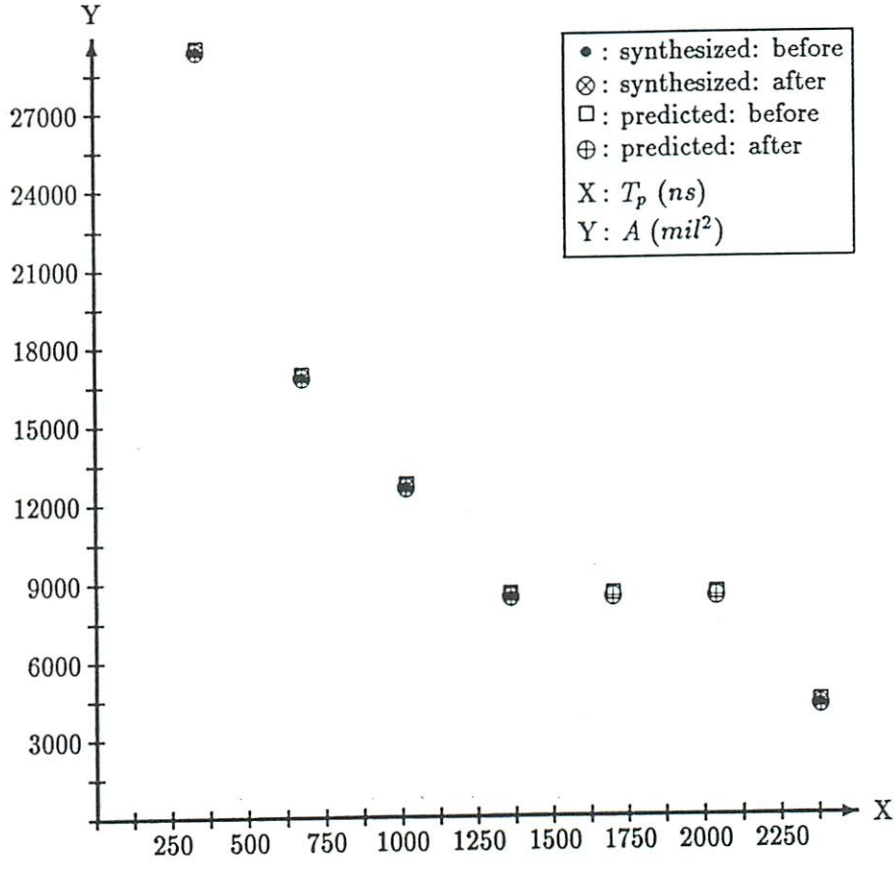


Figure 8: Pipelined Designs: Tree Height Reduction Transformation

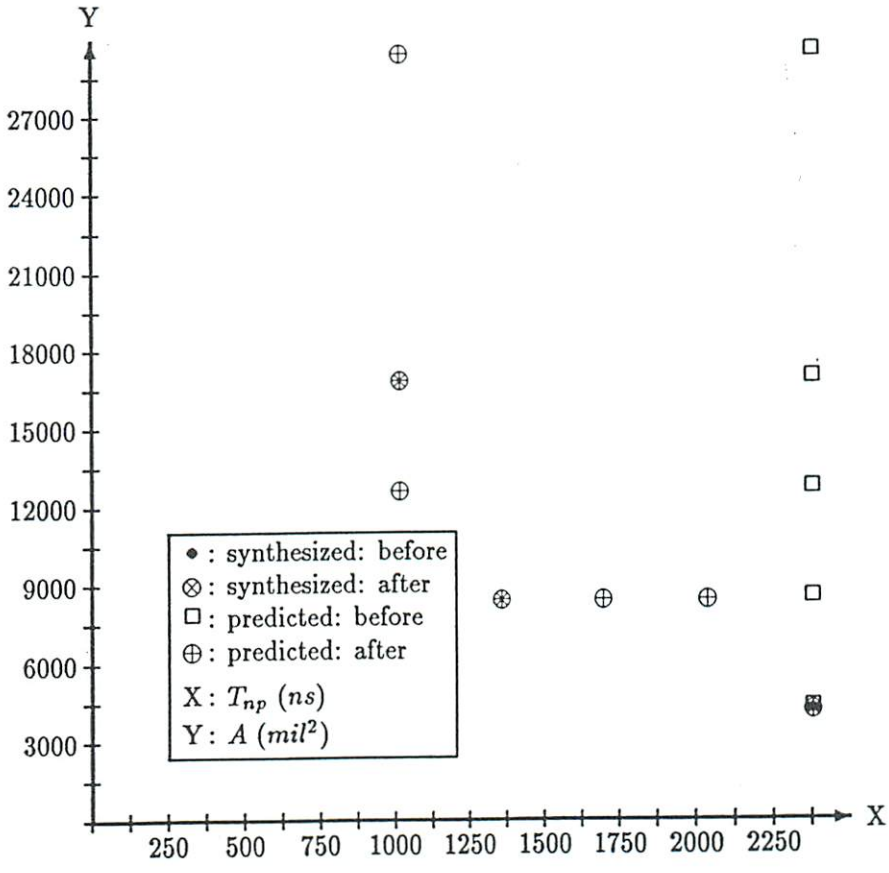


Figure 9: Non-Pipelined Designs: Tree Height Reduction

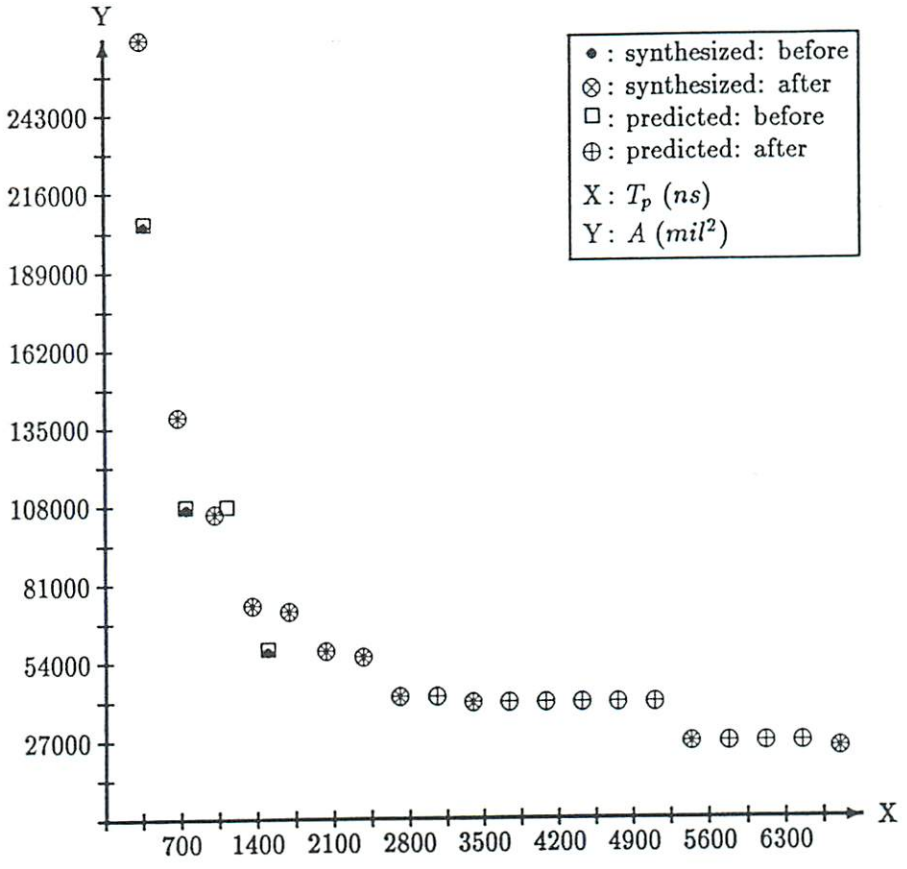


Figure 10: Pipelined Designs: Hierarchical Decomposition

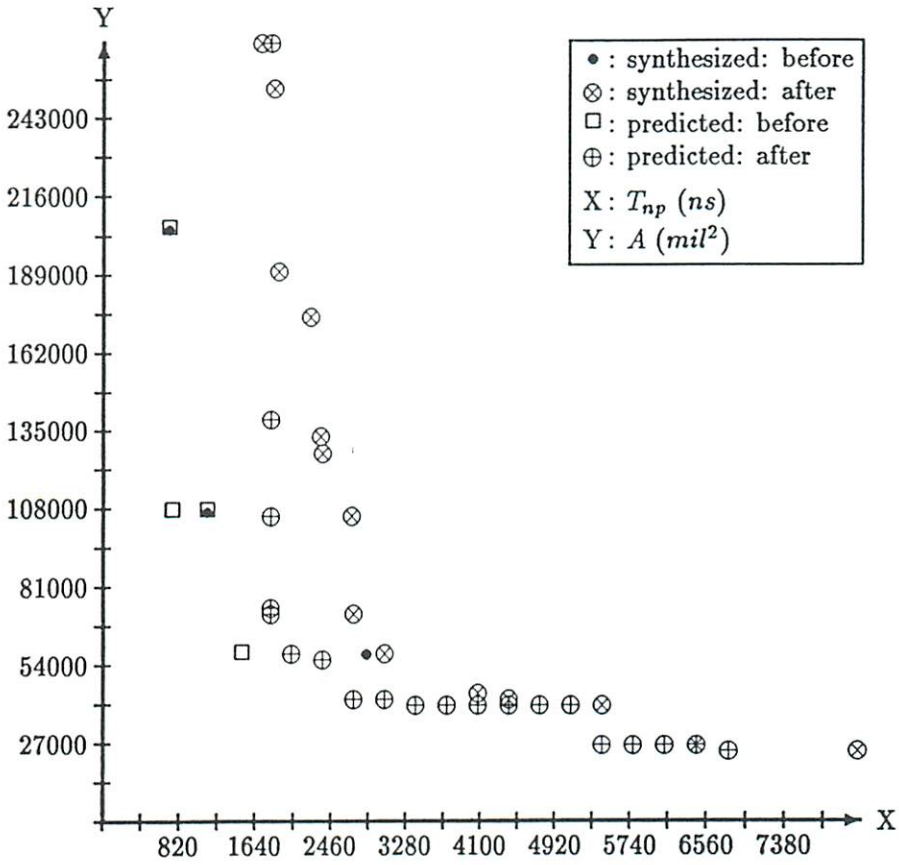


Figure 11: Non-Pipelined Designs: Hierarchical Decomposition