Timing Distribution in VHDL Behavioral Models

by

Ashish Gadagkar

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

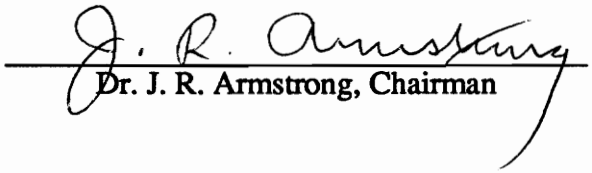
in partial fulfillment of the requirements for the degree of

Master of Science

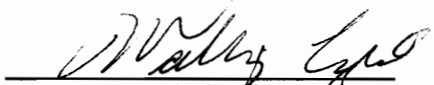
in

Electrical Engineering

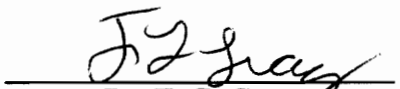
APPROVED



Dr. J. R. Armstrong, Chairman



Dr. W. R. Cyre



Dr. F. G. Gray

April, 1992

Blacksburg, Virginia

Acknowledgements

I thank my advisor Dr. James Armstrong for his support and encouragement. Studying under his guidance has been a very rewarding experience.

I thant Dr. W. R. Cyre and Dr. F. G. Gray for serving on the committee to review my thesis. I am grateful to Dr. F. G. Gray and Dr. J. G. Tront for providing me with invaluable knowledge and opportunities to learn.

I thank my friends and collegues for their support and encouragement.

I would like to dedicate this work to my wife, Priya and my parents for being loving and supportive through all my endeavors.

Timing Distribution in VHDL Behavioral Models

by

Ashish Gadagkar

Dr. James R. Armstrong, Chairman

Electrical Engineering

(ABSTRACT)

This thesis describes a new CAD tool, TIMESPEC, developed for solving the *timing distribution problem* of allocating realistic delays to the internal primitives of a digital device. The inconsistencies in the manufacturer's specifications are also detected and corrected. Therefore, TIMESPEC enables the use of *imbedded timing* in behavioral VHDL models, thereby providing accurate VHDL descriptions. Due to this modeling methodology, the end-to-end delays for all the paths in the digital device are made available. Also, due to the *Register Transfer Level* (RTL) of abstraction, which is represented by a *process model graph*, there is close correspondence with the actual device being modeled. Thus a better insight into the timing problems is provided and synthesis is possible from the resulting models.

A *linear programming* approach is employed for solving the *timing distribution problem.* An interface is provided with an X-windows based graphical tool, the *Modeler's Assistant.* This provides a graphical interface for TIMESPEC. An important feature, that is made available by this interface, is the enumeration of all the input-to-output paths in the device. Thus a CAD tool is made available for system or chip designers/modelers for building accurate VHDL models where the timing is incorporated using the *imbedded timing* method.

Table of Contents

Chapter 1. Introduction 1

1.1 Modeling Methodologies 2

1.2 Problem Definition 7

1.3 Contributions 9

1.4 Summary of Chapters 10

Chapter 2. Mathematical Background 11

2.1 Mathematical Model 11

2.2 Linear Programming Problem 15

2.3 The Modified Duoplex Method 20

Chapter 3. Algorithm development 29

3.1 Mapping to a LPP 29

3.2 A Feasible Solution 34

3.3 A Practical Solution 38

3.4 Other Programming Issues 49

3.5 Summary 52

Chapter 4. Interface With the Modeler's Assistant 54

4.1 The Modeler's Assistant 54

4.2 Data Structures 59

4.3 Problem Definition 59

4.4 Mathematical Model 62

4.5 Successor Matrix Formulation 65

4.6 Path Enumeration 68

4.6 TIMESPEC and Modeler's Assistant Interaction 71

Chapter 5. Results 74

5.1 Error Detection and Correction Chip 74

5.2 Arithmetic Logic Unit 83

Chapter 6. Conclusion and Future Scope 93

6.1 Conclusion 93

6.2 Future Scope 95

Bibliography 97

Appendix A. Functional Block Diagrams 101

Appendix B. Users Manual 104

The Interface 104

The Main Menu 105

Core Constraint Formulation 105

Conflict Removal and First Solution 106
Curve Fitting and Second Solution 107
Input-to-output Path Delays 108

Vita 110

List of Illustrations

Figure 1.1.1 The Separated Timing Method 4

Figure 1.1.2 The Imbedded Timing Method 4

Figure 1.2.1 Process Model Graph 8

Figure 2.1.1 I/O Path in Terms of the Generic Names 13

Figure 2.2.1 Solution to a LPP 17

Figure 2.3.1 Initial Duoplex Matrix (a) 23

Figure 2.3.2 Duoplex Matrix after one exchange 23

Figure 2.3.3 Duoplex Matrix at the end of Phase One 28

Figure 2.3.4 Duoplex Matrix at the end of Phase Two 28

Figure 3 Flow-chart of the TIMESPEC Algorithm 30

Figure 3.1.1 Tri-state Buffered Register 32

Figure 3.1.2 PMG for the Tri-state Register 33

Figure 3.2.1 Duoplex Matrix 36

Figure 3.3.1 Typical Delays for Typical Primitives 43

Figure 3.4.1 Matrix Data Structure 50

Figure 4.1.1 Sample Process Model Graph 55

Figure 4.1.2 Text Window displaying the VHDL code 57

Figure 4.2.1 Linked List for Storing the PMG of figure 4.1.1 58

Figure 4.3.1 PMG used as an example for Path Enumeration 61

Figure 4.4.1 Directed Graph for the PMG in Figure 4.3.1 63

Figure 4.4.2 Successor Matrix Format 64
Figure 4.7.1 Interaction between Modeler's Assistant and TIMESPEC 73
Figure 5.1.1 Process Model Graph for the EDAC chip 75
Figure 5.1.2 EDAC Core Constraint Formulation 76
Figure 5.1.3 Timing Specifications for the EDAC chip 77
Figure 5.1.4 EDAC Soft Constraints 79
Figure 5.1.5 TIMESPEC solutions for the EDAC chip 79
Figure 5.1.6 EDAC I/O Path Delays 82
Figure 5.2.1 Process Model Graph for the ALU 84
Figure 5.2.2 Timing Specifications for the ALU chip 86
Figure 5.2.3 The ALU Core Constraints 87
Figure 5.2.4 ALU Soft Constraints 89
Figure 5.2.5 TIMESPEC solutions for the ALU chip 89
Figure 5.2.6 Path Delays for the ALU 92
Figure A.1 Block Diagram of the EDAC chip 102
Figure A.2 Gate Level Diagram of the ALU 103

Chapter 1

Introduction

With rapid improvement in technology the complexity of VLSI chips and digital systems is growing at an enormous rate. These rapid advances in technology are causing the chip designs to become obsolete in just a couple of years. Hence the design issue is not just to build a novel system using the current technology, but to build a system fast enough before any technological changes render the design inefficient. This urgency in the design process has led to the development of a large number of computer-aided design (CAD) tools. It is important that the chip designer takes full advantage of the computer aids available and concentrates more on the critical issue of obtaining an efficient design. Modeling and simulation of digital systems has become a vital tool in meeting this objective. It has established its importance in the design, synthesis, testing and verification process for digital systems, particularly VLSI chips.

The most important goal of chip simulation is to accurately model the behavior and the timing characteristic of the chip. The VHSIC hardware description language (VHDL) provides powerful constructs which allows accurate modeling of digital systems, right from the system level down to the switch level [1]. It also provides a very efficient way of modeling the timing characteristic in chips. The IEEE has recognized these capabilities and developed a standard for VHDL [2].

The prominent role played by VHDL in chip design, verification and documentation [1,3] has led to the development of various CAD tools to assist in obtaining accurate VHDL models of digital systems. The amount of time spent on producing these models is thus being reduced considerably. This objective of assisting the production of accurate models for VLSI chips was the primary motivation behind the development of the CAD tool, TIMESPEC. The various issues involved in the design and development of this CAD tool are presented in this work. Although not limited to VHDL modeling TIMESPEC aims at incorporating proper timing delays in chip models to obtain accurate descriptions of VLSI chips.

1.1 Modeling Methodologies:

The methodology of incorporating timing delays into models of digital systems has been classified into two major categories, viz. *separated timing* and *imbedded timing* [4,5].

The *separated timing* refers to the separation of the behavioral description of the chip from its timing description [6]. In this category, a chip description consists of two separate modules (figure 1.1.1). One module provides the behavioral description using only the *delta-delays*. The second module provides only the timing delays for the chip. In the timing module, a delay is associated with each output pin corresponding to the delay provided in the timing specification for a given input pattern. Hence this method incorporates the input-to-output delays at the chip level.

On the other hand, the *imbedded timing* refers to incorporating the timing delays at the primitive level (figure 1.1.2). In this method of representation, a delay value is associated with each primitive in the chip. Therefore the end-to-end delays are distributed among the various primitives and hence the separate timing module is not required.

We will now discuss the above mentioned methodologies for incorporating timing delays in behavioral descriptions of digital devices. There are two primary reasons why the *separated timing* method is preferred over the *imbedded timing* method in modeling VLSI chips:

1. The internal block timings are not available from the manufacturer's specification sheets. In addition to this, there are inaccuracies present in the available timing specifications. Thus the *imbedded timing* cannot be directly employed. With the *separated timing* method, the device timing can be directly and easily incorporated in the VHDL models.
2. Due to fast advancing technology, the chips have to be redesigned into the new technology. The overall chip behavior, however, need not change.

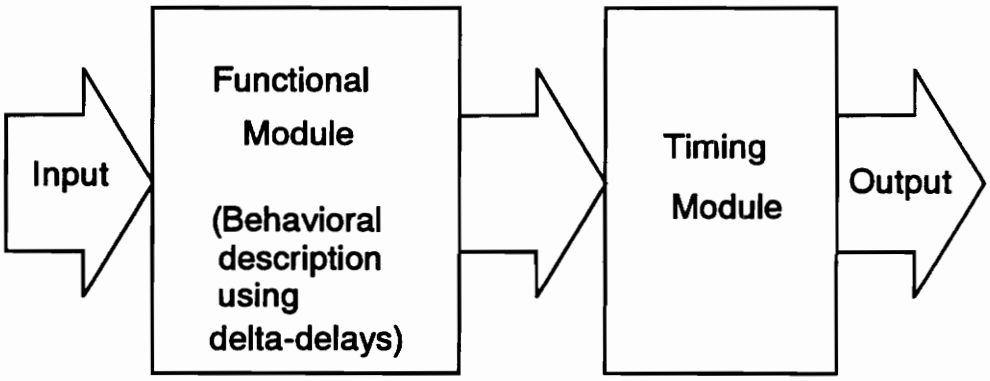


Figure 1.1.1 The Separated Timing Method

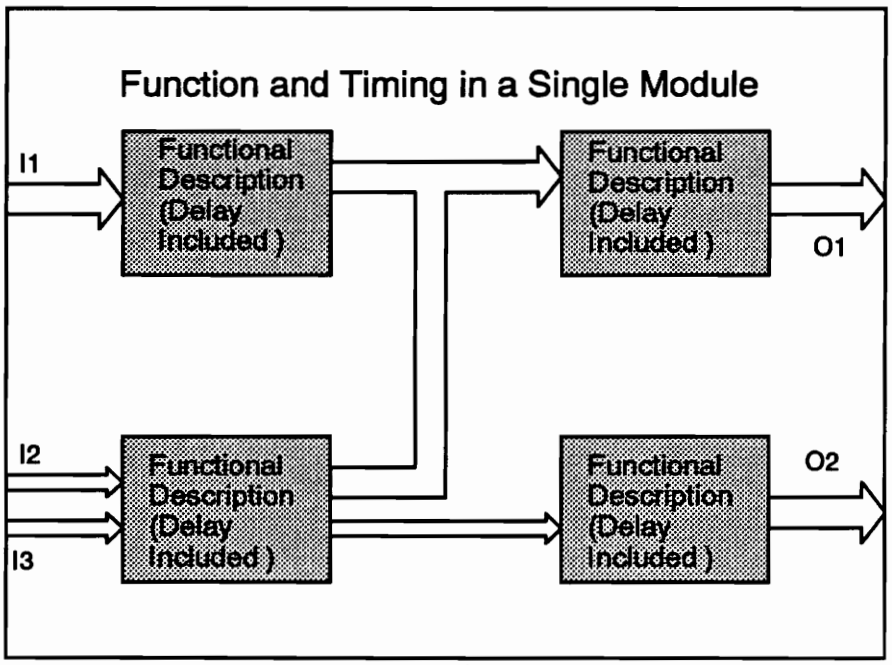


Figure 1.1.2 The Imbedded Timing Method

The models developed using *separated timing* do not need any alteration apart from the end-to-end delays, since there is no change in the overall chip specification. Only the timing module needs to be altered according to the change in the specifications. Whereas, the chip that is modeled using the *imbedded timing* needs a redistribution of delays, since the internal block delay timings may change.

There are some inherent limitations of using the *separated timing* method that can be overcome by using the *imbedded timing* method. These are now listed in the following discussion.

1. It is important to ensure that a digital system operates correctly at a desired speed. The modeler must verify that the propagation delay along any path from an input to an output is within a specified limit. This is especially important when the model is incorporated as part of a bigger system [7]. The models using *separated timing* can only be verified for those paths that are activated by the input patterns for which the path delays are defined by the timing specification sheets. For the models using the *imbedded timing*, the timing delays are incorporated at the primitive level, thus making it possible to test for other possible input patterns [8]. Thus the models using the *imbedded timing* methodology are a more complete and accurate description of the chip.
2. The modeling methodology of using the behavioral chip level modeling along with the *separated timing* results in VHDL models that provide an abstract view of the chip behavior. From a designers point of view, such a description is useful in the initial stages of the design.

However the final goal of the design process is to synthesize the chip it is necessary to go down in the abstraction hierarchy towards a Register Transfer Level (RTL) description of the chip [9]. Although research is in progress for synthesizing chips directly from a high level (algorithmic) description, most of the synthesis tools currently available can handle RTL descriptions rather than the algorithmic description. In addition, the designer has at his disposal, a number of primitives that are already designed and tested, which are required to be included as building blocks for the chip [10,11] (this is a partial top-down design, since some the primitives are already available). This is of particular interest because a number of commercial firms have shown an increasing interest in synthesizing circuits from behavioral descriptions of chips [12].

3. As the design progresses, a better insight is required into the timing problems that can arise in the chip, such as race conditions and other timing hazards [13,14]. The models using the *imbedded timing* represent a real system very closely and hence provide a better understanding of the chip. For the *imbedded timing* method, once the timing delays associated with each primitive are available, the model can be simulated to detect the timing hazards internal to the chip or other tools such as the timing analyzer/verifier [15,16] can be used. (eg. although two signals appear at the input at proper time, the delays through the internal primitives are such that they appear at the input of an internal latch in improper sequence). The designer can thus be made aware of the problem areas at an early stage. Therefore modifications to the design are relatively simpler than if the hazards are detected late in the design process. This is not possible when the *separated timing* method is incorporated in VHDL models of VLSI chips, which is more suitable for chips that exist already.

1.2 Problem Definition:

From the above discussion it is observed that the *imbedded timing* provides a much better modeling methodology than the *separated timing* method. In the *imbedded timing* methodology, the VHDL description of the chip consists of *processes* that correspond to the various circuit primitives. These processes are connected together by *signals* that model the wire connections between the primitive blocks. This can be represented graphically, as shown in figure 1.2.1 , by a *Process Model Graph* (PMG) [1,8,17].

To use the *imbedded timing* for VHDL models, it is necessary to associate timing delays to the various processes. *Generics* are used to represent the time delays associated with each process. For simulating any chip, it is necessary to provide the actual values for these generics. *This is the primary obstacle facing the use of the imbedded timing in VHDL models.* This problem arises because manufacturer's data sheets provide only the total input-to-output delay values and not the individual delays for the primitive block generic values. Thus, the problem is that of budgeting or distributing the input-to-output delay information among the various processes (generics) forming the PMG. This is henceforth referred to as the *Timing Distribution Problem.*

This problem is further aggravated by the fact that the data available is found to be inconsistent on many occasions. Also, due to the sheer number of the timing constraints provided it is not possible to determine, readily, the presence of the discrepancies. Since, the detection of the conflicts is itself difficult, the correction is an even more difficult problem.

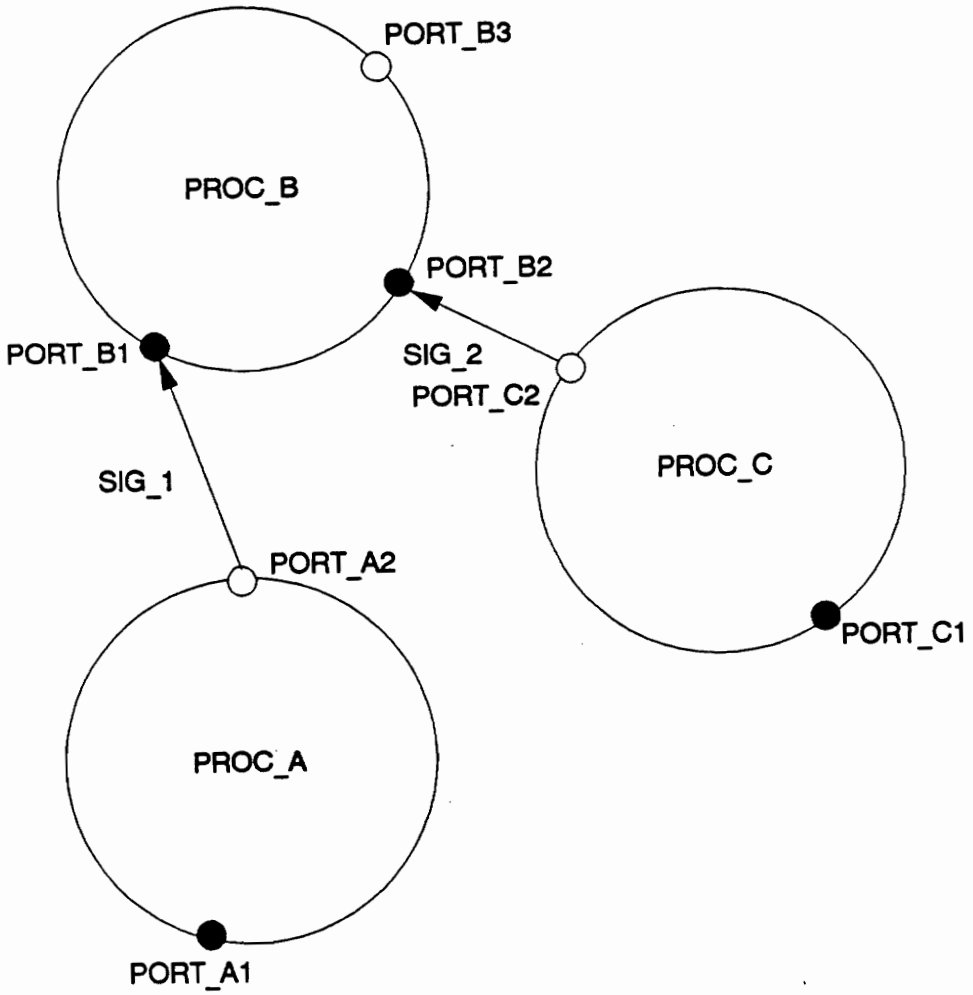


Figure 1.2.1 Process Model Graph

1.3 Contributions:

The primary objective of developing the software tool TIMESPEC was to solve the Timing Distribution Problem in order to facilitate the use of *the imbedded timing.* The various contributions of this work are now enumerated:

1. As mentioned above, the solution to the Timing Distribution Problem is obtained.
2. In addition, the timing inaccuracies that are present in the manufacturer's data sheets or in the interpreted specifications are corrected and thus the models obtained are devoid of these discrepancies. (In contrast, the models using the *separate timing* method carry forward the inaccuracies present in the timing specifications. It should be noted that if these inaccuracies are propagated in the VHDL descriptions of the chips, substantial amount of time and effort can be wasted in the modeling and simulation of digital IC's).
3. The removal of the timing inaccuracies also solves the problem facing the timing analyzer/verification tools which require the timing specifications to be accurate in order to function correctly [18].
4. When the internal block delay timings change, due to a change in technology, TIMESPEC can be invoked to easily obtain the new block delays. Since the VHDL models make use of *generics* to specify the timing in the models, the only change that is required is the change in the values assigned to these generics.
5. TIMESPEC is interfaced with an X-windows based graphical tool, the Modeler's Assistant [8,17]. Thus it was possible to enumerate all the paths in the PMG description of the chip. This capability is of immense importance since it is possible to verify the propagation delays along all the paths in the chip.

1.4 Summary of Chapters:

The development of the CAD tool TIMESPEC is described in detail in chapter 3. The mathematical model development and the mathematical background is discussed in the next chapter. Chapter 4 describes the interface of TIMESPEC with the Modeler's Assistant and provides a detailed description of the path enumeration algorithm utilized in obtaining the propagation delays along all the paths in the chip. The results obtained by applying TIMESPEC to two chips, the Error Detection and Correction Chip (EDAC) and the Arithmetic Logic Unit (ALU) are presented in Chapter 5. The conclusion and the scope for future developments are discussed in chapter 6.

Chapter 2.

Mathematical Background

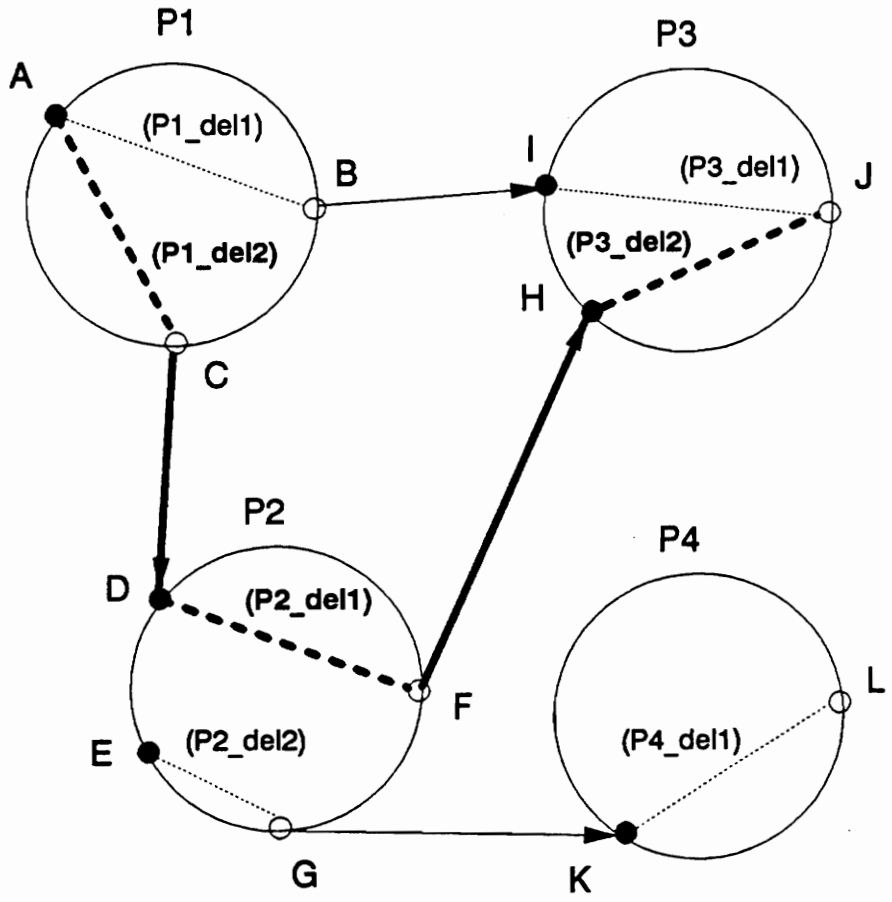
The first step towards obtaining a solution to the Timing Distribution Problem was to obtain mathematical relationships that model this problem. This is the topic of discussion in the next section. The rest of the chapter will deal with the mathematical background required to understand the design issues in developing the software tool TIMESPEC, which solves the Timing Distribution Problem.

2.1 Mathematical Model :

Right from the first page of this work, the term *model* has been used repeatedly. A model is basically a structure built to exhibit the characteristics and features of some other object under consideration. There are a large number of factors that motivate the construction of these models [19]. As mentioned earlier, the VHDL models are used for simulating the digital systems being modeled, for the design and synthesis of VLSI chips and digital systems and also for their documentation. Just the exercise of building

models, reveals more insight into the object being modeled. Abstract models are usually mathematical, and hence they enable the analysis of the object being modeled and provide paths to follow, which may otherwise be overlooked, to reach solutions to the problems. The essential feature of a mathematical model is that it involves a set of mathematical relationships such as equations, inequalities, logical dependencies, etc.

Focusing again on the Timing Distribution Problem, the aim is to obtain a mathematical model for mapping the input-to-output delay values onto the various process-generics in the block diagram description of the chip (PMG). The input-to-output delays are specified for a fixed input pattern and hence a path is defined from an input pin to an output pin, that traverses the encountered processes. Thus, this available timing information can be viewed as a set of paths, where each path provides a sum total of the delays associated with the primitive blocks that are traversed by the path. *Therefore, the Timing Distribution Problem to be solved is to allot delay values to these primitive elements or process-generics such that the sum totals for each path are realized.* This can be represented as a set of inequalities/equalities, where, each process-generic is a variable and the specified input-to-output path identifies the variables involved in a particular inequality. The delay value in the specification sheet provides the upper limit, lower limit or the exact value of the sum of all the variables involved in the path under consideration. Such a path is defined by the various generics as illustrated in figure 2.1.1. In this figure the dashed lines indicate the internal paths in the functional blocks (processes) and the corresponding delays (generics) are provided in parentheses. The solution of these set of inequalities/equalities will provide the necessary delays to be associated with the generics (variables) in the processes (path).



$$Path = P1\_del2 + P2\_del1 + P3\_del2$$

Figure 2.1.1 I/O Path in Terms of the Generic Names

It is observed, as mentioned earlier, that there are some conflicting constraints, ie., the inequalities/equalities are inconsistent. In case of conflicting constraints, it is obvious, that the solution is infeasible, ie., it is not possible to assign values to the variables such that all the inequalities/equalities are satisfied. Therefore, the available constraints become objectives that may or may not be satisfied. The conflicting constraints need to be modified, in order to obtain a feasible solution.

The Timing Distribution Problem defined above maps very well to a *linear programming problem* (LPP) [20,21]. This is a class of mathematical programming models that involves optimization of some quantity. The optimization involves maximizing or minimizing of an objective under a set of constraints. Since the constraints on the variables are linear this model is classified as a *linear programming* (LP) model. It is a well defined mathematical problem with a number of efficient algorithms already developed to solve it. The algorithm used here is a modification of the *Duoplex Algorithm* developed by Kunzi, Tzschach and Zehnder [22,23].

The algorithm for solving the given set of inequalities will be explained in detail in the later chapters. At this point, it is of interest to know that the algorithm provides a means of detecting whether or not a solution exists for the given set of constraints. Which implies, that the algorithm provides a way of detecting the presence of conflicts in the constraints. It also provides information as to the location of the inconsistencies. In addition and most importantly, information can be extracted from the output of this algorithm which can be used to correct the inconsistencies. Thus the use of the LP model for the Timing Distribution Problem becomes practical. The major difficulty observed is the absence of any optimizing function. This does not mean that this particular approach be dismissed. A way to get around this problem is explained in detail in chapter 3. The

next part of this chapter describes the LPP and gives a brief outline of the modified Duoplex method employed to solve the LPP.

2.2 Linear Programming Problem (LPP)

In a LPP [20,21], there is a single linear expression called the *objective function* that is to be optimized. The optimization can be a maximization or minimization operation, but the two are inter-convertible by simply negating the expression, hence, only the maximization operation is considered. The optimization of the objective function has to be performed subject to a given set of constraints. The LPP model can be represented as follows:

For **n** independant variables $\mathbf{x_i}$, i=1,2.....n, the objective function **z**, to be maximized is given as

$$z = a_{o1}x_1 + a_{o2}x_2 + ...+ a_{on}x_n$$

subject to $\mathbf{m = m_1+m_2+m_3}$ constraints, $\mathbf{m_1}$ of them of the form

$$a_{i1}x_1 + a_{i2}x_2 + .. + a_{in}x_n \leq b_i \qquad (b_i \geq 0) \qquad i=1,...,m_1$$

$\mathbf{m_2}$ of them of the form (2.2.1)

$$a_{j1}x_1 + a_{j2}x_2 + .. + a_{jn}x_n \geq b_j \qquad (b_j \geq 0) \qquad j=m_1+1,..,m_1+m_2$$

$\mathbf{m_3}$ of them of the form

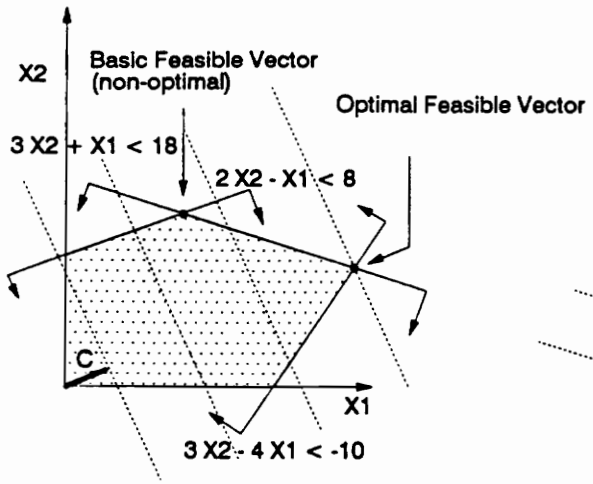
$$a_{k1}x_1 + a_{k2}x_2 + .. + a_{kn}x_n = b_k \qquad (b_k \geq 0) \qquad k=m_1+m_2,..,m_1+m_2+m_3$$

The primary restriction of nonnegativity holds for all x_i (i=1 to n). The set of values $x_1, x_2 \ldots x_n$ that satisfy the above inequalities and equalities (constraints) is called a *feasible vector*, and the set of all such feasible vectors is termed as the *feasible region* for the LPP.

To provide a good insight into the LPP, a geometric approach is adopted. A simple case using 2 independant variables x_1 and x_2 is depicted in figure 2.2.1. In this case the constrains on the variables are represented as lines in the 2-dimensional (2-D) plane. Only the first quadrant is considered because x_1 and x_2 are bounded by the nonnegativity constraint. The additional constraints, (three in figure 2.2.1a, figure 2.2.1b and figure 2.2.1d and two in figure 2.2.1c) then define the feasible region. The feasible region is indicated by the shaded area. The direction in which the objective function increases is indicated at the vertex by a vector pointing in the appropriate direction. The *optimal feasible vector* is obtained by choosing vectors from the feasible region, for which the objective function increases and finally reaches its maximum value. This process is indicated by the contour lines shown in the figure 2.2.1. The contour line (dotted line) is perpendicular to the objective function vector and is moved till it just touches an extreme point or an extreme edge.

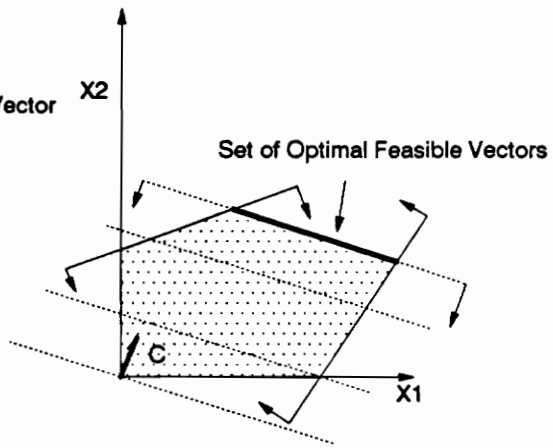
Four different cases are possible for a solution to a given LPP.

1. The optimal feasible solution is unique (figure 2.2.1a). As we move along the direction of the increasing objective function, a point is reached for which a contour line is tangential to the binding constraints. Thus, the optimal feasible vector is unique and appears at an extreme point.



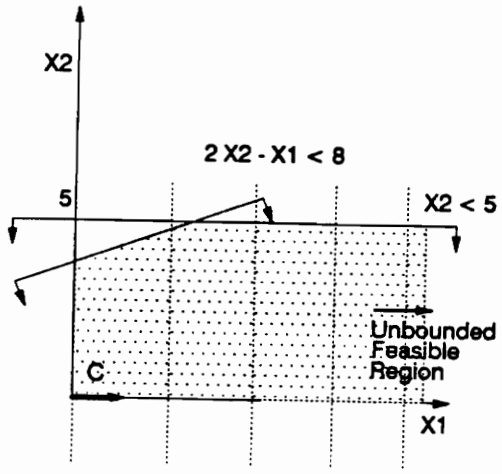
Unique Optimal Feasible Solution

(a)



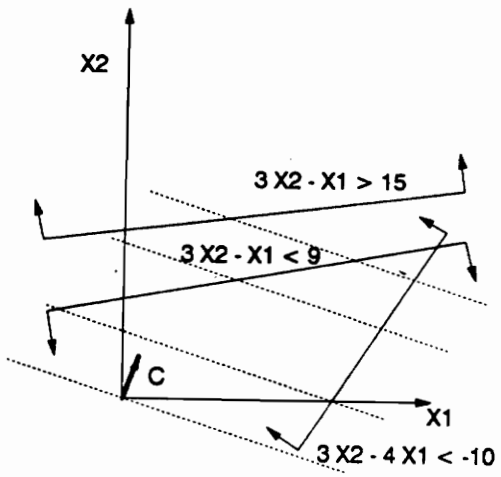
Non-Unique Optimal Feasible Solution

(b)



Unbounded Optimal Solution

(c)



No Feasible Solution

(d)

Figure 2.2.1 Solution to a LPP

2. This case is very similar to the first one, the only difference being, the optimal feasible vector is not unique. Instead it is a line segment which is parallel to the contour lines representing the increasing objective function. This is shown in figure 2.2.1b.
3. The feasible region is unbounded in the direction of the increasing objective function. The objective function can be increased arbitrarily by choosing feasible vectors lying on the boundary wall (constraint) that is parallel to the direction of the objective function, as shown in figure 2.2.1c.
4. There is no feasible solution to the given LPP. This implies that the binding constraints are conflicting and hence there is no feasible region (figure 2.2.1d).

Extending the above 2-dimensional example to a general n-dimensional problem space, the linear constraints binding the feasible region are now n-dimensional hyperplanes. The hyperplanes divide the n-dimensional space into three regions, the two half planes on either side representing the "$\leq$" and "$\geq$" constraints and the hyperplane itself representing the "=" constraint. If the equality constraint is true, the feasible region is reduced such that it is now defined by the hyperplane of a smaller dimension. If the constraint is an inequality then the feasible region is divided into allowed and non-allowed half spaces. In this way the binding feasible region is defined for a given LPP.

For a given LPP, if a feasible region exists, ie. the constraints are non-conflicting, then the optimal feasible solution cannot lie in the interior region away from the boundaries of the feasible region. In addition, if an optimal feasible solution exists then it corresponds to an *extreme point* of the feasible region. An analytical proof for this result is provided in reference [20]. A simple geometrical approach can be used to illustrate this

concept. The objective function is linear and can be increased by choosing points that are in the direction of the objective function till a bounding wall (hyperplane) is reached. The boundary of any geometrical region has one dimension less than that of the interior. For example in a 3-D case, the bounding plane **z**=0 corresponds to the 2-D space in which each point in this plane can be represented by the co-ordinates (**x**,**y**). Reaching a bounding wall fixes one co-ordinate, and hence a hyperplane less by one dimension is obtained. As we move in the direction of the *objective-function-gradient* projected into the boundary wall (which is a hyperplane with 1-D less than the previous hyperplane), another bounding hyperplane is reached which in turn has 1-D less than the current hyperplane. This process is continued till an extreme point is reached. Since, this point has all its **n**-co-ordinates defined, it is a feasible vector. Therefore, due to the linearity property of the objective function, the feasible solution cannot lie in the interior region since the objective function can always be increased till an extreme point is reached. Thus it is clear, from the procedure by which this feasible vector is obtained, that this feasible vector is the optimal feasible vector.

Another important observation is that only the extreme points of the feasible region are required to be considered in order to locate the optimal feasible vector. These extreme points are called the *basic feasible vectors*. Thus, the problem of finding an optimal feasible solution is equivalent to choosing from a collection of basic feasible vectors, an appropriate vector, such that the objective function is maximized. The modified Duoplex method gives a systematic and definite procedure of moving from one basic feasible vector to another till the optimum objective function is reached. The various steps taken to achieve the optimal basic feasible vector using the modified Duoplex method are now explained.

2.3 The Modified Duoplex Method:

The algorithm is explained with reference to an example for better understanding.

Consider the following LPP,

maximize $z = x_3 + x_4$

subject to

$$\begin{aligned} x_1 + 2x_2 + x_3 + 4x_4 &\leq 40 \\ 2x_1 - x_2 + x_4 &\geq 20 \\ x_1 + x_2 + x_3 + x_4 &= 16 \end{aligned} \tag{2.3.1}$$

where, m=3, n=4, $m_1=m_2=m_3=1$.

This particular example is chosen because it is representative of the different types of constraints defined in the general LP formulation (≤,≥,=). To solve the LPP using this algorithm, the LPP must be first transformed into a format called the *Restricted Normal Form* [23]. The Restricted Normal Form puts a restriction on the constraints such that they are equality constraints only and that there must be at least one variable, in each constraint, that has a positive coefficient and is unique to that constraint only. This transformation from a general LPP formulation to the Restricted Normal Form is achieved easily as shown below. The procedure mentioned in the following paragraphs is applied to all the constraints in order to convert any given LPP into Restricted Normal Form.

For the "≤" constraints, new variables called the *slack variables* are introduced, to convert the inequality into an equality. Thus, in equation (2.3.1) we get,

$$x_1 + 2x_2 + x_3 + 4x_4 + y_1 = 40 \qquad (2.3.2)$$

The addition of a slack variable y_1 automatically converts the "≤" constraint into Restricted Normal Form. For the "=" and "≥" constraints, new variables called the *auxiliary variables* (z_2 and z_3) are introduced. Thus the modified constraints in Restricted Normal Form are:

$$\begin{aligned} x_1 + 2x_2 + x_3 + 4x_4 + y_1 &= 40 \\ 2x_1 - x_2 + x_4 + z_2 &= 20 \\ x_1 + x_2 + x_3 + x_4 + z_3 &= 16 \end{aligned} \qquad (2.3.3)$$

The addition of the auxiliary variables modifies the original LPP. Thus, the Restricted Normal Form representation is equivalent to the original LPP only if all the auxiliary variables corresponding to the "=" constraint are zero and those for the "≥" constraint are non-positive. It is also observed that when the slack or the auxiliary variables are zero the corresponding constraints are satisfied as equalities. The auxiliary variables play a very important role in the development of the software tool TIMESPEC. This aspect will be dealt with more rigorously in the next chapter. It is observed in equations (2.3.4) that the slack and the auxiliary variables can be expressed in terms of the remaining variables.

$$\begin{aligned} y_1 &= 40 - x_1 - 2x_2 - x_3 - 4x_4 \\ z_2 &= 20 - 2x_1 - x_2 - x_4 \\ z_3 &= 16 - x_1 - x_2 - x_3 - x_4 \end{aligned} \qquad (2.3.4)$$

In such a representation the variables on the left hand side are called the *left hand variables* and the variables on the right are called the *right hand variables*. By definition the right hand variables are always assigned zero values. Thus, for the new formulation

we always have a starting basic feasible solution ($x_1 = x_2 = x_3 = x_4 = 0$ and $y_1 = 40$, $z_2 = 20$, $z_3 = 16$).

It is required to make the auxiliary variables the right hand variables, so that they will have a value of zero and the original problem will be unaltered. Hence the problem is solved in **two phases**. The **first phase** aims at solving the modified constraints such that the auxiliary variables are minimized ie. they are made zero. The objective function for phase one is to minimize the sum of the auxiliary variables. This is called the *auxiliary objective function* (**z'**).

$$z' = -(z_2 + z_3) \tag{2.3.5}$$

Once the auxiliary variables (z_2 and z_3) are made zero, the original objective is considered and phase two begins.

At the start of **phase two**, the original problem is retrieved ie. the original objective function is considered, with the added advantage of having a starting basic feasible solution. Thus a penalty is paid of going through two phases in order to insure a starting feasible solution at all times [20,21,23].

In each of the two phases, the steps taken in solving the constraints for the corresponding objectives are now considered. To discuss this, it is convinient to represent the LPP in a *tableau format* or a *matrix format* (figure 2.3.1). This matrix has (**m+2**) rows and (**n+1**) columns. The first element in the first row corresponds to the value of the given objective function in terms of the right hand variables. Thus the value of the objective function is always zero at the start of phase one.

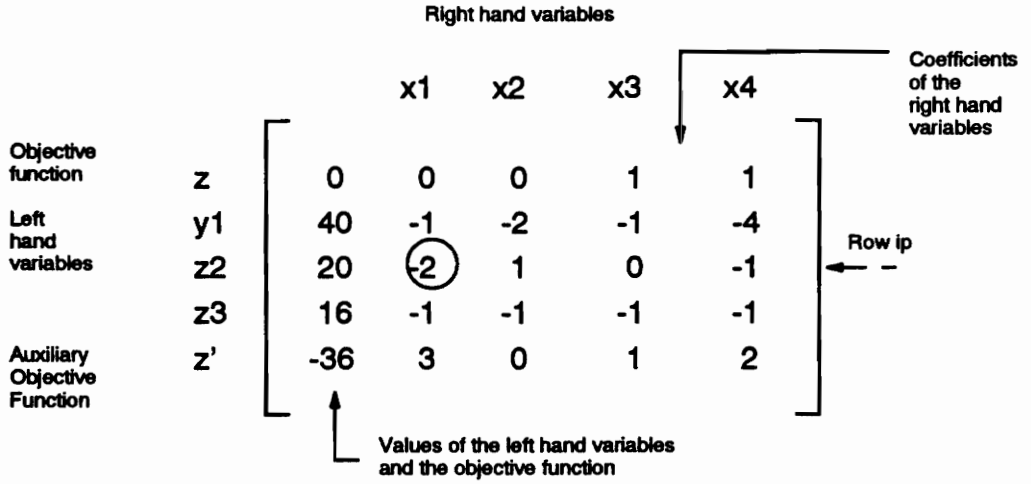


Figure 2.3.1 Initial Duoplex Matrix (a)

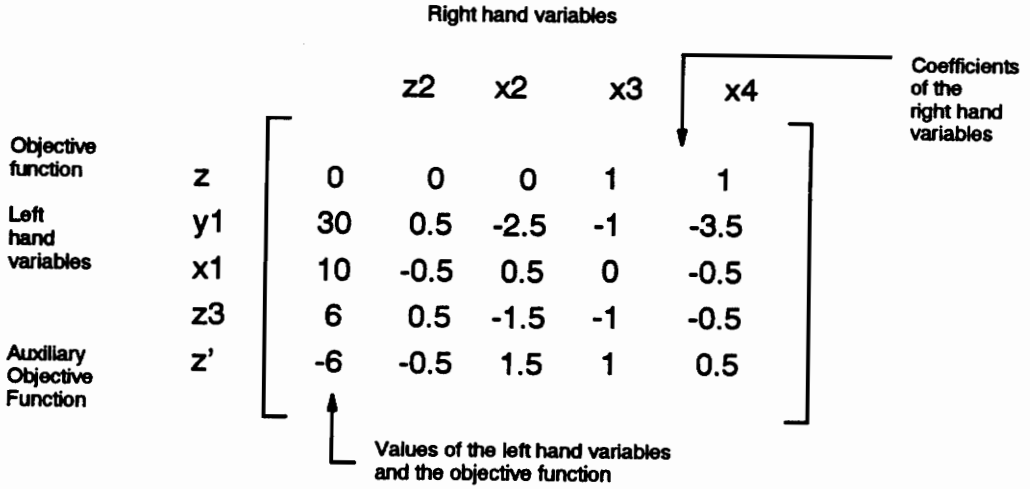


Figure 2.3.2 Duoplex Matrix after one exchange

The remaining entries in the first row correspond to the coefficients of the right hand variables used in expressing the objective function.

$$z = x_3 + x_4 \tag{2.3.6}$$

Similarly, the first element in the **(m+2)** row corresponds to the value of the auxiliary objective function expressed in terms of the right hand variables.

$$z' = -36 + 3x_1 + 2x_2 + x_3 + 2x_4 \tag{2.3.7}$$

Since x_1, x_2, x_3 and x_4 are right hand variables (zero value), $z' = -36$. The remaining entries in the **z'** row correspond to the coefficients of the auxiliary objective function (equation 2.3.7). It is evident, by comparing equations (2.3.3) with the matrix in figure 2.3.1, that the remaining rows correspond to the coefficients in the right hand side of the eqnations. Since the right hand variables are zero, the values of the left hand variables are given in the first column of the matrix. Thus initially $y_1 = 40$, $z_2 = 20$ and $z_3 = 16$.

In both the phases, the corresponding objective functions are to be maximized. The first phase is considered here to exemplify the steps involved in maximizing the objective function. As seen from equations (2.3.7), **z'** will increase if the right hand variable corresponding to the maximum positive coefficient is increased from its current value of zero by making it a left hand variable. Thus, the column corresponding to x_1, which has a **3** in its **(m+2)** row, is of interest. The right hand variable so considered can be increased, causing an increase in the objective function, till one of the left hand variable is driven negative, which is a violation of the non-negativity restriction on all the variables. The row corresponding to the left hand variable which causes this violation first is the restricting row. This row is determined by examining the following ratio,

$$\left| \frac{a[i][1]}{a[i][j]} \right| \qquad (2.3.8)$$

where **a** is the given matrix,

i = 2,3.....(m+1)

j is the column corresponding to the right hand variable selected.

The row **ip** corresponding to the smallest magnitude of this ratio, in which **a[i][j]** is negative, corresponds to the row which is most restrictive. The element corresponding to the intersection of the **j**th column and row **ip** is called the *pivot element*. The pivot element for the example considered is **a[4][2] = -2**.

If there are no negative entries in column **j**, then the objective function can be increased without bound. *Hence, if we are in phase one then we have an unbounded solution for the modified constraints and consequently no feasible solution exists for the original set of constraints*. Thus, this condition in phase one detects the infeasibility condition. *If we are in phase two, then we have an unbounded solution.* These conditions are flagged immediately and the algorithm terminates.

After identifying the pivot element, the next step is to do the increase in the current right hand variable by converting it to a left hand variable. This is achieved by representing all the new left hand variables in terms of the new right hand variables. Consequently the left hand variable corresponding to the row **ip** is converted to a right hand variable (value zero). Thus the equations (2.3.4) are transformed as follows :

$$\begin{aligned} y_1 &= 30 + 0.5\, z_2 - 2.5\, x_2 - x_3 - 3.5\, x_4 \\ x_1 &= 10 - 0.5\, z_2 + 0.5\, x_2 - 0.5\, x_4 \\ z_3 &= 6 + 0.5\, z_2 - 1.5\, x_2 + x_3 - 0.5\, x_4 \end{aligned} \qquad (2.3.9)$$

The matrix **a** in figure 2.3.2 , depicts all the changes that take place. It is observed that the value of **z'** has reduced by taking the above mentioned steps and one auxiliary variable (z_2) has indeed become zero. If in phase one, all the above steps are repeated till all the auxiliary variables become zero, then phase two is started. *If the first phase is not able to make all the auxiliary variables zero then this indicates that the original constraints are not consistent to begin with, and hence indicates an infeasibility and the algorithm terminates.* Once phase two is reached the above mentioned steps are repeated till it is not possible to increase the onjective function any further. This condition is detected when all the entries in the **z** row are non-positive. The various steps can be easily performed once the LPP is available in the matrix format. The various operations performed on the matrix are summerized as follows:

- Locate the pivot element and save it.
- Save the entire pivot column.
- Replace each row, except the pivot row, by the linear combination of itself and the pivot row which makes its pivot-column entry zero.
- Divide the pivot row by the negative of the pivot.
- Replace the pivot element by the rciprocal of its saved value.
- Replace the rest of the pivot column by its saved values divided by the saved pivot element.
- The terminating rules are as explained above.

The Duoplex tableau at the end of phase one and at the end of phase two are indicated in the figure 2.3.3 and figure 2.3.4 respectively. The solution set is obtained from column one of the final Duoplex matrix:

$$
\begin{aligned}
x_1 &= 6 \\
x_2 &= 0 \\
x_3 &= 2 \\
x_4 &= 8
\end{aligned}
\tag{2.3.10}
$$

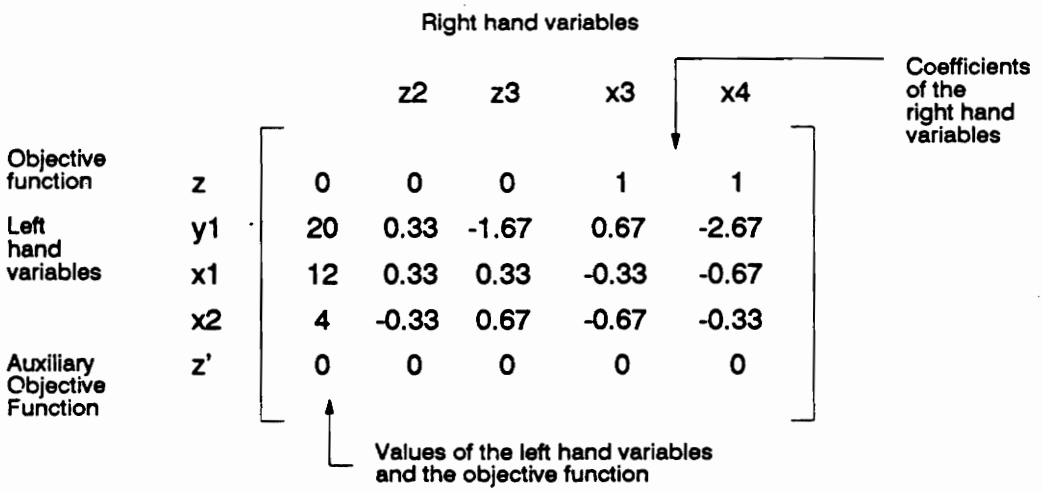


Figure 2.3.3 Duoplex Matrix at the end of Phase One

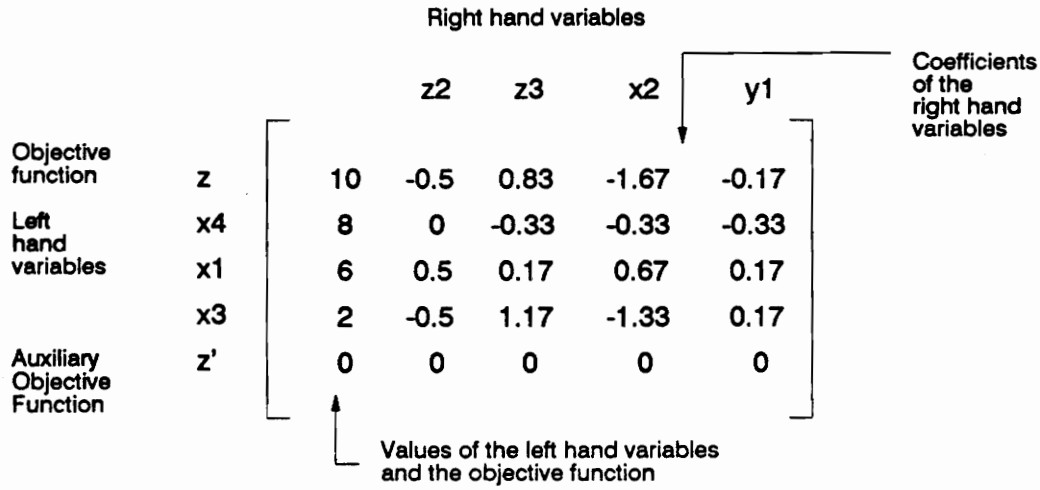


Figure 2.3.4 Duoplex Matrix at the end of Phase Two

Chapter 3.

Algorithm Development

The timing distribution problem defined in chapter 2, for the timing constraints available from the manufacturer's data sheets, was modeled as a Linear Programming Problem (LPP). Chapter 2 also dealt with briefly describing the LPP and the modified Duoplex method employed to solve the LPP. Now, a sufficient background has been established to explain the design and development of the software tool, TIMESPEC. The flowchart of the TIMESPEC algorithm is indicated in figure 3, which will be referenced as we proceed in the development of the algorithm.

3.1 Mapping to a LPP:

It was noted in section 2.1 that the only problem in mapping a Timing Distribution Problem onto a LPP was the absence of an objective function. This problem

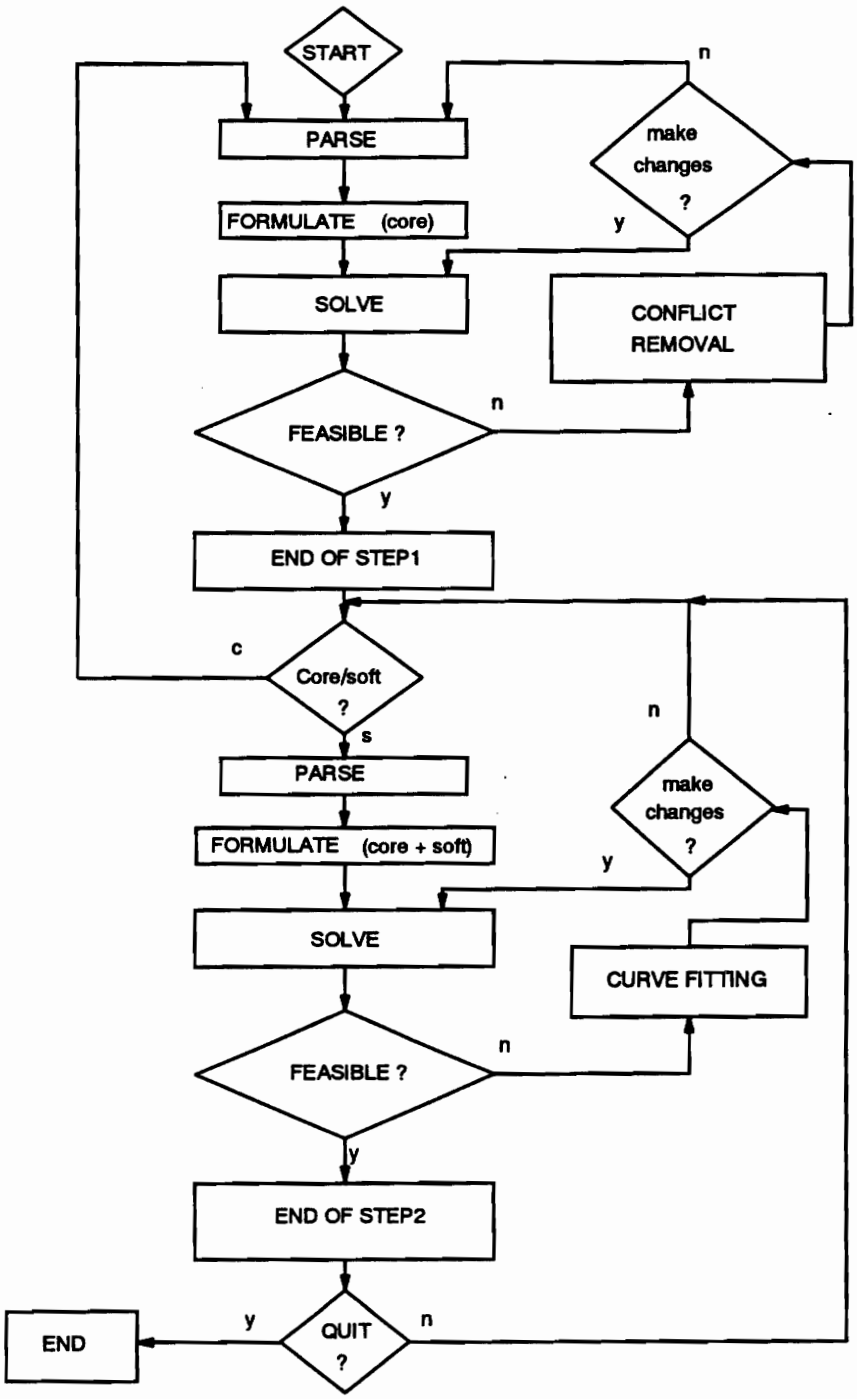
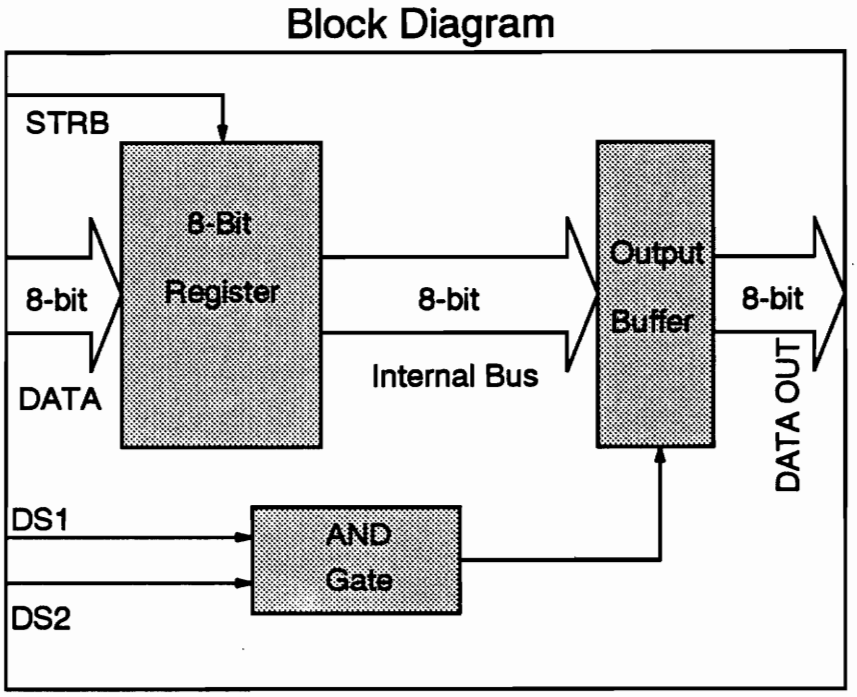


Figure 3 Flow-chart for the TIMESPEC Algorithm

can be easily overcome by noting that the existence or the non-existence of a feasible solution depends solely on the constraints and not on the objective function [19]. The objective function only identifies the best solution, for that objective, from a set of all possible solutions, if at all they exist. Thus a dummy objective function can be used which does not in any way conflict with the original Timing Distribution Problem. In addition, if any of the generic values are to be maximized/minimized, then they can be included as part of the objective function as well. In other words, any objective function can be specified and in case of no specific objectives to be met a default objective function "dummy" is used. Finding a solution which satisfies certain constraints may be by no means straightforward. This is especially true when the number of constraints is large and there is a possibility of presence of inconsistencies. Thus, converting a given Timing Distribution Problem into a LPP, as indicated above, at least enables one to find a feasible solution if at all it exists. Additional information about the conflicts is also gained in case of a non-feasibility. This is explained in the following sections. The mapping of a given Timing Distribution Problem onto a LPP is seen in the simple example of a register model. The VHDL description for this chip are described in detail in reference [1].

The block diagram of the 8-bit register and its corresponding timing specifications are shown in figure 3.1.1. The process model graph describing the register is indicated in figure 3.1.2. In this chip, the data is loaded into the register on the rising edge of the STRB input. If the output buffers are enabled then data will become available at the output of the register after a delay of Tsd ns. If the enable signals DS1 and DS2 change then this change is reflected at the output after a delay Ted. In this model the



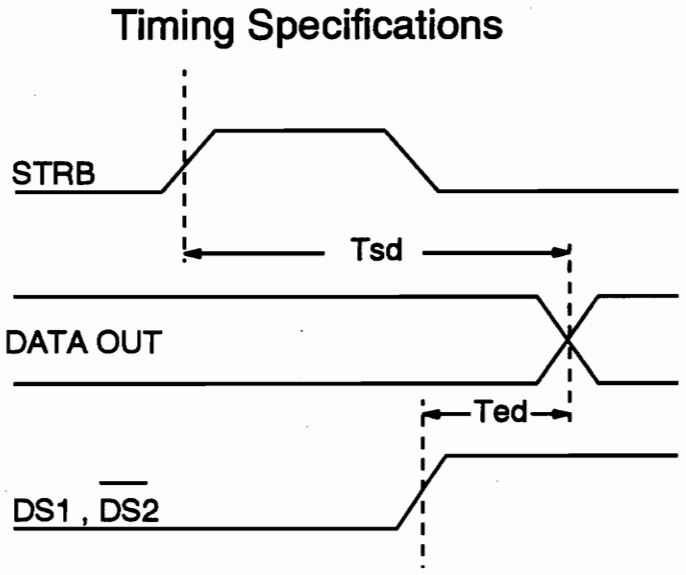


Figure 3.1.1 Tri-state Buffered Register

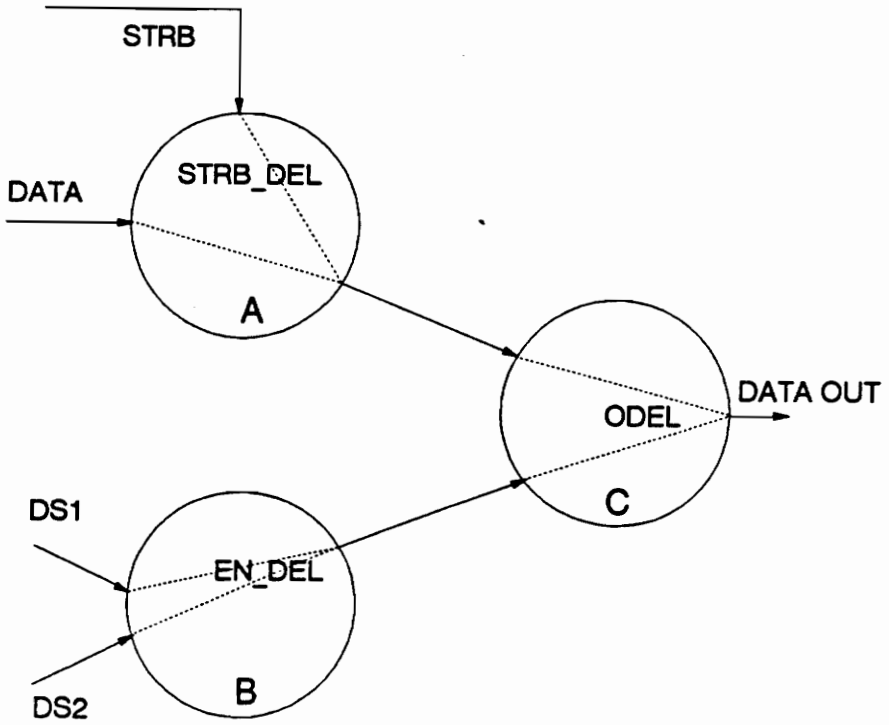


Figure 3.1.2 PMG for the Tri-state Register

generic STRB_DEL is associated with process A, EN_DEL is associated with process B and ODEL is associated with process C. This model assumes the following restrictions:

$$Tsd = STRB\_DEL + ODEL$$

$$Ted = EN\_DEL + ODEL \qquad (3.1.1)$$

Thus the Timing Distribution Problem deals with finding the values of STRB_DEL, EN_DEL and ODEL given the values of Tsd and Ted. *Let us assume that the following constraints are specified on Tsd and Ted*:

$$Tsd \geq 12$$

$$Ted \leq 10 \qquad (3.1.2)$$

To convert the distribution problem to a LPP, a dummy variable is introduced, which is itself used as the objective function (**z**)

$$z = dummy$$

The constraints under which this objective function is to be maximized are:

$$dummy = 1$$

$$STRB\_DEL + ODEL \geq 12 \qquad (3.1.3)$$

$$EN\_DEL + ODEL \leq 10$$

3.2 A Feasible Solution:

It has been mentioned time and again that the available set of constraints may be inconsistent. Thus the nature of the Timing Distribution Problem is completely unpredictable or unknown. Therefore an ill-formulated LPP can be encountered which can lead to an infeasible solution.

In order to explain the development of the algorithm, a contradiction is introduced in the register example considered in the previous section. The new set of constraints are stated as follows.

$$\begin{aligned} \text{dummy} &= 1 \\ \text{STRB\_DEL} + \text{ODEL} &\geq 12 \\ \text{EN\_DEL} + \text{ODEL} &\leq 10 \\ \text{EN\_DEL} - \text{STRB\_DEL} &= 2 \end{aligned} \tag{3.1.4}$$

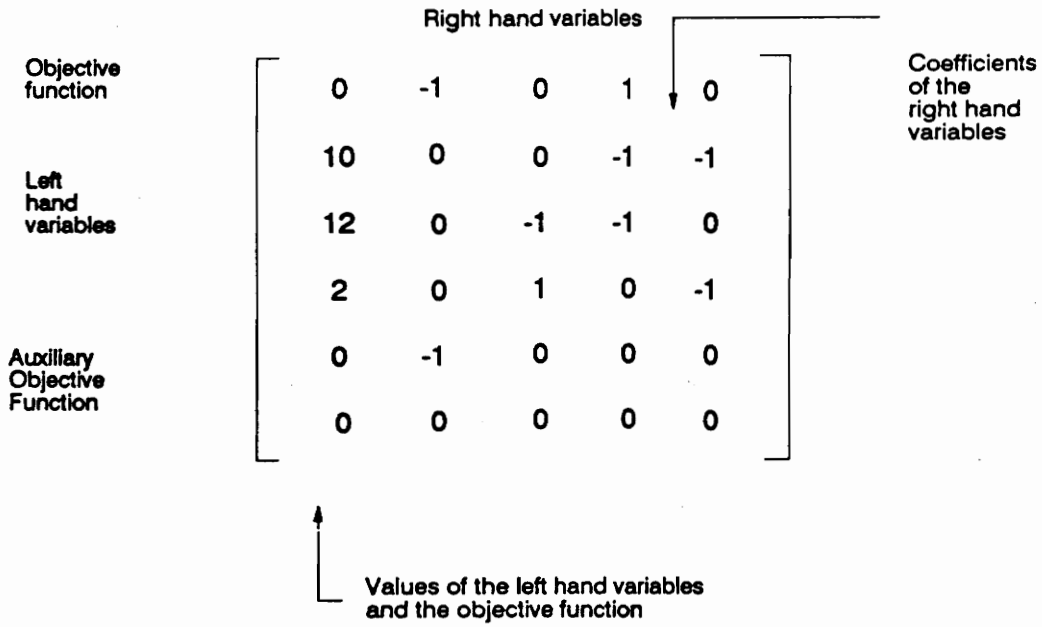
It is very easy to realize that the given set of constraints is contradictory. But in a larger set of constraints it is not possible to detect such inconsistencies.

To solve these constraints using the modified Duoplex method, they must first be converted into the Restricted Normal Form by the introduction of the slack and the auxiliary variables. Thus, the transformed constrains are:

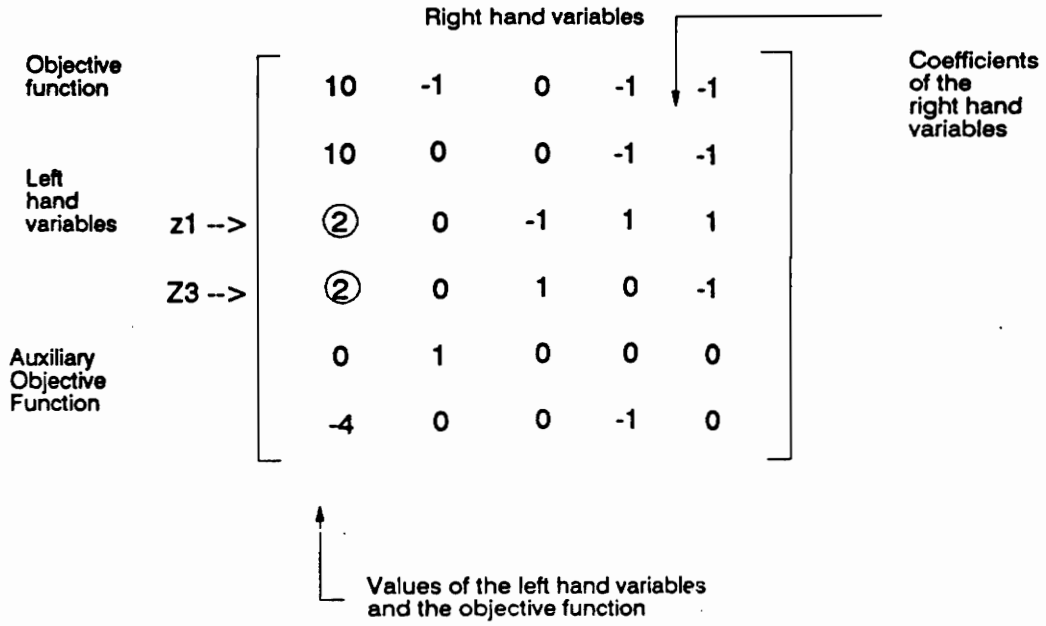
$$\begin{aligned} \text{dummy} + z_1 &= 1 \\ \text{STRB\_DEL} + \text{ODEL} + z_2 &= 12 \\ \text{EN\_DEL} + \text{ODEL} + y_1 &= 10 \\ \text{EN\_DEL} - \text{STRB\_DEL} + z_3 &= 2 \end{aligned} \tag{3.1.5}$$

This formulation is represented in Duoplex tableau format (matrix **a**) in figure 3.2.1a. The presence of the inconsistencies in the above formulation is detected by the Duoplex algorithm and it terminates with the matrix output given in figure 3.2.1b.

The matrix output given in figure 3.2.1b can be utilized to identify the location of the inconsistencies and to correct the given set of constraints. It is observed from the equations (3.1.5) that this formulation in the Restricted Normal Form is equivalent to the formulation in equations (3.1.4) only if all the auxiliary variables ($\mathbf{z_1, z_3}$) are zero and $\mathbf{z_2}$



(a) Duoplex Matrix for equations (3.1.5)



(b) Matrix Output at Termination, Infeasibility Detected

Figure 3.2.1 Duoplex Matrix

is less than or equal to zero. Since the slack variable (y_1) is always non-negative, the "≤" constraint is always satisfied. An infeasibility is detected by the Duoplex algorithm when it is not possible to make all the auxiliary variables zero. Thus by identifying the non-zero auxiliary variables at the termination of the Duoplex method, the locations of the inconsistencies can be detected. There are vectors available from the Duoplex output that keep track of all the variables (including the auxiliary and slack variables). Thus if an auxiliary variable is greater than zero, given in column 1 of the Duoplex output matrix (figure 3.2.1b), the corresponding constraint is not satisfied and the value of that auxiliary variable gives the amount by which the constraint is violated. For the example considered here the values of the auxiliary variables are:

$$\begin{aligned} z_1 &= 0 \\ z_2 &= 2 \\ z_3 &= 2 \end{aligned} \tag{3.1.6}$$

Thus the corresponding constraints in equations (3.1.5) are violated and need to be altered according to the values obtained for the auxiliary variables. The new corrected formulation is as follows:

$$\begin{aligned} \text{dummy} &= 1 \\ \text{STRB\_DEL} + \text{ODEL} &\geq 10 \\ \text{EN\_DEL} + \text{ODEL} &\leq 10 \\ \text{EN\_DEL} - \text{STRB\_DEL} &= 0 \end{aligned} \tag{3.1.7}$$

On solving this set of constraints the values obtained for the various generics are:

$$\begin{aligned} \text{STRB\_DEL} &= 0 \\ \text{EN\_DEL} &= 0 \\ \text{ODEL} &= 10 \end{aligned} \tag{3.1.8}$$

These are the precise steps implemented in the CONFLICT REMOVAL routine indicated in the flowchart for the TIMESPEC algorithm. The corrections that are required to be made to the given set of constraints are made automatically by the above routine or the user can intervene and make the modifications personally.

Since the given set of constraints can have many solutions, the one obtained may not be the one which is really desired. As in this case the STRB_DEL and the EN_DEL have been assigned zero delay. This is not desired because it is not possible to implement, at the silicon level, any functionality with zero delay. Thus, incorporating these delay values in the VHDL descriptions using the *imbedded timing*, provides no advantage over the *separated timing* method. The only assurance at the END OF STEP1 in the flowchart is a solution to a set of constraints which is free of inconsistencies. These models can only be tested for those paths that are specified by the manufacturer's data sheets thus not justifying the use of *imbedded timing* in the VHDL models. This is because, the delays associated with the individual primitives are not realistic delays and hence the input to output delays for the paths not specified by the manufacturer's data sheets will also be unrealistic. In view of this fact, it is necessary to obtain delay values that are close to the desired primitive block delay values. The technique used to obtain realistic values for the block delays is termed as *Curve Fitting* [19].

3.3 A Practical Solution:

The use of *imbedded timing* in VHDL models can be favoured over the *separated timing* only if it can provide substantial advantages over the *separated timing* method of delay specification. Assigning realistic delay values to the generics associated with each

process (primitive block) opens up a large number of advantages that justify this use, as discussed in chapter 1. A major advantage observed is that it is possible to verify all the paths in the VHDL models for VLSI chips, thus making the models more accurate and complete. During the testing and verification process (simulation) of these models it is possible to identify any timing hazards internal to the chip. (Eg. Although two signals appear at the input of the chip at the proper time, the delays through the internal primitives are such that they appear at the input of an internal latch in improper sequence). Since the timing delays assigned are close to actual primitive delay values, it is possible to synthesize the chip directly as designed or modeled in VHDL code by replacing the processes by the corresponding circuit blocks available. Thus the solution obtained at the end of the CONFLICT REMOVAL routine (END OF STEP1) is not sufficient and the CURVE FITTING routine (explained later in this section) needs to be invoked in order to obtain a practical solution.

It is observed that the timing specifications are usually an underspecified system (ie. the constraints are fewer than the variables). Thus the variables in the constraints have a high degree of freedom and hence the values obtained for the variables in the first step are usually far form realistic as seen in the previous section. It is necessary to put more bounds on the original set of constraints, forcing the variables to assume values close to the desired primitive block delay values. A technique called *Curve Fitting* is used in meeting this objective.

Additional information regarding the primitive block delay values needs to be provided which will be used in obtaining a realistic solution for the Timing Distribution Problem. When these new constraints are added it is important to differentiate them from

the original set of constraints. The original constraints are called the *core constraints* and the newly added constraints are called the *soft constraints*. The reason for this terminology will become apparent shortly.

Before defining the two terms, viz. core constraints and the soft constraints, two *viewpoints* are defined for VHDL users based on the application for which the VHDL models are created.

- A *designer's point of view* is defined for a VHDL user who uses the VHDL modeling and simulation capabilities for designing a chip. The designer usually starts with the input-to-output specifications for a chip to be designed. Then a partial top-down design technique is used to obtain the final design. The designer starts with an abstract model of the chip and then gradually proceeds to obtain a description in terms of the functional blocks, some of which are directly available as cells in a library and some of which need to be designed and implemented separately. Due to this partial availability of library cells, the term partial top-down design is used.

- A *modeler's point of view* is defined for a VHDL user who uses VHDL for modeling a given chip (already designed), to be incorporated in a bigger system. The specifications available to the modeler are the manufacturer published data sheets. The modeler does not have any information regarding the silicon level implementation of the chip.

These two view-points are identified by calling the VHDL user a *chip designer* or a *chip modeler*.

Having defined the above two terms, the concept of core constraints and the soft constraints can be better explained.

- For a *chip designer* the core constraints are the input-to-output timing specifications upon which the design of the chip is based and hence are necessarily error free otherwise the chip design will be fallacious too.
- The core constraints available to a *chip modeler* are the manufacturer's published specifications. This data sheet information is based on experiments for which the operating conditions may vary between measurements thus leading to an inconsistent specification. Errors in the printing of the data sheets cannot be ruled out either.

These core constraints available to the designer/modeler are solved in the first step of the TIMESPEC algorithm. Since there is a likelihood of contradictions the CONFLICT REMOVAL routine described in the previous section is applied to obtain a consistent set of constraints. Thus as mentioned in the previous section a solution is obtained after correcting any inconsistencies, if any exist (END OF STEP1 in the flowchart). It was mentioned that soft constraints are required to be added to obtain a better delay allocation. The soft constraints which are nothing but additional bounds on the feasible region for the solution are now explained.

The *chip designer* needs to distribute the input-to-output timing constraints among the various functional blocks of the design such that the functional blocks corresponding to the cells from the cell library are assigned their corresponding delay values. Then the timing values for the remaining blocks, which are to be implemented at the silicon level, are the timing restrictions within which these remaining functionalities

need to be implemented. Thus the soft constraints, from the designer's point of view, are the timing delay values associated with the blocks corresponding to the cells from the cell library. Additional **soft** limits (maximum/minimum ranges) can be provided as soft constraints for the non-standard blocks (to be implemented).

The silicon level implementation of the various functional blocks in the chip description are not known to the *chip modeler*. Thus the only way to predict or approximate these values to obtain a better solution than the one obtained at END OF STEP1 in the flowchart is to use typical delay values for the functional blocks, obtained by surveying various chip specifications and design cell libraries for the considered technology. *The assumption made here is that a typical functional block such as a latch, buffer etc. will be implemented using a standard cell and thus the delay values are assumed to correspond to these standard cell delays.* These delays are used as the soft constraints for the chip-model. Figure 3.3.1 shows a listing of some of the typical blocks and their associated timing delays obtained from the CMOS cell library [24].

The soft constraints (desired values) used by the designer or modeler may not necessarily be the exact value that can be assigned to the blocks under the core constraint specifications. Thus the soft constraints are allowed to vary from their assigned values so that the core constraints are not violated. This is the precise reason for this particular nomenclature. By differentiating the two set of constraints, priority is given to the core constraints and thus any contradictions resulting by solving the formulation using the modified Duoplex method are attributed to the soft constraints only. The soft constraints are then adjusted keeping the core constraints intact. This is the basis for the CURVE FITTING routine used.

CMOSN Cell Library 1.2 micron technology Load Capacitance C_L = 10 pf Temperature @ 25° C	
FUNCTION	DELAY (ns)
Inverter	4
2 Input Nand	7
2 input Nor	6
Exclusive-OR	11
Tristate-Buffer Data-Out En-Out	 5 5
Clocked Latch (W/Reset) Clock-Qbar Reset-Qbar	 8 8
D-Flip-Flop (W/Reset) Clock-Q Reset-Q	 5 4
Mux	15
Decoder C-D En-D	 9 7

Figure 3.3.1 Typical Delays for Typical Primitives

Consider the following set of soft constraints:

$$x_1 \leq 8$$
$$x_2 \geq 12 \qquad (3.3.1)$$
$$x_3 = 10$$

These soft constraints are made flexible by viewing them as follows:

$$x_1 + u_1 - v_1 = 8$$
$$x_2 + u_2 - v_2 = 12$$
$$x_3 + u_3 - v_3 = 10$$
$$u_1 \leq lower$$
$$v_1 \leq margin \qquad (3.3.2)$$
$$u_2 \leq margin$$
$$v_2 \leq upper$$
$$u_3 \leq margin$$
$$v_3 \leq margin$$

lower, *upper* and *margin*define the limiting range for the soft constraints. *lower* and *upper* are used for the "≤" and "≥" constraints respectively, along with margin. Whereas, for the equality constraint, only *margin* is used. Thus, for the example in equations (3.3.1) the soft constraints can have values in the following ranges:

$$(8 - lower) \leq x_1 \leq (8 + upper)$$
$$(12 - margin) \leq x_2 \leq (upper + 12) \qquad (3.3.3)$$
$$(10 - margin) \leq x_3 \leq (10 + margin)$$

The value of *lower* is made equal to the right hand side of the corresponding constraint so as to have a lower limit of zero for the "≤" constraints. The value of *upper* can be made arbitrarily high. It is important to convert all the constraints into equalities to make them flexible as shown above, in order to avoid tampering with the core constraints. This is

especially true for the "≤" constraints because the Duoplex algorithm assigns highest priority to the "≤" constraints and thus they are always satisfied. After converting the soft constraints into the format in equations (3.3.2), the entire system of core and soft constraints is solved. The **u** and **v** variables are minimized in order to satisfy the soft constraitns as closely as possible within the core constraint restrictions. Now if an infeasibility occurs then the following steps are taken.

- Identify the *soft constraints* for which the auxiliary variables are non-zero. Thus the constraints leading to an infeasible solution are identified.
- The **u** and **v** variables corresponding to these constraints are examined. Those **u** or **v** variables that have reached their maximum limit are identified and their limits are increased by the value of the auxiliary variable. Thus the ranges of the soft constraints are relaxed.
- If none of the auxiliary variables corresponding to the soft constraints are non-zero, but instead the auxiliary variables corresponding to the core constraints are non-zero, then also the infeasibility is due to the newly added soft constraints. This is because the infeasibility occurred only after the addition of the soft constraints. In this case it is very difficult to identify the soft constraints that cause the infeasibility. It is observed that the **u** and **v** variables that have reached their maximum limit identify the soft constraints that are satisfied at the boundary of their limiting range. Thus, these constraints are the possible cause of the infeasibility. Therefore, in order to relax the soft constraint ranges the upper limits on these **u** and **v** variables are raised by a small amount.
- Solve again and repeat the above steps until a feasible solution is reached.

This completes the second step of the algorithm (END OF STEP2 in the flowchart) in which a practical solution is obtained after a feasible solution has been guaranteed at the end of the first step.

For the register example considered the solution obtained at the END OF STEP1 was

$$\begin{aligned} \text{STRB\_DEL} &= 0 \\ \text{ODEL} &= 10 \\ \text{EN\_DEL} &= 0 \end{aligned} \tag{3.3.4}$$

Suppose it is desired that the delay for STRB_DEL is 5 ns, EN_DEL is 5 ns and ODEL is 6 ns, then the soft constraints would be:

$$\begin{aligned} \text{STRB\_DEL} &= 5 \\ \text{ODEL} &= 6 \\ \text{EN\_DEL} &= 5 \end{aligned} \tag{3.3.5}$$

The solution obtained after applying the CURVE FITTING ROUTINE (with *margin* = 0.1) is:

$$\begin{aligned} \text{STRB\_DEL} &= 4.5 \\ \text{ODEL} &= 5.5 \\ \text{EN\_DEL} &= 4.5 \end{aligned} \tag{3.3.6}$$

Thus a very good delay allocation is possible using the Curve Fitting technique.

A Possible Problem:

Another example is considered to illustrate a possible problem that could be encountered. Consider the following set of constraints for the 8-bit register:

$$STRB\_DEL + ODEL \geq 12$$

$$EN\_DEL + ODEL \leq 10 \qquad (3.3.7)$$

$$EN\_DEL + STRB\_DEL = 7$$

The solution obtained after the application of TIMESPEC at END OF STEP1 is :

$$STRB\_DEL = 7$$

$$ODEL = 5 \qquad (3.3.8)$$

$$EN\_DEL = 0$$

Since the solution obtained is not the desired solution, the soft constraints given in equations (3.3.5) are added to obtain a better delay allocation. The solution obtained at the end of the CURVE FITTING routine, ie., at END OF STEP2 , is:

$$STRB\_DEL = 5.3$$

$$ODEL = 6.7 \qquad (3.3.9)$$

$$EN\_DEL = 1.7$$

It is observed that EN_DEL delay is very low in comparison with the expected delay provided in the soft constraints. Thus, in this example, the addition of the soft constraints fails in obtaining a desired solution. The reason for this is that the core constraints are very restrictive and the delay allocation expected by the addition of the soft constraints for the given core constraints is incorrect. If this condition occurs for a *chip modeler* who has no information about the implementation of the various functional blocks then there are two possibilities:

1. The core constraints provided by the manufacturer are erroneous.
2. The implementation of the functional blocks is a non-standard implementation (or not according to the standard blocks chosen by the *chip modeler*) and thus the timings specified in the data sheets are correct whereas the soft constraints are incorrect and the delay allocation obtained by TIMESPEC is indeed correct.

The *chip modeler* is required to make a proper choice from among the two possibilities considered above before proceeding with the modeling of the chip. It is very important to take a proper decision otherwise a correct formulation may be altered to an incorrect one.

If such a situation is encountered one way of dealing with this is to specify, as core constraints, the minimum and maximum delay limits for the generics for which undesired values have been obtained. These limits can be the minimum or maximum delays possible in any implementation of the functional blocks comprising the chip for the considered technology. If an inconsistency is detected when the core constraints are solved alone (END OF STEP1), the original core constraints can be assumed to be erroneous since the implementation of the various functional blocks is not possible within the technological constraints. Then the inconsistencies for this formulation will be removed by TIMESPEC and then the soft constraints are applied towards obtaining a desired delay allocation.

In the Tri-state register example considered, with the core constraints as given in equations (3.3.7), the value of the generic EN_DEL is found to be 1.7 ns. Let us assume that the implementation of the AND gate (figure 3.1.1), with which the generic EN_DEL is associated, is possible only if the value of EN_DEL is greater than 3.5 ns. Thus the following constraint is added to the core constraint formulation of equations (3.3.7):

$$EN\_DEL \geq 3.5 \qquad (3.3.10)$$

On solving the new core constraints using TIMESPEC an inconsistency was detected which resulted in the modification of the given core constraints by the CONFLICT REMOVAL routine. The modified core constraints are:

$$EN\_DEL \geq 3.5$$

$$STRB\_DEL + ODEL \geq 10$$

$$EN\_DEL + ODEL \leq 10 \qquad (3.3.11)$$

$$EN\_DEL + STRB\_DEL = 7$$

The solution obtained at END OF STEP2, ie. after the application of the CURVE FITTING routine, is:

$$EN\_DEL = 3.5$$

$$STRB\_DEL = 3.5 \qquad (3.3.12)$$

$$ODEL = 6.5$$

This example illustrates the use of TIMESPEC in dealing with timing specifications that are very contradictory and restrictive and for which the Timing Distribution Problem is otherwise extremely difficult. Another example in which the above mentioned problem occurs is the Arithmetic Logic Unit (ALU) discussed in chapter 5.

The *chip designer* does not face this particular problem. This is because the core constraints are necessarily error free. Thus the only possibility remaining is that the choice of the design cells is erroneous. Thus the designer can use a different set of cells (soft constraints) and try the delay allocation again or in the worst case the various functional blocks can be implemented at the silicon level using non standard cells. It is not very likely that such a situation will arise because the timing specifications available to the designer are usually in the form of a set of maximum/minimum ranges and thus the core constraints are not very restrictive. Thus the delay allocation using the proper design cells can be easily achieved.

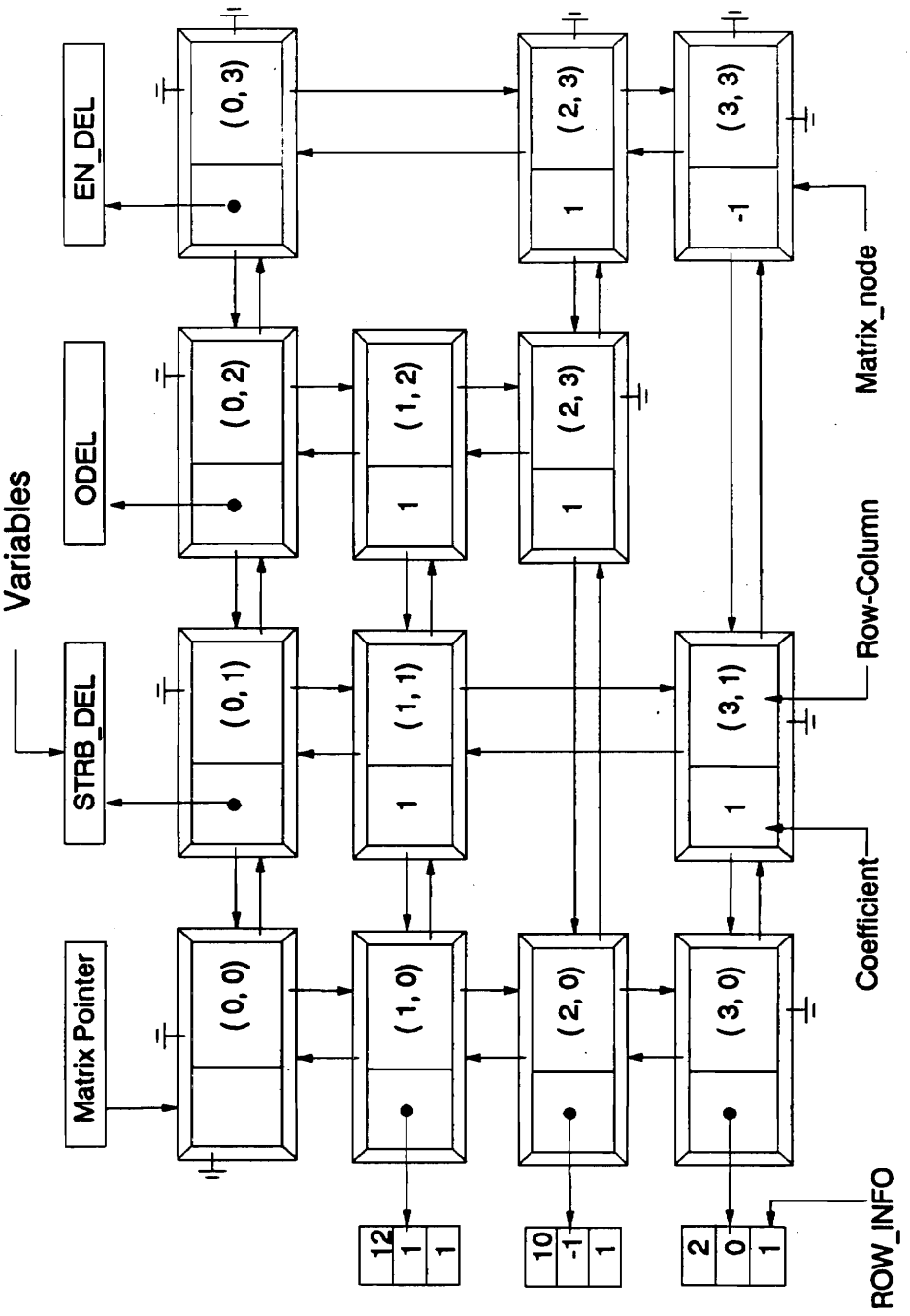


Figure 3.4.1 Matrix Data Structure

3.4 Other Programming Issues:

The core and soft constraint formulation is specified in a file which is then made available to TIMESPEC. Since the number of variables and the number of constraints is not known initially, the information extracted from the file needs to be stored in dynamically allocated memory. The obvious choice of data structure is a linked list [25,26]. A doubly linked list in the form of a matrix, shown in figure 3.4.1, is used, which is employed fairly often in storing sparse matrices [25]. The programming language used is C [26] where the structure is defined as follows:

```
typedef struct {
   int              row, col;
   generic_ptr      *datapointer;
   matrix_node      *nextrow, *prevrow;
   matrix_node      *nextcol, *prevcol;
} matrix_node;
```

Each constraint corresponds to a line in the input file. The fields that are scanned by the PARSE routine are, the coefficient (with sign) of the variable, the variable names, the symbol ($\leq$,=,$\geq$) and the right hand side value. Any field occurring out of proper sequence (Eg. x $\pm$ y + z = 10) is detected and the user is notified about the bad input formulation. As each line in the file is scanned by the text parser, the extracted information is added to the above defined `matrix` data structure. The row information in the file corresponds to the rows in the `matrix` whereas the `matrix` columns correspond to the variables. Thus a new row is added for each new line and a new column corresponds to a newly encountered variable. The coefficients of the variables are added at the (**i**,**j**) position of the `matrix`, where **i** is the **i**th row and **j** corresponds to the variable (column) under consideration. The `datapointer` field of the `matrix_node` points to the coefficient value. The `datapointer` in the row zero points to the variable name. Thus the

variable names are also stored in the same data structure along with the other information. This information is useful at the time of the solution output. The `datapointer` in column zero points to a structure, `ROW_INFO`, which stores the information about the row.

```
typedef struct {
   float  RHS;
   int    LEG;
   int    CORE;
} ROW_INFO;
```

The `RHS` field stores the right hand side value of the corresponding row. The `LEG` field stores integers **-1**, **1** or **0** indicating whether the constraint is a "$\leq$", "$\geq$" or "=" constraint respectively. The `CORE` field specifies whether the constraint is a core (**1**) or soft(**0**) constraint. Once the complete `matrix` data structure is set up, the routine (FORMULATE) which converts this information into the exact Duoplex format is invoked. This FORMULATE routine merely rearranges the rows according to the Duoplex tableau format (2-D array) and makes it available to the core routines of TIMESPEC. This routine also takes care of the insertion of the dummy variable, dummy objective function or any other specific objectives, and in case of the CURVE FITTING routine, the **u**, **v** variables. Thus the input required to solve the Timing Distribution Problem is now available.

3.5 Summary:

Having explained all the building blocks essential for solving the Timing Distribution Problem, a brief summary of the algorithm is now provided to obtain a global view of the algorithm. The flow-chart for the algorithm is shown in figure 3. The

PARSE routine, explained in the previous section, accomplishes the text parsing of the constraints provided by the user in textual form. In the first step the core constraints are made available to TIMESPEC by the user. This information is then converted to the Modified Duoplex format by the FORMULATE routine. Thus the core constraints are available in the format required by the Duoplex method at this point. The Modified Duoplex Method (SOLVE) is then applied to the formulation. If any inconsistencies are detected, the CONFLICT REMOVAL routine is invoked which modifies the given constraints so as to obtain a feasible solution. At this point (END OF STEP1) a solution to the given set of constraints is available. In addition the given constraints are rid of the inconsistencies, solving a major portion of the problem.

Soft constraints are now added by the user with the objective of obtaining a realistic solution. The Modified Duoplex Method (SOLVE) is applied again to this new formulation. The CURVE FITTING routine is invoked until a much more realistic approximation of the delay values is obtained (END OF STEP2).

At every step in the algorithm, user intervention is allowed. Thus the user can make necessary modifications of corrections to the core/soft constraints according to the solution requirements.

Chapter 4.

Interface With the Modeler's Assistant

The Modeler's Assistant [8,17] is a graphical CAD tool which provides a graphical interface for VHDL model development. The inclusion of TIMESPEC as part of the Modeler's Assistant is discussed in this chapter. Before proceeding with the discussion about the interface between the Modeler's Assistant and TIMESPEC a brief introduction of the Modeler's Assistant and its data structures is addressed.

4.1 The Modeler's Assistant:

The Modeler's Assistant provides an X-windows environment for the input of the information required for generating VHDL models. The models are restricted to being behavioral only and are represented by a PMG. The PMG is input interactively in the Modeler's Assistant environment and this tool generates the corresponding VHDL code.

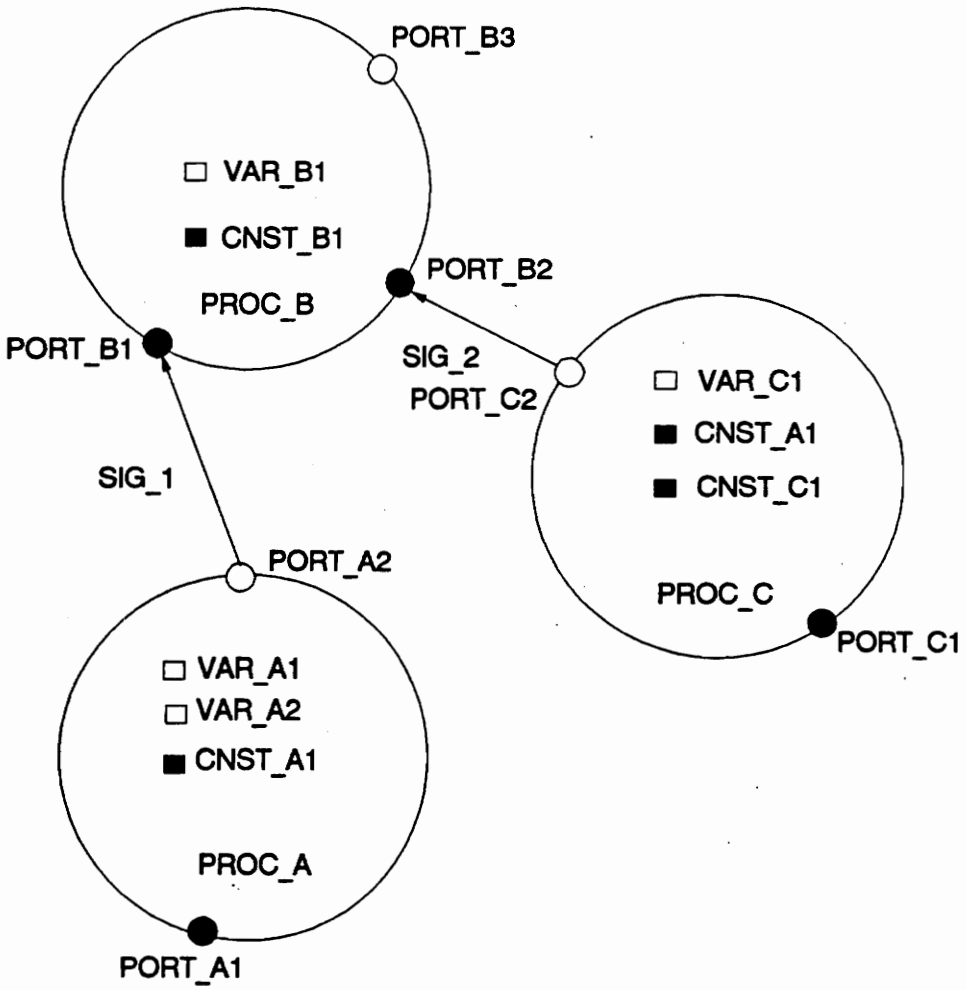


Figure 4.1.1 Sample Process Model Graph

For the modeling style considered in this work, each process in the PMG corresponds to a primitive block in the block diagram description of the chip.

For each process corresponding to a primitive block, the information associated with it viz. ports, constants, variables and the generics is provided interactively in the graphical X-windows environment (figure 4.1.1). The sequential portion which describes the functionality of the process is input textually in an editor window (figure 4.1.2). Once all the processes corresponding to all the primitive blocks in the chip (system) are available, either through direct input or through a library of primitives, then a complete model is compiled. This is called a ***unit*** in the Modeler's Assistant jargon. The various processes are placed in a window and the wire connections between the processes are provided through the signals that facilitate the communication between the various processes. This is the concurrent portion of the PMG and is added graphically. From the description of the creation of the unit (PMG), it is clear that this environment is particularly suited for describing a behavioral model of a system (chip) that utilizes the *imbedded timing*. Once the PMG is completed the VHDL code for the unit is generated. This exemplifies the time saving achieved in building a VHDL model by making use of the CAD tools. The detailed information on building a process model graph and the corresponding VHDL models is provided in reference [17].

At this stage the generic names corresponding to the delays in the primitive blocks in a chip are known to the entity description of the model but the actual values to be used for these generics are not available. The objective of interfacing TIMESPEC with the Modeler's Assistant is to add this capability, of assigning numerical values to the generic names, to the Modeler's Assistant.

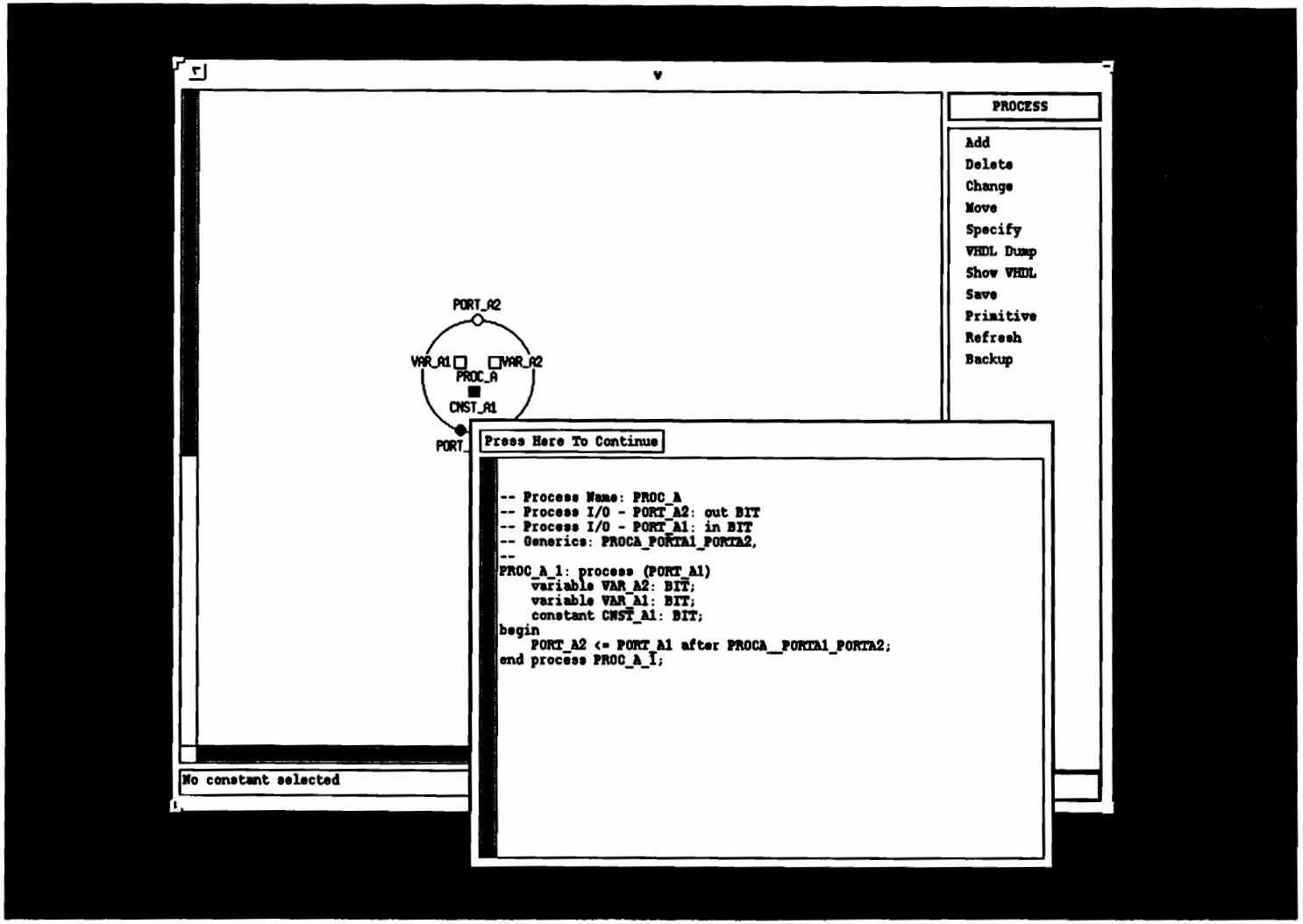


Figure 4.1.2 Text Window displaying the VHDL code

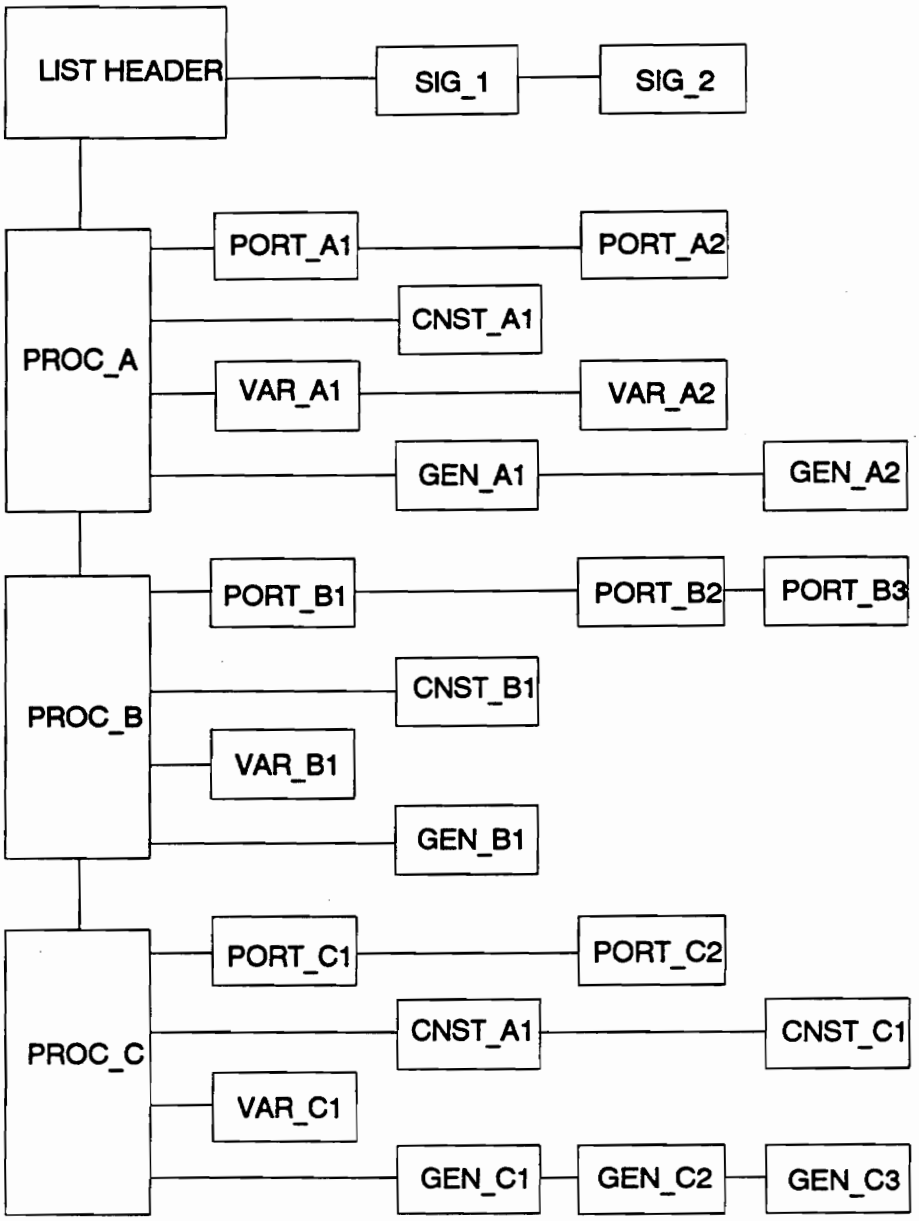


Figure 4.2.1 Linked List for Storing the PMG of figure 4.1.1

4.2 Data Structures:

In order to interface TIMESPEC with the Modeler's Assistant, it is necessary to extract information associated with the PMG and use it to generate the textual input required for TIMESPEC. The Modeler's Assistant stores all the information regarding the PMG in a linked list as shown in figure 4.2.1. Actually two linked lists store the complete PMG, one for storing the process information and the other for storing the signal interconnection information. Each node in the list is defined as follows:

```
typedef struct {
   char name[32];
   int  type;
   int  ptr[6];
   rect R;
} a_node;
```

The structure field `name` stores the name of the construct. The type of the construct (signal,generic,port etc.) is stored in the field `type`. Pointers to other nodes are stored in the `ptr[6]` field. The `rect` field is a structure that stores the information regarding the display to the screen. Since the graphical interface is not considered in the interface between TIMESPEC and the Modeler's Assistant, this information is entirely ignored.

4.3 Problem Definition:

The objective of interfacing the Timing Distribution Problem solver software tool TIMESPEC with the Modeler's Assistant is to provide the actual delay values to the generic names that are provided as part of the PMG description in the Modeler's Assistant environment. In adding this capability to the Modeler's Assistant it is observed that the capabilities of both the Modeler's Assitant and TIMESPEC are substantially

enhanced. The interface problem deals with extracting the necessary information from the PMG data structure and formulating it into the textual input format recognized by TIMESPEC. *This can be achived by identifying all the paths from all primary inputs to all primary outputs in the PMG and enumerating them in terms of the generic names traveresed by these paths.* For example, a path from the primary input A to the primary output F, for the PMG shown in figure 4.3.1, is defined in terms of the process generics as follows:

P1__A_C + P2__D_F

The dotted lines indicate the signal assignments internal to the processes and the corresponding generics associated with these paths are given in the parentheses. In order to represent the paths in terms of the generic names, it was found convenient to put the following restrictions on the generics :

- In a given process a generic **must** be associated from an input port to an output port where a path exists between these process ports. Thus, for the process P1, shown in figure 4.3.1, paths exist from port A to port C and from port B to port C. A generic name P1__A_C is associated with the path from port A to port C, but if no generic is associated with the path from port B to C then this path will be ignored since the above restriction is not met. If a δ-delay is required for a generic, a value of zero can be assigned later on to the generic name provided.
- There is a restriction on the way the generic name is specified. The generic name must have the following format:

  xxxxxxx__ip_op(_#)

  The total length of the generic name must be less than 32 characters. The ***xx***'s can be any character string, but for clarity it is recommended that this be the name of the process to which the generic belongs.

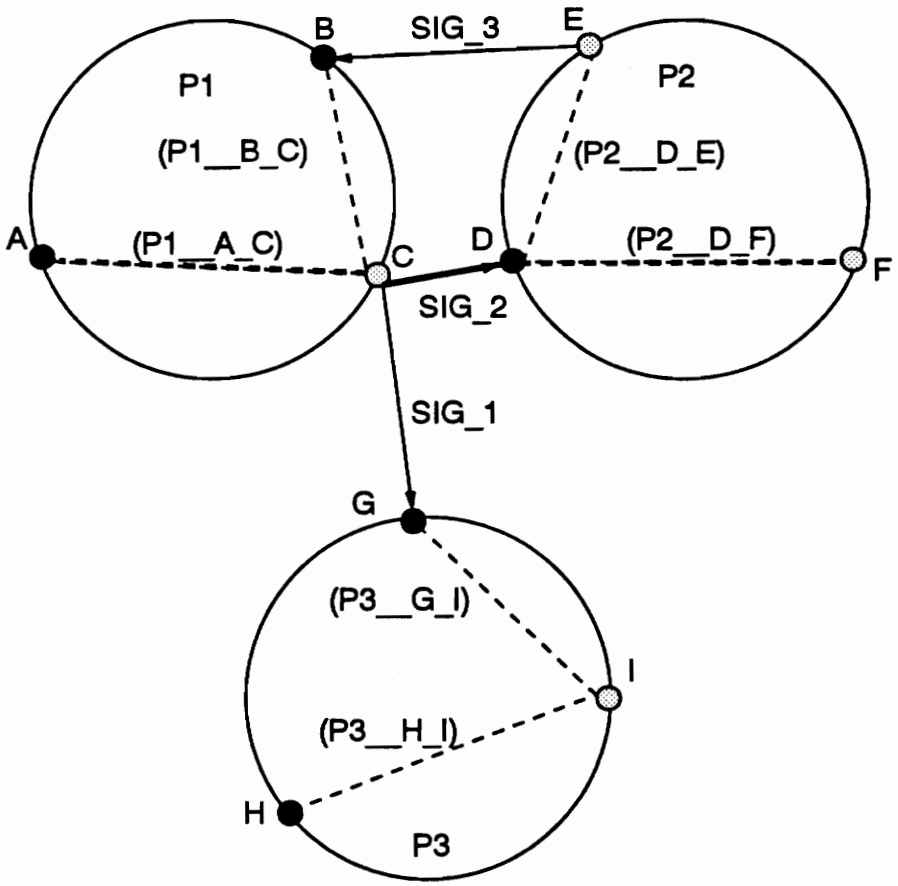


Figure 4.3.1 PMG used as an example for Path Enumeration

ip must be the name of the input port and ***op*** must be the name of the output port corresponding to the process, where the generic provides the time delay for an input-to-ouput port signal assignment. In case of multiple delays between the same input-output pair, the generic names are distinguished by a number (**#**). This is used as a counter for keeping track of the multiple generics. This is the optional field and is used only when there are multiple delays. The delimiters '_' used, two before ***ip***, one before ***op*** and in case of multiple delays, one before **#**, are mandatory.

These restrictions are essential to facilitate the identification of the port input-output pairs associated with the generics and hence make path enumeration possible.

4.4 Mathematical Model:

The information which describes the entire PMG is available through the linked list data structure in figure 4.2.1. This information is to be used to solve the path enumeration problem defined in the previous section. A very efficient way of doing this was to first convert the information available into a directed graph [27,28] in which the *edges* of the graph represent the generic names (process input-to-output connections). The *vertices* represent the signals in the PMG. Signals with a common source or destination are lumped together to form a single vertex. This will become clear from the example conversion shown in the figure 4.4.1 for the PMG described in figure 4.3.1. In this figure the digraph is converted into a single-source and single-destination graph. This form is required for applying the path enumeration algorithm [29] to obtain all the possible paths from the source to the destination by the addition of vertices ***S*** and ***T*** and the corresponding edges.

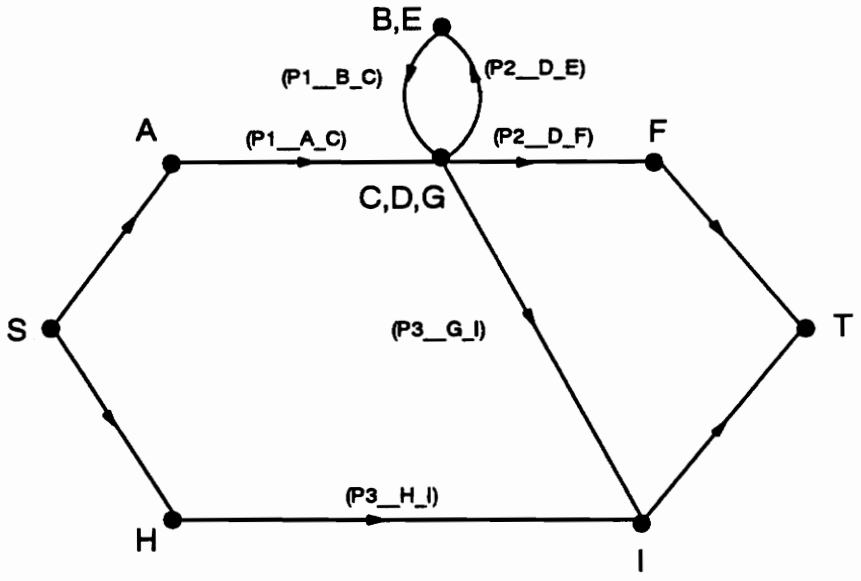


Figure 4.4.1 Directed Graph for the PMG in figure 4.3.1

Edges and the corresponding Generics

1 P3__H_I

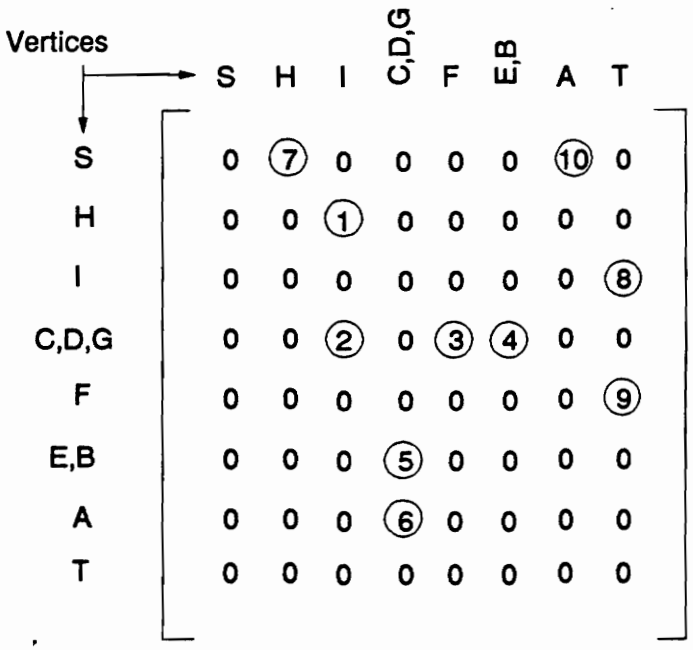
2 P3__G_I

3 P2__D_F

4 P2__D_E

5 P1__B_C

6 P1__A_C



Successor Matrix

Example Path:

Path in terms of vertices (S A C E C F T)

Path in terms of edges (0 6 4 5 3 9)

Path in terms of generic names

P1__A_C + P2__D_E + P1__B_C + P2__D_F

Figure 4.4.2 Successor Matrix Format

The digraph thus formed is represented in a matrix called the *successor matrix* [28]. The successor matrix is a ***nxn*** matrix in which ***n*** corresponds to the total number of vertices in the digraph. The ***i***th row and the ***i***th column of this matrix correspond to the same vertex whereas the ***i***th row and the ***j***th row correspond to different vertices for ***i*** and ***j*** not equal. The columns corresponding to the non-zero entries in any row represent the successor vertices for the vertex represented by the row. In other words, the vertices that are connected, by directed edges, to the current vertex are marked by the non-zero entries in the successor matrix. In this way the entire digraph is defined by the successor matrix [28]. The successor matrix for the digraph shown in figure 4.4.1 is indicated in figure 4.4.2.

4.5 Successor Matrix Formulation:

The PMG information essential for the successor matrix formulation is:

1. The generic names in the proper format and the process to which they belong.
2. The port names, their corresponding processes and the signals originating or terminating in these ports.
3. The signals, the names of their source and destination ports and the processes to which these ports belong.

All this information is available in the linked list data structure describing the PMG shown in figure 4.2.1. The generic and port information is obtained from the process linked list whereas the signal information is obtained from the signal linked list. The steps involved in obtaining the successor matrix (digraph) from this information are now enumerated:

- The generic names are scanned sequentially as they are obtained from the linked list data structure. For each generic name, the process input port and the process output port are extracted by parsing the generic name. The input format for the generics is provided precisely to incorporate this information which is used in formulating the successor matrix. Since the port names in different processes can be identical, the processes to which these ports belong are identified as well to remove any ambiguity. This information is also available from the linked list.
- For a process input port, the signal terminating in this port is identified. The source port (along with its process) of the signal is also identified. These two ports correspond to the same vertex in the successor martix and thus if any row has already been assigned to any of the two ports then the same row is considered for the *current source vertex*. If no row has already been assigned then a new row is made available for the source vertex.

  If no signal terminates in the process input port then this port is identified as a primary input (pin input for the chip) and a new row is assigned for this source vertex.
- The same operation is performed in determining the column for the process output port except that the signal identified with this port originates at the current output port. By identifying the process output port the succesor vertex for the current source vertex is identified.

  In case the output port has no signals originating from it, it is identified as a primary output and a new column is assigned to this vertex.
- A non-zero entry is placed at the location identified, by the row and column selected in the previous step, in the successor matrix.

This non-zero value entered in the matrix is a pointer to the generic name under consideration. By storing the pointers to the generic names in the successor matrix, the operation of textual path output in terms of the generic names becomes easier. The enumerated paths are represented in terms of the edges in the graph which correspond to the non-zero entries in the successor matrix (figure 4.4.2). These edges are represented by the non-zero entries in the matrix which are pointers to the generic names (edges) and hence obtaining the paths in terms of the generic names is a simple matter of replacing the edges by the contents of these pointers.

- When the current vertex is identified as a primary input then a non-zero entry (not a pointer to any generic name, but an identifier for the edge formed) is placed at position identified by the corresponding column and the zeroth row. Thus this primary input is the successor of the additional vertex added to convert the digraph into a singe source digraph. Similarly, for the primary output, an edge is defined between this vertex and the added single destination by placing an identifying non-zero entry in the position identified by the row corresponding to the primary output and the last column in the matrix which corresponds to the single destination vertex.

There is a special case that needs more attention. When there are multiple generic names between the same pairs of process input and output ports, only a single path is assumed between these ports as far as the path enumeration algorithm is concerned. A track of these multiple generic names is maintained using the # field in the generic name and at the time of path output this multiplicity is accounted for. Consider a path

P1__A_C + P2__D_E + P1__B_C + P3__G_I

Suppose there are two generics between ports A and C and three between G and I. Then this path actually represents six different paths and these will be output with the proper sufixes attatched as shown:

P1__A_C_1 + P2__D_E + P1__B_C + P3__G_I_1

P1__A_C_2 + P2__D_E + P1__B_C + P3__G_I_1

P1__A_C_1 + P2__D_E + P1__B_C + P3__G_I_2

P1__A_C_2 + P2__D_E + P1__B_C + P3__G_I_2

P1__A_C_1 + P2__D_E + P1__B_C + P3__G_I_3

P1__A_C_2 + P2__D_E + P1__B_C + P3__G_I_3

The successor matrix formulation for the digraph depicted in figure 4.4.1 is shown in figure 4.4.2. The pointers to the edges are indicated in the listing of all the generic names enumerated in the figure.

4.6 Path Enumeration:

The digraph is stored in the successor matrix format because this format is ideally suited for the path enumeration algorithm that is employed here. The algorithm is an extension of the path enumeration algorithm developed by Asano and Sato [29]. Their algorithm deals with enumerating all input to output paths for acyclic graphs [27,28]. The algorithm considered here is not restricted to this special case. The edges in the diagraph considered here represent the paths internal to the various functional blocks (processes) in the chip. The input-to-output path identifies the various blocks encountered in going from an input pin to an output pin. Thus an input-to-output path in the digraph does not traverse any edge more than once since no path internal to a

functional block of a chip will be travered more than once for any given input. Although the diagraph can be cyclic, no egde is repeated for a given path. This feature is incorporated as an extension to the algorithm developed by Asano and Sato. A sightly different terminology is used here in extending this algorithm.

Given a digraph G corresponding to a PMG representation of a chip, another digraph G^* is obtained by adding a source vertex **s**, a terminal vertex **t**, a directed edge **(s,v)** for every starting vertex **v** and an edge **(v',t)** for every ending vertex **v'**. Thus an input-to-output path from a primary input v_1 to a primary output v_m in a PMG is defined as:

$$P^* = (s,v_1,v_2.....v_m,t).$$

The successor vertices for a given vertex, represented by a row in the successor matrix, are obtained by scanning the corresponding row from column 1 to **n** for non-zero entries. These non-zero entries corresponding to the edges in the digraph are used also as pointers to flags which indicate whether or not the corresponding edge is *available* for the current path. Edges are not *available* if the corresponding path has already been considered in the enumeration. Also, the edge is not *available* if the edge is already present in the current path.

The term $f_{as}(v)$ denotes the first *available* successor vertex of **v**. Thus the column corresponding to the first *available* non-zero entry in row **v** identifies the successor vertex of **v**. If no vertex is *available* then $f_{as}(v)$ = NIL. **v(i)** denotes the **i**th vertex in a given path.

The algorithm is now explained refering to the above defined terms.

Step 1. Find a path **P = (s=v(0),v(1).....v(q)=t)** between **s** and **t** by tracing $\mathbf{f_{as}}()$ to the successor at each node, starting at the source. Thus the following operation is performed:

$$\mathbf{v(i) = f_{as}(v(i-1))\ for\ i = 1,2.....until\ v(i) = t.}$$

Step 2. (Backtrack): Find the largest **i** such that $\mathbf{f_{as}(v(i))}$ is not NIL. If there exists no such **i** then all paths have been enumerated and the algorithm terminates.

Step 3. (Next Path): Set **v(i+1) = fas(v(i))** and then trace the $\mathbf{f_{as}}()$ from **v(i+1)** to the terminal **t** to find the other paths.

Go to Step 2.

By using this algorithm the paths corresponding to the PMG shown in figure 4.3.1 are enumerated below:

$$P3\_\_H\_I \leq 10$$

$$P1\_\_A\_C + P3\_\_G\_I \leq 10$$

$$P1\_\_A\_C + P2\_\_D\_F < 10$$

$$P1\_\_A\_C + P2\_\_D\_E + P1\_\_B\_C + P3\_\_G\_I \leq 10$$

$$P1\_\_A\_C + P2\_\_D\_E + P1\_\_B\_C + P2\_\_D\_F \leq 10$$

For each path that is output, an upper limiting delay value is provided which is defined by the user. Here, the "<" constraint is used for each path because the TIMESPEC software treats this as the "≤" constraint. This is because it is not possible to input the "≤" constraint using the standard keyboards. This upper limiting value should be made much

greater than the expected delay of the path with the greatest delay in the chip. Due to the large number of paths in the chips it was found convenient to provide this delay limit for all the paths and then allow the user to change the delay constraints by identifying paths according to the timing specifications. Thus, for the paths not specified by the timing specifications there is already an upper limiting value defined, such that these paths are non-restrictive and do not affect the solution obtained by TIMESPEC. The number of paths available through the timing specifications is small compared to the total number of enumerated paths and therefore this method of delay specification was found attractive.

4.7 TIMESPEC and Modeler's Assistant Interaction:

A global view of the interaction between the Modeler's Assistant and TIMESPEC is shown in figure 4.7.1. The first step is to create the PMG description of the system/chip to be modeled, using the Modeler's Assistant environment. The Modeler's Assistant creates a linked list data structure of all the information associated with the PMG. Once this is complete, the various input-to-output paths in a given PMG, built according to the restrictions provided in section 4.3, are enumerated automatically, thus making the core constraint formulation task substantially simpler. The user is required to modify or add only a few constraints to complete the core constraint formulation. The software tool TIMESPEC is then applied to the constraint formulation to solve the Timing Distribution Problem and allocate realistic delay values to the various generics that are input using the Modeler's Assistant.

The VHDL code for the given PMG is generated automatically by the Modeler's Assistant from the linked list structure and the functionality specified for each process in

the PMG. The generic values obtained by the use of TIMESPEC can then be used in this VHDL code or can be added to a module containing all the technology dependent information [30]. At this stage the mapping of the values into the VHDL code is required to be done manually. Automatic mapping can be incorporated as further developments are made to the Modeler's Assistant.

Delays associated with all the I/O paths in the PMG are also available from TIMESPEC. This facility provided by TIMESPEC is of immense importance in the design and modeling of VLSI chips. The input-to-output path delays associated with any input pattern are readily available without having to resort to actual measurement nor to the simulation of the chip. The availability of this information is of particular interest, since this enables the identification of path delays that are too long or too short and the designer/modeler can then make necessary modifications in the core/soft constraints to obtain the required delay values. Thus a complete and accurate VHDL model of a chip is possible by the use of *imbedded timing* discussed in chapter 1.

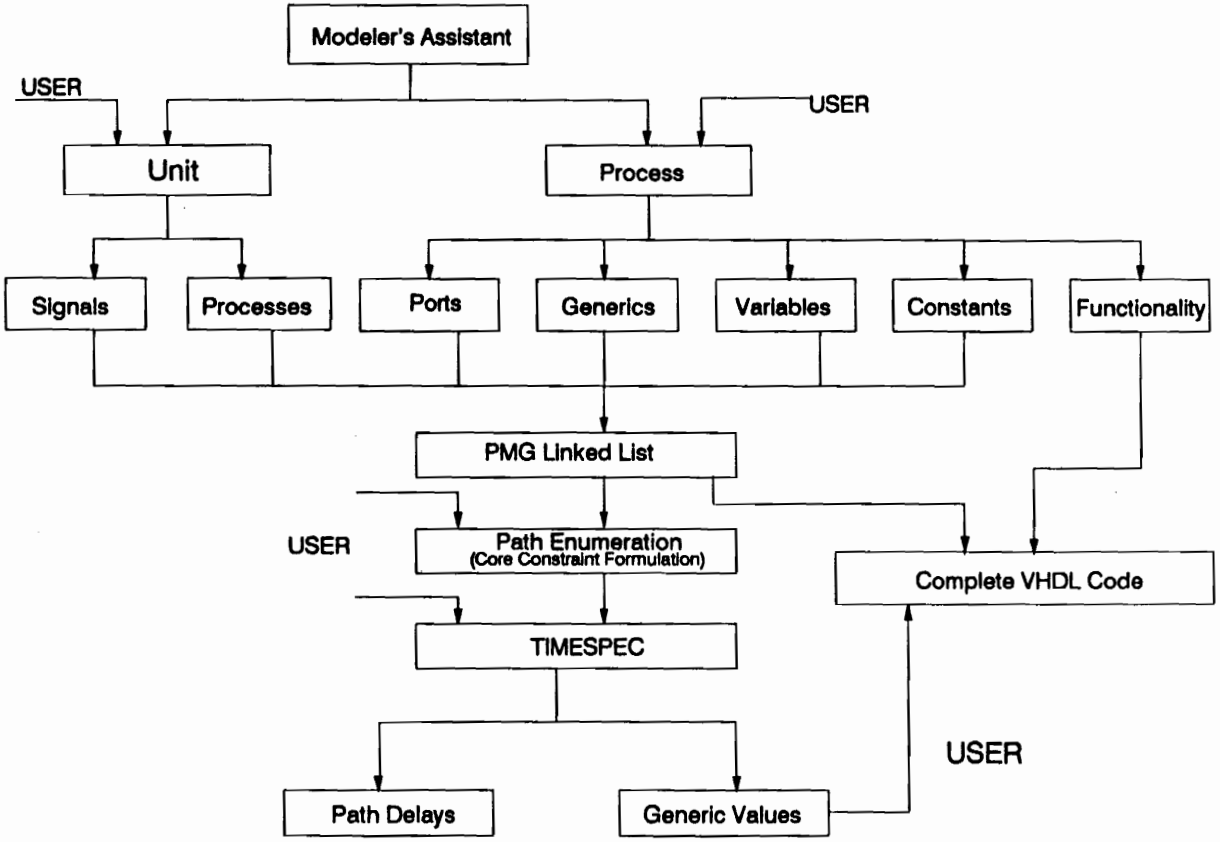


Figure 4.7.1 Interaction between Modeler's Assistant and TIMESPEC

Chapter 5.

Results

The results obtained by applying TIMESPEC to two chips will be examined closely to validate the usefulness of this tool as a part of the Modeler's Assistant. The two chips considered here are the Error Detection And Correction (EDAC) chip and the Arithmetic Logic Unit (ALU) (Appendix).

5.1 Error Detection And Correction Chip:

The PMG for the EDAC chip is shown in figure 5.1.1. Generic names that are associated with each process are also input as part of the PMG description. The generic names used are such that the processes and the input-output ports for which the generics are used can be easily identified. For example, the generic LATCH2__DB_DBL corresponds to the delay in the signal assignment from port DB to DBL in process LATCH2 shown in figure 5.1.1.

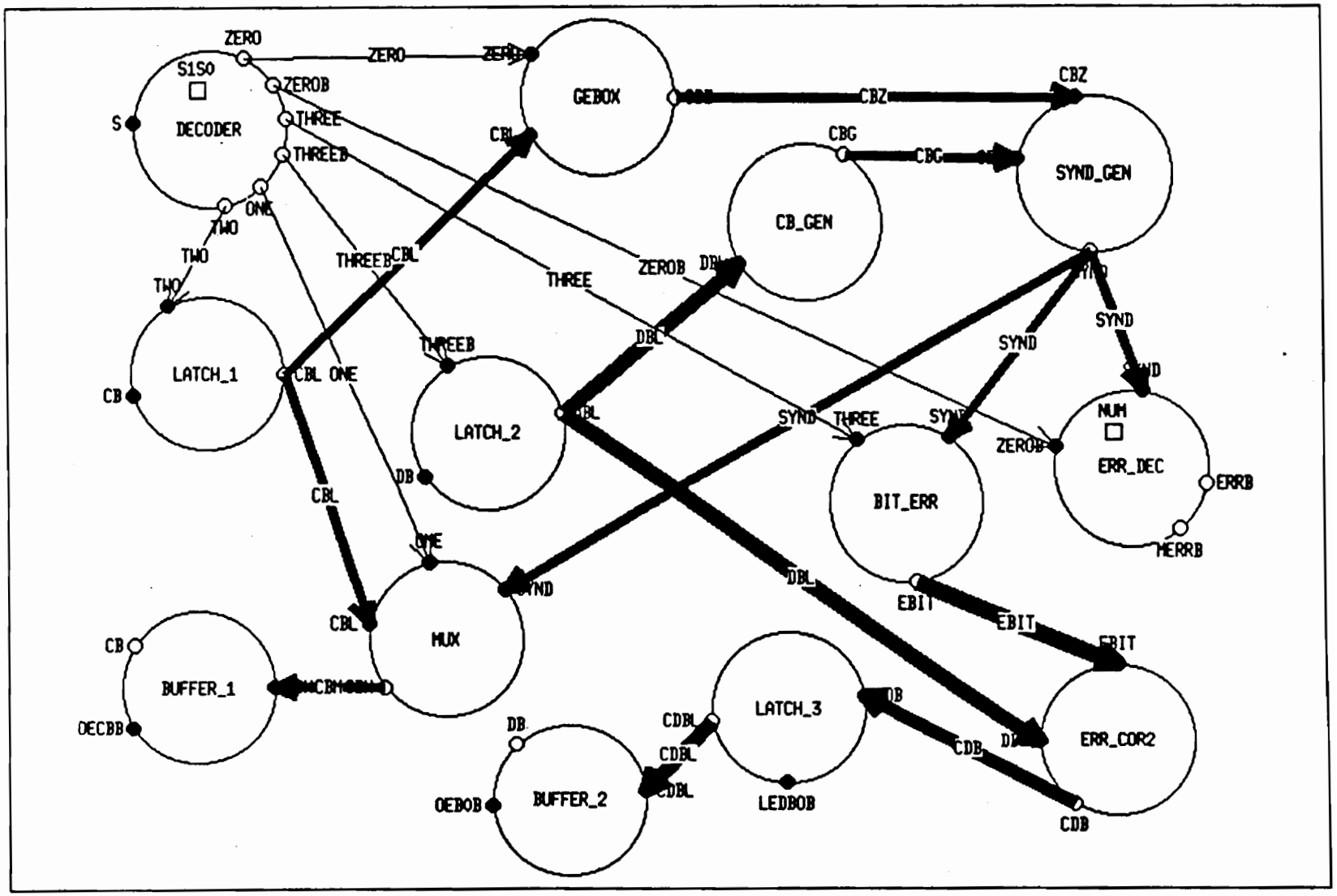


Figure 5.1.1. Process Model Graphs for the EDAC chip

```
     LATCH_2__DB_DBL + ERR_COR2__DBL_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB < 1000
1    LATCH_2__DB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + BIT_ERR__SYND_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 65
5    LATCH_2__DB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + MUX__SYND_CBM + BUFFER_1__CBM_CB = 40
3    LATCH_2__DB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + ERR_DEC__SYND_MERRB = 40
2    LATCH_2__DB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + ERR_DEC__SYND_ERRB = 26
9    BUFFER_2__OEBOB_DB < 24
6    LATCH_3__LEDBOB_CDBL + BUFFER_2__CDBL_DB = 26
8    BUFFER_1__OECBB_CB < 24
     LATCH_1__CB_CBL + MUX__CBL_CBM + BUFFER_1__CBM_CB < 1000
1    LATCH_1__CB_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + BIT_ERR__SYND_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 65
3    LATCH_1__CB_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + ERR_DEC__SYND_MERRB = 40
2    LATCH_1__CB_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + ERR_DEC__SYND_ERRB = 26
     DECODER__S_THREEB + LATCH_2__THREEB_DBL + ERR_COR2__DBL_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB < 1000
     DECODER__S_THREEB + LATCH_2__THREEB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + MUX__SYND_CBM + BUFFER_1__CBM_CB < 1000
     DECODER__S_THREEB + LATCH_2__THREEB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + ERR_DEC__SYND_MERRB < 1000
     DECODER__S_THREEB + LATCH_2__THREEB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + ERR_DEC__SYND_ERRB < 1000
     DECODER__S_THREE + BIT_ERR__THREE_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB < 1000
     DECODER__S_TWO + LATCH_1__TWO_CBL + MUX__CBL_CBM + BUFFER_1__CBM_CB < 1000
     DECODER__S_TWO + LATCH_1__TWO_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + MUX__SYND_CBM + BUFFER_1__CBM_CB < 1000
     DECODER__S_TWO + LATCH_1__TWO_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + ERR_DEC__SYND_MERRB < 1000
     DECODER__S_TWO + LATCH_1__TWO_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + ERR_DEC__SYND_ERRB < 1000
7    DECODER__S_ONE + MUX__ONE_CBM + BUFFER_1__CBM_CB = 40
     DECODER__S_ZEROB + ERR_DEC__ZEROB_MERRB < 1000
     DECODER__S_ZEROB + ERR_DEC__ZEROB_ERRB < 1000
1    DECODER__S_ZERO + GEBOX__ZERO_CBZ + SYND_GEN__CBZ_SYND + BIT_ERR__SYND_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 65
4    DECODER__S_ZERO + GEBOX__ZERO_CBZ + SYND_GEN__CBZ_SYND + MUX__SYND_CBM + BUFFER_1__CBM_CB = 40
```

Figure 5.1.2. EDAC Core Constraint Formulation

	t_w Pulse duration	$\overline{LEDBO}$ low		45	25	ns
	t_{su} Setup time	(1) Data and check word before $S0\uparrow$ (S1 = H)		15	12	ns
		(2) S0 high before $\overline{LEDBO}\uparrow$ (S1 = H)[†]		45	45	
		(3) $\overline{LEDBO}$ high before the earlier of $S0\downarrow$ or $S1\downarrow$[†]		0	0	
		(4) $\overline{LEDBO}$ high before $S1\uparrow$ (S0 = H)		0	0	
		(5) Diagnostic data word before $S1\uparrow$ (S0 = H)		25	12	
		(6) Diagnostic check word before the later of $S1\downarrow$ or $S0\uparrow$		15	12	
		(7) Diagnostic data word before $\overline{LEDBO}\uparrow$ (S1 = L and S0 = H)[†]		35	20	
	t_h Hold time	(8) Read-mode, S0 low and S1 high		35	30	ns
		(9) Data and check word after $S0\uparrow$ (S1 = H)		20	15	
		(10) Data word after $S1\uparrow$ (S0 = H)		20	15	
		(11) Check word after the later of $S1\downarrow$ or $S0\uparrow$		20	15	
		(12) Diagnostic data word after $\overline{LEDBO}\uparrow$ (S1 = L, S0 = H)[‡]		0	0	
1	t_{corr} Correction time (see Figure 1)			70	65	ns

	PARAMETER	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	SN54ALS617 MIN	SN54ALS617 TYP[†]	SN54ALS617 MAX	SN74ALS617 MIN	SN74ALS617 TYP[†]	SN74ALS617 MAX	UNIT
2	t_{pd}	DB and CB	$\overline{ERR}$	S1 = H, S0 = L, R_L = 500 Ω		25			25		ns
		DB	$\overline{ERR}$	S1 = L, S0 = H, R_L = 500 Ω		25			25		
3	t_{pd}	DB and CB	$\overline{MERR}$	S1 = H, S0 = L, R_L = 500 Ω		40			40		ns
				S1 = L, S0 = H, R_L = 500 Ω		40			40		
4	t_{pd}	$S0\downarrow$ and $S1\downarrow$	CB	R_L = 680 Ω		40			40		ns
5	t_{pd}	$\overline{DB}$	$\overline{CB}$	S1 = L, S0 = L, R_L = 680 Ω		40			40		ns
6	t_{pd}	$\overline{LEDBO}\downarrow$	DB	S1 = X, S0 = H, R_L = 680 Ω		25			25		ns
7	t_{pd}	$S1\uparrow$	$\overline{CB}$	S0 = H, R_L = 680 Ω		40			40		ns
8	t_{PLH}	$\overline{OECB}\uparrow$	CB	S1 = X, S0 = H, R_L = 680 Ω		24			24		ns
	t_{PHL}	$\overline{OECB}\downarrow$	CB	S1 = X, S0 = H, R_L = 680 Ω		24			24		ns
9	t_{PLH}	$\overline{OEB0}$ and $\overline{OEB1}\uparrow$	DB	S1 = X, S0 = H, R_L = 680 Ω		24			24		ns
	t_{PHL}	$\overline{OEB0}$ and $\overline{OEB1}\downarrow$	DB	S1 = X, S0 = H, R_L = 680 Ω		24			24		ns

Figure 5.1.3. Timing Specifications for the EDAC chip

Core Constraint and the First Solution:

The various input-to-output paths, in terms of the generic names, are enumerated in figure 5.1.2 by using the path enumeration algorithm developed in chapter 4. Due to the restriction on the generic names, as defined in chapter 4, it is very easy to identify the paths in figure 5.1.2 with the PMG shown in figure 5.1.1. For the core and soft constraint formulation, it should be noted that the "<" and ">" constraints actually represent the "≤" and "≥" constraints. This terminology is used since the "≤" and the "≥" symbols cannot be used with standard keyboards.

A *chip modeler's* viewpoint is considered in modeling this EDAC chip and thus the core constraints are based on the manufacturer's specifications indicated in the figure 5.1.3. The paths are numbered so as to show their correspondence with the timing specification in the data sheets. Only a few of the non-restrictive constraints, for which an over-estimated upper limiting value is provided, are shown in the core constraint formulation. Since the upper limiting value is much higher than the values that are possible, these non-restrictive paths do not play a role in the determination of the solution.

The solution obtained by applying TIMESPEC to the EDAC chip is now examined. The solution at the END OF STEP1 (flowchart in figure 3) is indicated under column SOL1 in figure 5.1.5. No contradictory constraints were detected thus indicating a consistent core constraint formulation. As expected the solution obtained is of no practical application since most of the delay values obtained are zero. In order to obtain a realistic solution, the technology dependant soft constraints are formulated.

EDAC Soft Constraints		
BUFFER_2__OEBOB_DB	=	12
BUFFER_2__CDBL_DB	=	10
LATCH_2__DB_DBL	=	10
LATCH_2__THREEB_DBL	=	12
LATCH_3__LEDBOB_CDBL	=	12
LATCH_3__CDB_CDBL	=	10
ERR_COR2__DBL_CDB	>	10
ERR_COR2__EBIT_CDB	>	10
CB_GEN__DBL_CBG	=	5
MUX__CBL_CBM	=	10
MUX__ONE_CBM	=	14
MUX__SYND_CBM	=	10
BUFFER_1__OECBB_CB	=	12
BUFFER_1__CBM_CB	=	10
BIT_ERR__THREE_EBIT	>	10
BIT_ERR__SYND_EBIT	>	10
ERR_DEC__SYND_MERRB	>	20
ERR_DEC__ZEROB_MERRB	>	20
ERR_DEC__ZEROB_ERRB	>	10
ERR_DEC__SYND_ERRB	>	10
SYND_GEN__CBG_SYND	=	5
SYND_GEN__CBZ_SYND	=	5
LATCH_1__CB_CBL	=	10
LATCH_1__TWO_CBL	=	12
GEBOX__CBL_CBZ	>	3
GEBOX__ZERO_CBZ	>	3
DECODER__S_ZERO	=	10
DECODER__S_ZEROB	=	10
DECODER__S_ONE	=	10
DECODER__S_TWO	=	10
DECODER__S_THREE	=	10
DECODER__S_THREEB	=	10

Figure 5.1.4.

GENERIC NAME	SOL1	SOL2
BUFFER_2__OEBOB_DB	0	12
BUFFER_2__CDBL_DB	26	12
LATCH_2__DB_DBL	0	8.2
LATCH_2__THREEB_DBL	0	12
LATCH_3__LEDBOB_CDBL	0	13.9
LATCH_3__CDB_CDBL	39	12.1
ERR_COR2__DBL_CDB	0	10
ERR_COR2__EBIT_CDB	0	12.6
CB_GEN__DBL_CBG	0	5
MUX__CBL_CBM	0	10
MUX__ONE_CBM	0	16.1
MUX__SYND_CBM	0	10
BUFFER_1__OECBB_CB	0	12
BUFFER_1__CBM_CB	40	11.8
BIT_ERR__THREE_EBIT	0	10
BIT_ERR__SYND_EBIT	0	10
ERR_DEC__SYND_MERRB	40	21.8
ERR_DEC__ZEROB_MERRB	0	20
ERR_DEC__ZEROB_ERRB	0	10
ERR_DEC__SYND_ERRB	26	7.8
SYND_GEN__CBG_SYND	0	5
SYND_GEN__CBZ_SYND	0	5.2
LATCH_1__CB_CBL	0	10
LATCH_1__TWO_CBL	0	12
GEBOX__CBL_CBZ	0	3
GEBOX__ZERO_CBZ	0	3
DECODER__S_ZERO	0	10
DECODER__S_ZEROB	0	10
DECODER__S_ONE	0	12.1
DECODER__S_TWO	0	10
DECODER__S_THREE	0	10
DECODER__S_THREEB	0	10

Figure 5.1.5. TIMESPEC solutions for the EDAC chip

Soft Constraints and the Second Solution:

Since a *chip modeler's* viewpoint is considered in building the VHDL model for the EDAC chip, the soft constraints are the typical values corresponding to the typical blocks for the technology under consideration. The Texas Instruments EDAC chip (SN74ALS617) is from the TTL family of Advanced Low power Schottky (ALS) chips. For this chip, the soft constraints are obtained by surveying various smaller Texas Instruments chips (Latch, Tri-state Buffer etc.) from the above mentioned family of integrated circuits. The values used are slightly less than those available for these surveyed chips in order to account for the I/O pad and wire delays which are not included when these functionalities are the primitives, internal to the chip under consideration.

The standard blocks for the EDAC chip are: Mux, Decoder, Latch and the Tri-state Buffer and typical delay values, obtained as explained above, are assigned to these blocks (figure 5.1.5). The knowledge of the functionality of some blocks makes it possible to predict their implementation and consequently the associated delay values. For example, the functionality of the SYN_GEN and CB_GEN blocks is nothing but an XOR operation and a delay of 5 ns (typical for the TTL-ALS chips) is assigned to these two blocks. The behavior of the remaining blocks is also known, but since these blocks are much more complex, it is not possible to predict the implementation and hence the associated delay values for these blocks cannot be estimated. A **soft** lower limit is placed on these blocks depending on the complexity of the functionality. The soft constraint formulation is shown in figure 5.1.4.

The solution obtained after the addition of the soft constraints is indicated under column SOL2 of figure 5.1.4. It is seen that the TIMESPEC algorithm (CURVE FITTING routine in figure 3) was able to provide a very close approximation of the delays estimated in the soft constraints thus indicating the usefulness of this software tool in using the structural timing feature and its associated advantages, mentioned in chapters 1 and 3, for VHDL modeling. The path delays for all the I/O paths in the chip, not just the ones corresponding to the manufacturer's specifications, are enumerated in figure 5.1.6 thus illustrating the additional advantages associated with using TIMESPEC.

```
LATCH_2__DB_DBL + ERR_COR2__DBL_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 42.4
LATCH_2__DB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + BIT_ERR__SYND_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 65
LATCH_2__DB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + MUX__SYND_CBM + BUFFER_1__CBM_CB = 40
LATCH_2__DB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + ERR_DEC__SYND_MERRB = 40
LATCH_2__DB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + ERR_DEC__SYND_ERRB = 26
BUFFER_2__OEBOB_DB = 12
LATCH_3__LEDBOB_CDBL + BUFFER_2__CDBL_DB = 26
BUFFER_1__OECBB_CB = 12
LATCH_1__CB_CBL + MUX__CBL_CBM + BUFFER_1__CBM_CB = 31.79
LATCH_1__CB_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + BIT_ERR__SYND_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 65
LATCH_1__CB_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + ERR_DEC__SYND_MERRB = 40
LATCH_1__CB_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + ERR_DEC__SYND_ERRB = 26
DECODER__S_THREEB + LATCH_2__THREEB_DBL + ERR_COR2__DBL_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 56.19
DECODER__S_THREEB + LATCH_2__THREEB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + BIT_ERR__SYND_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 78.79
DECODER__S_THREEB + LATCH_2__THREEB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + MUX__SYND_CBM + BUFFER_1__CBM_CB = 53.79
DECODER__S_THREEB + LATCH_2__THREEB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + ERR_DEC__SYND_MERRB = 53.79
DECODER__S_THREEB + LATCH_2__THREEB_DBL + CB_GEN__DBL_CBG + SYND_GEN__CBG_SYND + ERR_DEC__SYND_ERRB = 39.79
DECODER__S_THREE + BIT_ERR__THREE_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 56.79
DECODER__S_TWO + LATCH_1__TWO_CBL + MUX__CBL_CBM + BUFFER_1__CBM_CB = 43.79
DECODER__S_TWO + LATCH_1__TWO_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + BIT_ERR__SYND_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 77
DECODER__S_TWO + LATCH_1__TWO_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + MUX__SYND_CBM + BUFFER_1__CBM_CB = 52
DECODER__S_TWO + LATCH_1__TWO_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + ERR_DEC__SYND_MERRB = 52
DECODER__S_TWO + LATCH_1__TWO_CBL + GEBOX__CBL_CBZ + SYND_GEN__CBZ_SYND + ERR_DEC__SYND_ERRB = 38
DECODER__S_ONE + MUX__ONE_CBM + BUFFER_1__CBM_CB = 40
DECODER__S_ZEROB + ERR_DEC__ZEROB_MERRB = 30
DECODER__S_ZEROB + ERR_DEC__ZEROB_ERRB = 20
DECODER__S_ZERO + GEBOX__ZERO_CBZ + SYND_GEN__CBZ_SYND + BIT_ERR__SYND_EBIT + ERR_COR2__EBIT_CDB + LATCH_3__CDB_CDBL + BUFFER_2__CDBL_DB = 65
DECODER__S_ZERO + GEBOX__ZERO_CBZ + SYND_GEN__CBZ_SYND + MUX__SYND_CBM + BUFFER_1__CBM_CB = 40
```

Figure 5.1.6 EDAC I/O path delays

5.2 Arithmetic Logic Unit:

The second chip considered here is the Texas Instruments ALU (SN74AS181A) which is from the Advanced Schottky (AS) TTL family of integrated circuits. For this chip the functionality is represented at the gate level in the data sheets provided by the manufacturer (Appendix). The PMG description for the ALU shown in figure 5.2.1 is obtained by partitioning this gate level description and providing a higher level of abstraction for modeling the chip.

Core Constraints and the First Solution:

The timing specifications for the ALU given in figure 5.2.2 correspond to the PMG description of figure 5.2.1. As in the case of the EDAC chip, the correspondence between the data sheet specifications and the core constraints shown in figure 5.2.3 is indicated by highlighting and numbering the restrictive core constraints. Since the non-restrictive paths do not play a significant role in determining the delay allocation, these paths, although enumerated, are not included in the core constraints shown in figure 5.2.3.

While formulating the core constraints certain ambiguities present in the data sheets were discovered. These ambiguities and the way they are accounted for while formulating the core constraints are now discussed.

- A SUM mode and a DIFF mode are defined in the timing specifications. These modes correspond to the two different paths in the **GenPG** block (figure 5.2.1), and are defined by the **S** inputs.

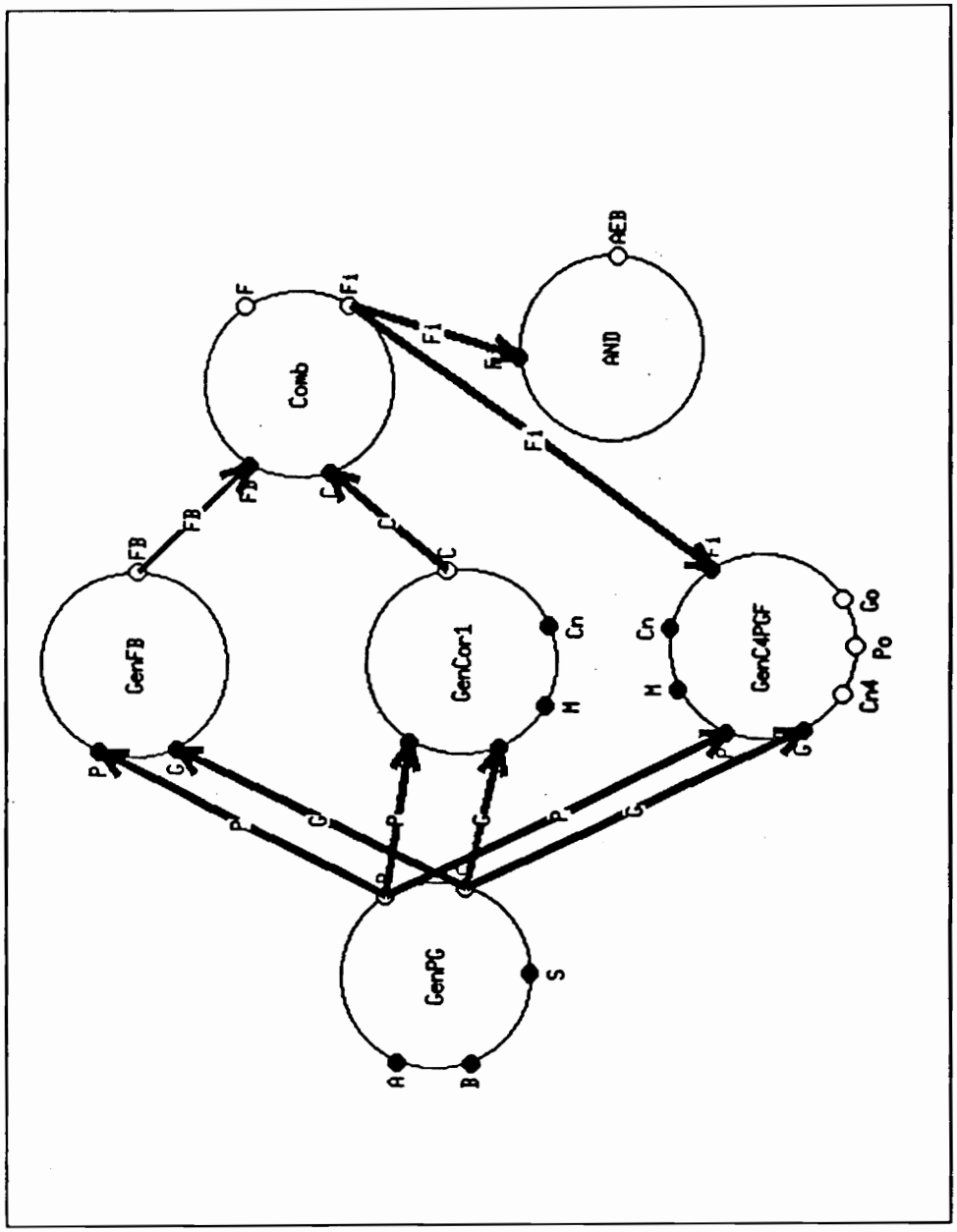


Figure 5.2.1. Process Model Graphs for the ALU

The difference between these two modes is the inversion or non-inversion of the input **B** as seen in the gate level description of the ALU (Appendix). Thus the difference in the delay for the two modes is that of one INVERTER gate delay. In some of the specifications this difference is 1 ns while in some cases there is zero difference (Eg. specifications 5,6 and 7,8), which is clearly a contradiction. In order to correct this ambiguity the following constraints are added to complete the core constraint formulation:

GenPG__B_P_2 - GenPG__B_P_1 = 0.4

GenPG__B_G_2 - GenPG__B_G_1 = 0.4

GenPG__A_P_2 - GenPG__A_P_1 = 0.4

GenPG__A_G_2 - GenPG__A_G_1 = 0.4

Here the generic names with the last digit **2** correspond to the DIFF mode and those with the last digit **1** correspond to the SUM mode. The difference is set to 0.4 ns with the assumption that the delay in one INVERTER gate is 0.4 ns. For a fast chip such as the ALU this assumption is reasonable, and in addition this value falls between the two extremes (1 ns and 0 ns) as specified in the data sheets.

- Also, the constraint numbered **12** in the figure 5.2.2, providing the delay path between **A/B** to **AEB** (figure 5.2.1), has a load condition different from the other specifications. Therefore, instead of using the value provided as the typical delay, the maximum limiting value of 26 ns is used in the core constraints, so as to make this constraint less restrictive.

These user-defined constraints are indicated in italics in figure 5.2.3.

switching characteristics (see Note 1)

	PARAMETER	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	V_{CC} = 5 V, C_L = 15 pF, R_L = 500 Ω (280 Ω for A = B), T_A = 25 °C	V_{CC} = 4.5 V to 5.5 V, C_L = 50 pF (15 pF for A = B), R_L = 500 Ω (280 Ω for A = B), T_A = MIN to MAX						UNIT
					'AS181A 'AS881A	SN54AS181A SN54AS881A			SN74AS181A SN74AS881A			
					MIN TYP† MAX	MIN	TYP†	MAX	MIN	TYP†	MAX	
1	t_{pd}	C_n	C_{n+4}		5	2	7	11	2	7	9	ns
2	t_{pd}	Any $\overline{A}$ or $\overline{B}$	C_{n+4}	M = 0 V, S1 = S2 = 0 V, S0 = S3 = 4.5 V ($\overline{SUM}$ mode)	6	2	8	14	2	8	12	ns
3	t_{pd}	Any $\overline{A}$ or $\overline{B}$	C_{n+4}	M = 0 V, S0 = S3 = 0 V, S1 = S2 = 4.5 V ($\overline{DIFF}$ mode)	7	2	8	20	2	8	16	ns
4	t_{pd}	C_n	Any $\overline{F}$	M = 0 V ($\overline{SUM}$ or $\overline{DIFF}$ mode)	5	3	6	11	3	6	9	ns
5	t_{pd}	Any $\overline{A}$ or $\overline{B}$	$\overline{G}$	M = 0 V, S1 = S2 = 0 V, S0 = S3 = 4.5 V ($\overline{SUM}$ mode)	4	2	5	9	2	5	7	ns
6	t_{pd}	Any $\overline{A}$ or $\overline{B}$	$\overline{G}$	M = 0 V, S0 = S3 = 0 V, S1 = S2 = 4.5 V ($\overline{DIFF}$ mode)	5	2	6	12	2	6	9	ns
7	t_{pd}	Any $\overline{A}$ or $\overline{B}$	$\overline{P}$	M = 0 V, S1 = S2 = 0 V, S0 = S3 = 4.5 V ($\overline{SUM}$ mode)	5	2	6	11	2	6	8	ns
8	t_{pd}	Any $\overline{A}$ or $\overline{B}$	$\overline{P}$	M = 0 V, S0 = S3 = 0 V, S1 = S2 = 4.5 V ($\overline{DIFF}$ mode)	5	2	6	13	2	6	10	ns
9	t_{pd}	$\overline{A}_i$ or $\overline{B}_i$	$\overline{F}_i$	M = 0 V, S1 = S2 = 0 V, S0 = S3 = 4.5 V ($\overline{SUM}$ mode)	5	2	5	11	2	5	8	ns
10	t_{pd}	$\overline{A}_i$ or $\overline{B}_i$	$\overline{F}_i$	M = 0 V, S0 = S1 = 0 V, S1 = S2 = 4.5 V ($\overline{DIFF}$ mode)	5	2	6	12	2	6	10	ns
11	t_{pd}	$\overline{A}_i$ or $\overline{B}_i$	$\overline{F}_i$	M = 4.5 V (LOGIC mode)	6	2	6	16	2	6	11	ns
12	t_{pd}	Any $\overline{A}$ or $\overline{B}$	A = B	M = 0 V, S0 = S3 = 0 V, S1 = S2 = 4.5 V ($\overline{DIFF}$ mode)	12	4	14	26	4	14	21	ns

additional 'AS881A switching characteristics involving status checks (see Note 1)

	PARAMETER	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	V_{CC} = 5 V, C_L = 15 pF, R_L = 500 Ω, T_A = 25 °C	V_{CC} = 4.5 V to 5.5 V, C_L = 50 pF, R_L = 500 Ω, T_A = MIN to MAX						UNIT
					'AS881A	SN54AS881A			SN74AS881A			
					MIN TYP† MAX	MIN	TYP†	MAX	MIN	TYP†	MAX	
13	t_{pd}	Any $\overline{A}$ or $\overline{B}$	$\overline{P}$	C_n = 4.5V, M = 4.5 V, S0 = S3 = 4.5 V, S1 = S2 = 0 V, Equality ($\overline{A}_i = \overline{B}_i$ or $\overline{A}_i \neq \overline{B}_i$)	8	2	10	19	2	10	15	ns
14	t_{pd}	Any $\overline{A}$ or $\overline{B}$	C_{n+4}	C_n = 4.5 V, M = 4.5 V, S0 = S3 = 4.5 V, S1 = S2 = 0 V, Equality ($\overline{A}_i = \overline{B}_i$ or $\overline{A}_i \neq \overline{B}_i$)	10	2	12	24	2	12	18	ns
15	t_{pd}	Any $\overline{A}$ or $\overline{B}$	$\overline{P}$	C_n = 4.5 V, M = 4.5 V, S2 = 4.5 V, S0 = S1 = S3 = 0 V, ($\overline{A}_i = \overline{B}_i$ = H or $\overline{A}_i$ or $\overline{B}_i$ = L)	8	2	10	19	2	10	15	ns
16	t_{pd}	Any $\overline{A}$ or $\overline{B}$	C_{n+4}	C_n = 4.5 V, M = 4.5 V, S2 = 4.5 V, S0 = S1 = S3 = 0 V, ($\overline{A}_i = \overline{B}_i$ = H or $\overline{A}_i$ or $\overline{B}_i$ = L)	‡1	2	13	25	2	13	19	ns

t_{pd} = t_{PHL} or t_{PLH}
‡All typical values are at V_{CC} = 5 V, T_A = 25 °C.
NOTE 1: Load circuit and voltage waveforms are shown in Section 1.

Figure 5.2.2. Timing Specifications for the ALU chip

```
GenPG__B_P_2 - GenPG__B_P_1 = 0.4
GenPG__B_G_2 - GenPG__B_G_1 = 0.4
GenPG__A_P_2 - GenPG__A_P_1 = 0.4
GenPG__A_G_2 - GenPG__A_G_1 = 0.4
1       Gen4PG__Cn_Cn4 = 5
4       GenCor1__Cn_C + Comb__C_F = 5
6       GenPG__B_P_2 + Gen4PG__P_Go = 5
5       GenPG__B_P_1 + Gen4PG__P_Go = 4
8       GenPG__B_P_2 + Gen4PG__P_Po = 5
7       GenPG__B_P_1 + Gen4PG__P_Po = 5
3       GenPG__B_P_2 + Gen4PG__P_Cn4 = 7
2       GenPG__B_P_1 + Gen4PG__P_Cn4 = 6
10      GenPG__B_P_2 + GenCor1__P_C + Comb__C_F = 5
9       GenPG__B_P_1 + GenCor1__P_C + Comb__C_F = 5
15      GenPG__B_P_2 + GenFB__P_FB + Comb__FB_Fi + Gen4PG__Fi_Po = 8
13      GenPG__B_P_1 + GenFB__P_FB + Comb__FB_Fi + Gen4PG__Fi_Po = 8
16      GenPG__B_P_2 + GenFB__P_FB + Comb__FB_Fi + Gen4PG__Fi_Cn4 = 11
14      GenPG__B_P_1 + GenFB__P_FB + Comb__FB_Fi +  Gen4PG__Fi_Cn4 = 10
12      GenPG__B_P_2 + GenFB__P_FB + Comb__FB_Fi + AND__Fi_AEB < 26
11      GenPG__B_P_2 + GenFB__P_FB + Comb__FB_F = 6
11      GenPG__B_P_1 + GenFB__P_FB + Comb__FB_F = 6
6       GenPG__B_G_2 + Gen4PG__G_Go = 5
5       GenPG__B_G_1 + Gen4PG__G_Go = 4
8       GenPG__B_G_2 + Gen4PG__G_Po = 5
7       GenPG__B_G_1 + Gen4PG__G_Po = 5
3       GenPG__B_G_2 + Gen4PG__G_Cn4 = 7
2       GenPG__B_G_1 + Gen4PG__G_Cn4 = 6
10      GenPG__B_G_2 + GenCor1__G_C + Comb__C_F = 5
9       GenPG__B_G_1 + GenCor1__G_C + Comb__C_F = 5
15      GenPG__B_G_2 + GenFB__G_FB + Comb__FB_Fi + Gen4PG__Fi_Po = 8
13      GenPG__B_G_1 + GenFB__G_FB + Comb__FB_Fi + Gen4PG__Fi_Po = 8
16      GenPG__B_G_2 + GenFB__G_FB + Comb__FB_Fi + Gen4PG__Fi_Cn4 = 11
14      GenPG__B_G_1 + GenFB__G_FB + Comb__FB_Fi + Gen4PG__Fi_Cn4 = 10
11      GenPG__B_G_2 + GenFB__G_FB + Comb__FB_F = 6
11      GenPG__B_G_1 + GenFB__G_FB + Comb__FB_F = 6
6       GenPG__A_P_2 + Gen4PG__P_Go = 5
5       GenPG__A_P_1 + Gen4PG__P_Go = 4
8       GenPG__A_P_2 + Gen4PG__P_Po = 5
7       GenPG__A_P_1 + Gen4PG__P_Po = 5
3       GenPG__A_P_2 + Gen4PG__P_Cn4 = 7
2       GenPG__A_P_1 + Gen4PG__P_Cn4 = 6
10      GenPG__A_P_2 + GenCor1__P_C + Comb__C_F = 5
9       GenPG__A_P_1 + GenCor1__P_C + Comb__C_F = 5
15      GenPG__A_P_2 + GenFB__P_FB + Comb__FB_Fi + Gen4PG__Fi_Po = 8
13      GenPG__A_P_1 + GenFB__P_FB + Comb__FB_Fi + Gen4PG__Fi_Po = 8
16      GenPG__A_P_2 + GenFB__P_FB + Comb__FB_Fi + Gen4PG__Fi_Cn4 = 11
14      GenPG__A_P_1 + GenFB__P_FB + Comb__FB_Fi + Gen4PG__Fi_Cn4 = 10
11      GenPG__A_P_2 + GenFB__P_FB + Comb__FB_F = 6
11      GenPG__A_P_1 + GenFB__P_FB + Comb__FB_F = 6
```

Figure 5.2.3 The ALU Core Constraints

The solution obtained at the END OF STEP1 (flowchart) by the application of TIMESPEC is shown under SOL1 of figure 5.2.5. Numerous inconsistencies were detected in the core constraint formulation, thus indicating the erroneous data sheet specification. The CONFLICT REMOVAL routine is thus invoked which modifies the given set of core constraints to obtain a solvable formulation. Here again, the core constraints being an underspecified system, most of the generics have zero value. Thus soft constraints are required to be added to put additional bounds on the solution.

Soft Constraints and the Final Solution:

The gate level description available in the data sheets is utilized to provide the soft constraints for the various blocks in the PMG. For a fast chip such as the ALU, the delays associated with the various gates are very difficult to estimate, without actually having access to a design library used in the design of the chip. This is because the circuit is usually implemented after a series of modifications and minimizations to obtain as low an input-to-output delay as possible. But the delay assignment according to the available gate level description is the best estimate that the *chip modeler* can provide, considering the limited information available from the data sheets. In order to overcome this problem, the delays assigned to the various gates are a very conservative estimate as indicated below:

```
NOT           > 0.4 ns
2 input NAND  > 0.8 ns
2 input NOR   > 0.8 ns
4 input NOR   > 1.0 ns
XOR           > 1.2 ns
3 input AND   > 1.2 ns
4 input AND   > 1.4 ns
5 input AND   > 1.6 ns
6 input AND   > 1.8 ns
```

ALU Soft Constraints		
GenC4PG__Cn_Go	>	2.6
GenC4PG__Cn_Po	>	2.6
GenC4PG__Cn_Cn4	>	3.4
GenC4PG__M_Go	>	3.0
GenC4PG__M_Po	>	3.4
GenC4PG__M_Cn4	>	3.8
GenCor1__Cn_C	>	2.6
Comb__C_Fi	>	1.2
GenPG__B_P_2	>	2.4
GenPG__B_P_1	>	2.0
GenPG__B_G_2	>	2.4
GenPG__B_G_1	>	2.0
GenPG__A_P_2	>	2.4
GenPG__A_P_1	>	2.0
GenPG__A_G_2	>	2.4
GenPG__A_G_1	>	2.0
Comb__C_F	>	1.2
GenC4PG__Fi_Po	>	2.4
GenC4PG__Fi_Cn4	>	3.4
AND__Fi_AEB	>	1.4
GenCor1__M_C	>	3.0
GenC4PG__P_Go	>	2.6
GenC4PG__P_Po	>	2.4
GenC4PG__P_Cn4	>	3.4
GenCor1__P_C	>	2.6
GenFB__P_FB	>	1.2
Comb__FB_Fi	>	1.2
Comb__FB_F	>	1.2
GenC4PG__G_Go	>	2.6
GenC4PG__G_Po	>	2.4
GenC4PG__G_Cn4	>	3.4
GenCor1__G_C	>	2.6
GenFB__G_FB	>	1.2

Figure 5.2.4.

GENERIC NAME	SOL1	SOL2	SOL3	SOL4
GenC4PG__Cn_Go	0.0	2.6	0.0	2.6
GenC4PG__Cn_Po	0.0	2.6	0.0	2.6
GenC4PG__Cn_Cn4	5.0	5.0	5.0	5.0
GenC4PG__M_Go	0.0	3.0	0.0	3.0
GenC4PG__M_Po	0.0	3.4	0.0	3.4
GenC4PG__M_Cn4	0.0	3.8	0.0	3.8
GenCor1__Cn_C	4.4	3.9	5.0	4.1
Comb__C_Fi	0.6	1.2	0.0	1.2
GenPG__B_P_2	4.4	2.3	3.0	2.1
GenPG__B_P_1	4.0	1.9	2.6	1.7
GenPG__B_G_2	4.4	2.3	3.0	2.1
GenPG__B_G_1	4.0	1.9	2.6	1.7
GenPG__A_P_2	4.4	2.3	3.0	2.1
GenPG__A_P_1	4.0	1.9	2.6	1.7
GenPG__A_G_2	4.4	2.3	3.0	2.1
GenPG__A_G_1	4.0	1.9	2.6	1.7
Comb__C_F	0.6	1.1	0.0	0.9
GenC4PG__Fi_Po	0.0	2.4	0.0	2.4
GenC4PG__Fi_Cn4	2.4	4.8	2.4	4.8
AND__Fi_AEB	0.0	1.4	0.0	1.4
GenCor1__M_C	0.0	3.0	0.0	3.0
GenC4PG__P_Go	0.0	2.1	1.4	2.3
GenC4PG__P_Po	0.6	2.7	2.0	2.9
GenC4PG__P_Cn4	2.0	4.1	3.4	4.3
GenCor1__P_C	0.0	1.6	2.0	2.0
GenFB__P_FB	1.6	1.2	3.0	1.2
Comb__FB_Fi	2.0	2.1	2.0	2.3
Comb__FB_F	0.0	2.5	0.0	2.7
GenC4PG__G_Go	2.0	2.1	1.4	2.3
GenC4PG__G_Po	0.6	2.7	2.0	2.9
GenC4PG__G_Cn4	2.0	4.1	3.4	4.3
GenCor1__G_C	0.0	1.6	2.0	2.0
GenFB__G_FB	1.6	1.2	3.0	1.2

Figure 5.2.5. TIMESPEC solutions for the ALU chip

Therefore the soft constraints, for the various blocks in the ALU (figure 5.2.1), based on the above gate delay restrictions are formulated as shown in figure 5.2.4. Once the soft constraints are available, the aim is to obtain a solution which tries to satisfy the restrictions set up in the soft constraints. The soft constraint formulation shown in figure 5.2.4 provides additional bounds on the feasible region defined by the core constraints.

The solution obtained after the addition of the soft constraints is indicated in figure 5.2.5 (column SOL2). Inspite of the low minimum values provided for the various blocks, the delay obtained for the generics **GenCor1__P_C** and **GenCor1__G_C** are a little too low for their implementation. Thus the problem mentioned in section 3.3 is exemplified by the ALU, where TIMESPEC fails to provide a good delay allocation. This is not unexpected since the given core constraints were contradictory to begin with. The CONFLICT REMOVAL routine invoked in the first step of the algorithm modifies the given core constraints and makes available a solvable set of constraints. This procedure does not guarantee a formulation that will provide the desired delay allocation. Thus the technique described in section 3.3 is utilized in obtaining a better core constraint formulation and a better delay allocation.

As mentioned in chapter 3, the user is required to provide limiting values for the generics, identified as having an undesired delay allocation. This limit is included as a core constraint and is based on the implementation limitation, for the technology considered. It is assumed, in this case, that the implementation limit, which is the minimum delay possible to implement the given functionality, requires the following constraints to be added to the core constraint formulation:

GenCor1__P_C > 2

GenCor1__G_C > 2

Thus it is assumed that the 2-level NAND-NOR operation for the functional block **GenCor1** can be implemented only with a resulting delay of at least 2 ns. On solving the new core constraints again, inconsistencies were detected and the CONFLICT REMOVAL routine was invoked to obtain a consistent (solvable) set of constraints. Therefore the given set of constraints are corrected in accordance with the implementation limits hence providing a better core constraint formulation. This solution is indicated in figure 5.2.5, under column SOL3. The solution obtained at this stage does not provide any useful delay allocation since the soft constraints are not added to increase the bounds on the feasible region. The solution obtained after the addition of the soft constraints of figure 5.2.4 is indicated in column SOL4. It is seen that this solution is a much better approximation of the desired delay values specified in the soft constraints.

The path delays associated with all the I/O paths in the chip are enumerated in figure 5.2.6. An examination of these path delays indicates the various modifications made by TIMESPEC, to the original contradictory core constraint formulation, in order to obtain a delay allocation in accordance with the given soft constraints. It is observed that the delays obtained for the paths are within the data sheet specifications.

GenC4PG__Cn_Cn4 = 5.0
GenCor1__Cn_C + Comb__C_Fi + GenC4PG__Fi_Po = 7.7
GenCor1__Cn_C + Comb__C_Fi + GenC4PG__Fi_Cn4 = 10.1
GenCor1__Cn_C + Comb__C_Fi + AND__Fi_AEB = 6.7
GenCor1__Cn_C + Comb__C_F = 5.0
GenCor1__M_C + Comb__C_Fi + GenC4PG__Fi_Po = 6.6
GenCor1__M_C + Comb__C_Fi + GenC4PG__Fi_Cn4 = 9.0
GenCor1__M_C + Comb__C_Fi + AND__Fi_AEB = 5.6
GenCor1__M_C + Comb__C_F = 3.9
GenPG__B_P_2 + GenC4PG__P_Go = 4.4
GenPG__B_P_1 + GenC4PG__P_Go = 4.0
GenPG__B_P_2 + GenC4PG__P_Po = 5.0
GenPG__B_P_1 + GenC4PG__P_Po = 4.6 GenPG__B_P_2 + GenC4PG__P_Cn4 = 6.4
GenPG__B_P_1 + GenC4PG__P_Cn4 = 6.0
GenPG__B_P_2 + GenCor1__P_C + Comb__C_Fi + GenC4PG__Fi_Po = 7.7
GenPG__B_P_1 + GenCor1__P_C + Comb__C_Fi + GenC4PG__Fi_Po = 7.3
GenPG__B_P_2 + GenCor1__P_C + Comb__C_Fi + GenC4PG__Fi_Cn4 = 10.1
GenPG__B_P_1 + GenCor1__P_C + Comb__C_Fi + GenC4PG__Fi_Cn4 = 9.7
GenPG__B_P_2 + GenCor1__P_C + Comb__C_Fi + AND__Fi_AEB = 6.7
GenPG__B_P_1 + GenCor1__P_C + Comb__C_Fi + AND__Fi_AEB = 6.3
GenPG__B_P_2 + GenCor1__P_C + Comb__C_F = 5.0
GenPG__B_P_1 + GenCor1__P_C + Comb__C_F = 4.6
GenPG__B_P_2 + GenFB__P_FB + Comb__FB_Fi + GenC4PG__Fi_Po = 8.0
GenPG__B_P_1 + GenFB__P_FB + Comb__FB_Fi + GenC4PG__Fi_Po = 7.6
GenPG__B_P_2 + GenFB__P_FB + Comb__FB_Fi + GenC4PG__Fi_Cn4 = 10.4
GenPG__B_P_1 + GenFB__P_FB + Comb__FB_Fi + GenC4PG__Fi_Cn4 = 10.0
GenPG__B_P_2 + GenFB__P_FB + Comb__FB_F = 6.0
GenPG__B_P_1 + GenFB__P_FB + Comb__FB_F = 5.6
GenPG__B_G_2 + GenC4PG__G_Go = 4.4
GenPG__B_G_1 + GenC4PG__G_Go = 4.0
GenPG__B_G_2 + GenC4PG__G_Po = 5.0
GenPG__B_G_1 + GenC4PG__G_Po = 4.6
GenPG__B_G_2 + GenC4PG__G_Cn4 = 6.4
GenPG__B_G_1 + GenC4PG__G_Cn4 = 6.0
GenPG__B_G_2 + GenCor1__G_C + Comb__C_Fi + GenC4PG__Fi_Po = 7.7
GenPG__B_G_1 + GenCor1__G_C + Comb__C_Fi + GenC4PG__Fi_Po = 7.3
GenPG__B_G_2 + GenCor1__G_C + Comb__C_Fi + GenC4PG__Fi_Cn4 = 10.1
GenPG__B_G_1 + GenCor1__G_C + Comb__C_Fi + GenC4PG__Fi_Cn4 = 9.7
GenPG__B_G_2 + GenCor1__G_C + Comb__C_Fi + AND__Fi_AEB = 6.7
GenPG__B_G_1 + GenCor1__G_C + Comb__C_Fi + AND__Fi_AEB = 6.3
GenPG__B_G_2 + GenCor1__G_C + Comb__C_F = 5.0
GenPG__B_G_1 + GenCor1__G_C + Comb__C_F = 4.6
GenPG__B_G_2 + GenFB__G_FB + Comb__FB_Fi + GenC4PG__Fi_Po = 8.0
GenPG__B_G_1 + GenFB__G_FB + Comb__FB_Fi + GenC4PG__Fi_Po = 7.6
GenPG__B_G_2 + GenFB__G_FB + Comb__FB_Fi + GenC4PG__Fi_Cn4 = 10.4
GenPG__B_G_1 + GenFB__G_FB + Comb__FB_Fi + GenC4PG__Fi_Cn4 = 10.0

Figure 5.2.6 Path Delays for the ALU

Chapter 6.

Conclusion and Future Development

6.1 Conclusion:

The primary aim the development of TIMESPEC was to incorporate timing in VHDL models using the *imbedded timing* method. This provides models for VLSI chips or digital systems which can be synthesized. TIMESPEC thus provides a very useful tool in the chip design and verification process.

The importance of *imbedded timing*, discussed in chapter 1, promises a substantial role for TIMESPEC in chip design, verification and synthesis. The ability of TIMESPEC, as seen from chapters 3 and 5, to allocate delays, to various functional primitives, that are very close to the desired values (soft constraints) substantially

enhances the design process, especially in a partial top-down design. The results obtained from TIMESPEC define the timing restrictions for the design of the primitives for whom design cells are not available from a cell library. Therefore a design path is defined for the chip designer which will reduce design iterations in order to obtain a chip confirming to all the timing restrictions. Also available, from the TIMESPEC output, are the delays associated with all the input-to-output paths in the chip. Due to the availability of this information early in the design process, the *chip designer* can identify undesired (delays too long or too short) path delays and then TIMESPEC can be utilized, as mentioned in chapter 3, to redesign the various primitives. Due to the *imbedded timing* method of delay representation, the identification of the timing hazards is made possible through the use of simulation or by using timing analyzer/verification tools. The use of these CAD tools in determining the timing problems is possible because of the availibility of timing constraints devoid of ambiguities, made available by TIMESPEC.

The example considered in section 3.3 for an 8-bit Tri-state Register and the two chips considered in chapter 5 illustrate the role of TIMESPEC in assisting a *chip modeler* towards obtaining a VHDL model free of timing inconsistencies. Also the model has a delay allocation which is a very close approximation of the desired timing values. The delay information regarding all the I/O paths in the chip is available, thereby allowing a model which is a better fit in a bigger system model.

The inclusion of TIMESPEC as part of the Modeler's Assitant provides a step towards having a complete environment for VHDL modeling useful in the design and verification of digital hardware. As elaborated in chapter 4, the Modeler's Assistant

graphical environment simplifies the formulation of the input required by TIMESPEC and allows the enumeration of all the paths in the chip along with their delay values. This information, as mentioned time and again, is of immense importance in the design and verification of VLSI chips and other digital systems.

6.2 Future Scope:

With the aim of making available a complete set of CAD tools for VHDL modeling and simulation, there is a wide scope for future enhancements to TIMESPEC in the Modeler's Assistant environment.

There is work in progress, dealing with developing a software tool for automatic hazard detection of timing problems in VHDL models at the gate level. An automatic timing hazard detection for behavioral models is an open problem for research. The delay allocation made available by TIMESPEC can be utilized in such a tool, if and when it is developed, thereby futher increasing the capabilities of the Modeler's Assistant.

For large models with very complex PMG descriptions and a large number of generics, the number of enumerated paths will be extremely large. Hence the selection of the paths corresponding to the timing specifications will be a very tedious task. In order to reduce the effort and time in formulating the core constraints for TIMESPEC, it would be very helpful if these paths could be specified beforehand in the graphical environment describing the PMG. This could be achieved by allowing the user to select the paths by

simply clicking the *mouse* on the various ports in the PMG to define a path. This would necessitate a code conversion to utilize the X-windows programming environment.

The path enumeration is achieved by defining the generic names under specific restrictions (chapter 4). These restrictions are provided to simplify the formulation of the successor matrix describing the digraph representation of the PMG. Removal of these restrictions on the generic names is another modification that needs to be addressed.

For very small path delay limits (<1), the values are required to be scaled up in order to obtain a solution with a lesser number of iterations in the Duoplex Algorithm employed for solving the LPP (chapter 2,3). The scaling of the input data is done manually at present, but an automatic scaling feature can be incorporated in the TIMESPEC routines.

In addition to all these additional features, there is always the possibility of finding a better algorithm for solving the Timing Distribution Problem.

Bibliography

1. J. R. Armstrong, "Chip Level Modeling with VHDL," Prentice Hall, New Jersey, 1989.

2. "IEEE Standard VHDL Language Refernce Manual,"IEEE, New York, 1988.

3. R. Waxman, L. Saunders, H. Carter, "VHDL Links Design, Test and Maintanence," IEEE Spectrum, May 1989, pp.40-44.

4. D. Giles, C. Berking, K. Wacks, "Integrated Functional/Structural Timing for Digital Simulation," Digest of Papers, 1982, International Test Conference, IEEE, New York, New York, pp.153-160.

5. H. Levin, et. al., "Design of a New Test Generation System for Performance Testing of LSI Digital Printed Circuit Boards," Digest of Papers, 1982, International Test Conference, IEEE, New York, New York, pp.541-547.

6. D. R. Coelho, "Follow Simple Rules to Create VHDL Models," Electronic Design, 1990, June 14, pp.65-74.

7. B. Harding, "System Simulation Assures that Chips Play Together," Computer Design, 1989, Aug. 1, pp.70-84.

8. J. R. Armstrong, D. G. Burnette, "A Systematic Application to Chip Level Modeling with VHDL," WESCON 1989, pp.335-338.

9. P. D. Linderman, "Top-Down Design Synthesis Using VHDL," WESCON 1990, pp.382-383.

10. N. D. Dutt, T. Hadley, D. D. Gajski, "An Intermediate Representation for Behavioral Synthesis," 27th ACM/IEEE Design Automation Conference, 1990, pp-14-19.

11. N. D. Dutt, D. D. Gajski, "Designer Controlled Behavioral Synthesis," 26th ACM/IEEE Design Automation Conference, 1989, pp.754-757.

12. R. Damiano, "Logic Synthesis for ASIC's," IEEE Spectrum, 1991, Nov., pp.26-33.

13. K. Bowden, "Design Goals and Implementation Techniques for Time Based Digital Simulation and Hazard Detedtion," Digest of Papers, 1982, International Test Conference, IEEE, New York, New York, pp.147-152.

14. E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," IBM Journal, 1965, pp.90-99.

15. T. M. McWilliams, "Verification of Timing Constraints of Large Digital Systems," Proceedings of the 17th ACM/IEEE Design Automation Conference, 1980, pp.139-147.

16. R. B. Hitchcock, SR. et. al., "Timing Analysis of Computer Hardware," IBM J. Res. Develop. 26, (1982), pp.100-105.

17. B. Singh, "A Parametrized CAD tool for VHDL Model Development with X-Windows," Masters Thesis, Virginia Polytechnic Institute and State University, 1990.

18. S. K. Sherman, "Algorithms for Timing Requirement Analysis and Generation," 25th ACM/IEEE Design Automation Conference, 1988, pp.724-727.

19. H. P. Williams, "Model Building in Mathematical Programming," Wiley, New York, 1985.

20. M. S. Bazaraa, J. J. Jarvis, "Linear Programming and Network Flows," Wiley, New York, 1977.

21. G. Hadley, "Linear Programming," Addison-Wesley, Massachussets, 1962.

22. H. P. Kuenzi, H. G. Tzschach, C. A. Zehnder, "Numerical Methods of Mathematical Optimization," Academic Press, New York, 1971.

23. W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, "Numerical Recipes: The Art of Scientific Computing," Cambridge University Press, New York, 1986.

24. "CMOSN Cell Library".

25. J. Esakov, "Data Structures and Advanced Approach using C," Prentice Hall, New Jersey, 1989.

26. B. W. Kernighan, D. M. Ritchie, "The C Programming Language," Prentice Hall, New Jersey, 1977.

27. J. A. Bondy, U. S. R. Murty,"Graph Theory with Applications," Elsevier North Holland, Inc., New York, 1979.

28. N. Deo, "Graph Theory with Applications to Engineering and Computer Science," Prentice Hall, New Jersey, 1974.

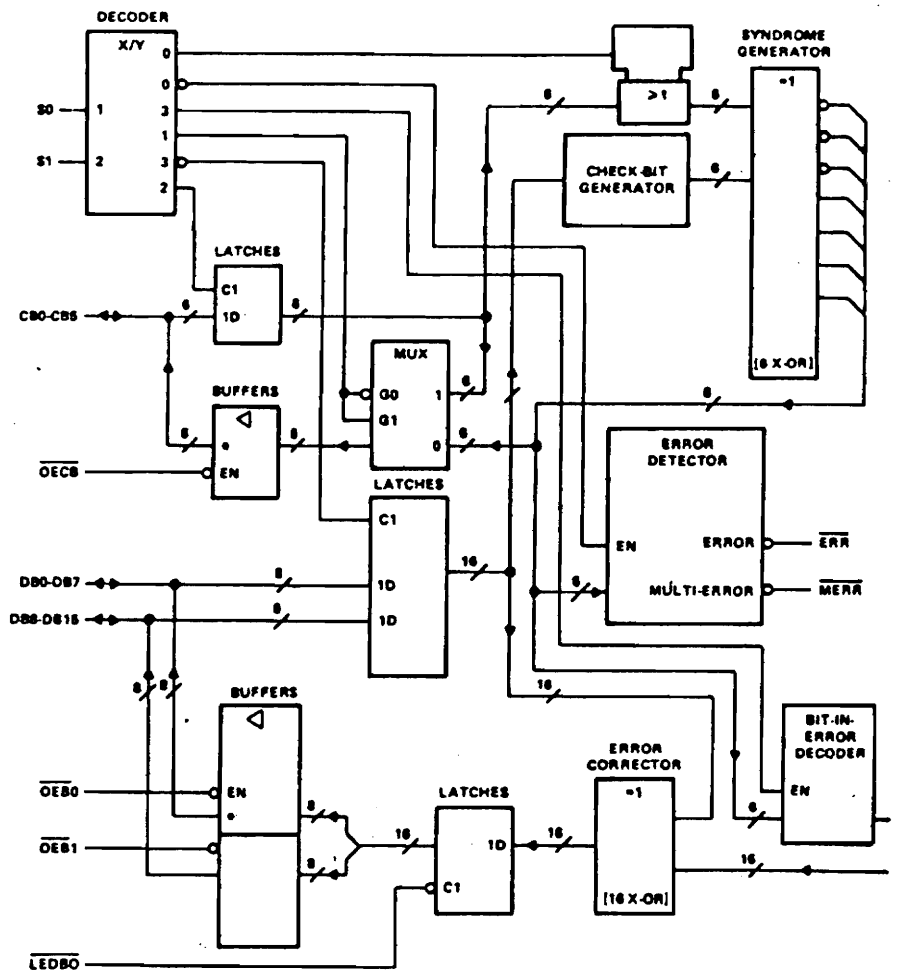
29. T.Asano, S. Sato, "Long Path Enumeration Algorithms for Timing Verification on Large Digital Systems," 5th Quadranneil Int'l Conf. on the Theory and Applications of Graphs with special emphasis on Algorithms and Computer Science, Michigan, June 4-8, 1984.

30. J. R. Armstrong, "Timing Module Specifications," SIGDA, 1989, pp 46-49

Appendix A. Functional Block Diagrams

The functional block diagrams provided by the manufacturer's specification sheets for the Error Detection and Correction chip and the Arithmetic Logic Unit are shown in this appendix.

logic diagram (positive logic)



*'ALS616 has 3 state (▽) check bit and data outputs
'ALS617 has open collector (⏢) check bit and data outputs

TEXAS INSTRUMENTS

Figure A.1 Block Diagram of the EDAC chip

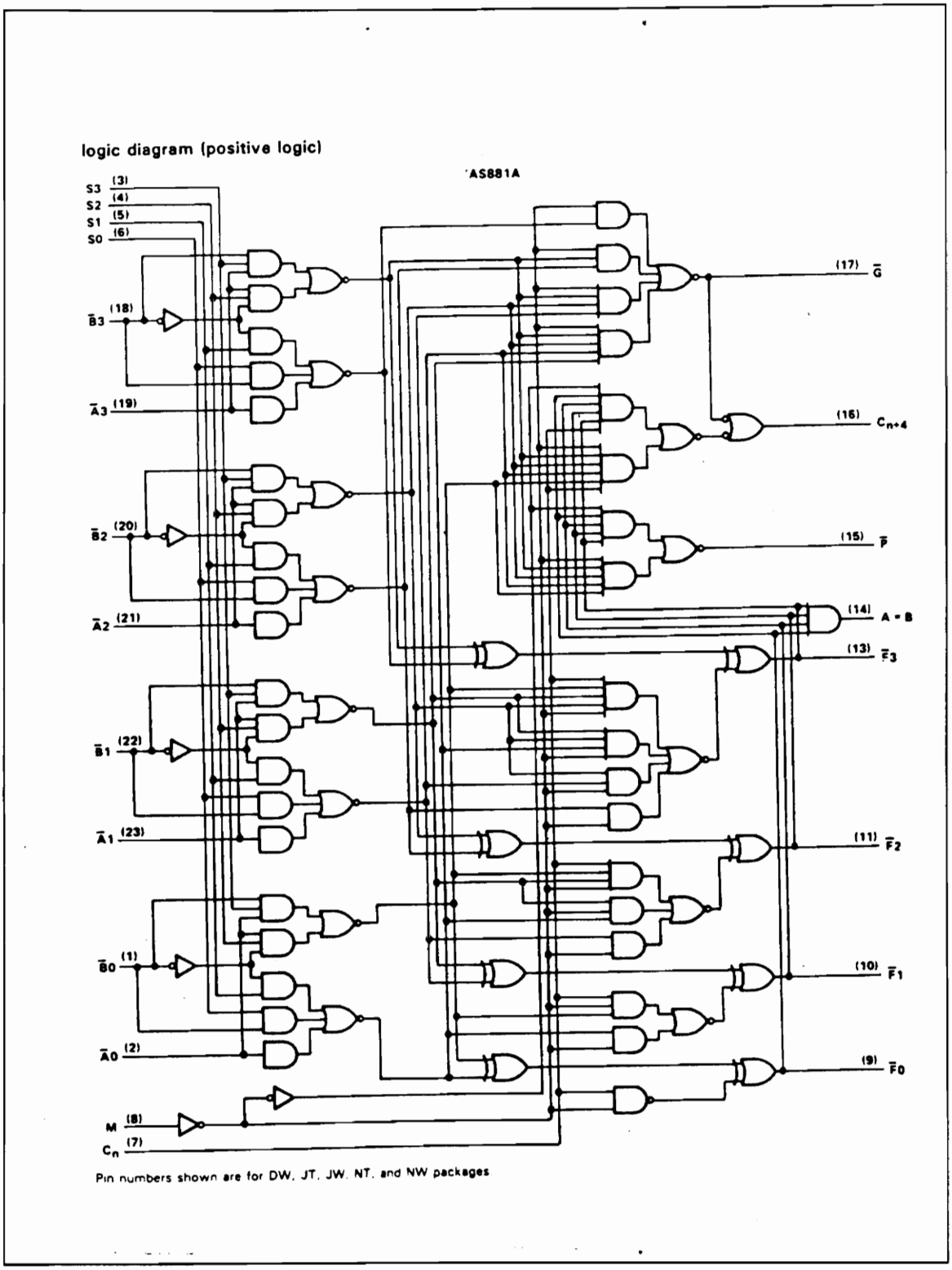


Figure A.2 Gate Level Diagram of the ALU

Appendix B. Users Manual

The Interface:

The X-windows based graphical CAD tool, the Modeler's Assistant, is used to input the PMG description of the digital device under consideration. For information regarding the Modeler's Assistant, the user is refered to reference [17]. Certain restrictions to be followed in the Modeler's Assistant environment are enumerated below:

- Port names used should not contain a '_'.
- The generic names must not be more than 32 characters long and should have the following format:

  xxx__ip_op(_#)

  xxx generic name identifier

  ip process input port

  op process output port

  # In case of multiple generics, the number of the generics associated with the ip-op process port pair.
- Generics should be associated with every ip-op port pair

Once the PMG description is complete, TIMESPEC is invoked.

The Main Menu:

The command "tspec" is entered at the "%" prompt. The main menu is as follows:

```
% tspec

OPTIONS
            1> INPUT THE CONSTRAINTS
            2> EDIT
            3> SOLVE
            4> PATH ENUMERATION
            5> EXIT
ENTER SELECTION NUMBER -->
```

To illustrate the usage of the software tool TIMESPEC, a sample session is illustrated for the 8-bit register example considered in chapter 3.

Core Constraint Formulation:

For automatic generation of all the input-to-output paths in terms of the generic names, the following steps are taken:

```
OPTIONS
            1> INPUT THE CONSTRAINTS
            2> EDIT
            3> SOLVE
            4> PATH ENUMERATION
            5> EXIT
ENTER SELECTION NUMBER --> 4

            IS THE ENUMERATION FOR INPUT OR OUTPUT ?
            Enter 1 for Input and 0 for Output --> 1

            THE PATHS WILL BE ENUMERATED IN A FILE
            Enter the name of the file --> 8bitCOR

            ENTER THE NAME OF THE UNIT --> 8bitREG

            ENTER MAXIMUM LIMITING VALUE --> 100
                                                        (contd.)
```

```
                STRB_DEL + ODEL < 100
                ENDEL + ODEL < 100

                                                    (Return to main menu)
```

The main menu is available again. The "vi" editor is invoked by selecting option 2 at the "ENTER SELECTION NUMBER" prompt to alter the values of the right hand sides of the various constraints according to the timing specifications in order to obtain the core constraint formulation. Additional constraints can also be added as in this case. The core constraint formulation used in this example session is:

$$STRB\_DEL + ODEL > 12$$

$$EN\_DEL + ODEL < 10$$

$$EN\_DEL - STRB\_DEL = 2$$

The core constraints can also be directly input by using the "vi" editor. The constraints must be terminated by "!". The core constraints indicated above are formulated in a file "8bitCOR".

Conflict Removal and First Solution:

Once the core constraints are available the formulation is solved by taking the following steps:

```
OPTIONS
            1> INPUT THE CONSTRAINTS
            2> EDIT
            3> SOLVE
            4> PATH ENUMERATION
            5> EXIT
ENTER SELECTION NUMBER --> 3
                                                    (contd.)
```

```
            ARE THESE THE CORE OR THE CORE+SOFT CONSTRAINTS ?
            Enter 1 for CORE and 0 for CORE+SOFT  --> 1
            Input the filename for the CORE constraints --> 8bitCOR
            Do you wish to optimize any generic ? (y/n) --> n

Non-feasible Solution

The error value row 1 is -2

OPTIONS
     1> Continue....TIMESPEC will modify the row RHS by the error value
     2> Continue without modifying
     3> Return to the main menu
Enter Selection Number --> 1

The error value row 3 is -2

OPTIONS
     1> Continue....TIMESPEC will modify the row RHS by the error value
     2> Continue without modifying
     3> Return to the main menu
Enter Selection Number --> 1

OPTIMAL FEASIBLE SOLUTION
            Enter the name of the solution file --> 8bitSOL1

                  STRB_DEL            0
                  EN_DEL              0
                  ODEL                10

                                                    (Return to main menu)
```

Curve Fitting and Second Solution:

The soft constraints are added using the "vi" editor invoked by selecting option 2 from the main menu. The file name in which the soft constraints are provided is "8bitSOF". The soft constraints used in this session are:

$$STRB\_DEL = 5$$

$$EN\_DEL = 6$$

$$ODEL = 5$$

The complete set of constraints are now solved to obtain a better solution.

```
OPTIONS
            1> INPUT THE CONSTRAINTS
            2> EDIT
            3> SOLVE
            4> PATH ENUMERATION
            5> EXIT
ENTER SELECTION NUMBER --> 3

            ARE THESE THE CORE OR THE CORE+SOFT CONSTRAINTS ?
            Enter 1 for CORE and 0 for CORE+SOFT  --> 0

            Input the filename for the SOFT constraints --> 8bitSOF
            Do you wish to optimize any generic ? (y/n) --> n

            OPTIMAL FEASIBLE SOLUTION
            Enter the name of the solution file --> 8bitSOL2

                  STRB_DEL            4.5
                  EN_DEL              4.5
                  ODEL                5.5

                                                    (Return to main menu)
```

Input-to-output Path Delays:

Now that the desired solution is obtained the delays along all the paths in the given PMG can be easily obtained by following the procedure given below:

```
OPTIONS
            1> INPUT THE CONSTRAINTS
            2> EDIT
            3> SOLVE
            4> PATH ENUMERATION
            5> EXIT
ENTER SELECTION NUMBER --> 4
                                                    (contd.)
```

```
IS THE ENUMERATION FOR INPUT OR OUTPUT ?
Enter 1 for Input and 0 for Output --> 0

THE PATHS WILL BE ENUMERATED IN A FILE
Enter the name of the file --> 8bitPATHS

ENTER THE NAME OF THE UNIT --> 8bitREG
ENTER MAXIMUM LIMITING VALUE --> 100

       STRB_DEL + ODEL = 10
       EN_DEL + ODEL = 10
```

(Return to main menu)

VITA

Ashish Gadagkar was born in New Delhi, India, on the 27th of December, 1967. He received a Bachelor of Engineering degree in Electronic and Telecommunication from the University of Poona at Pune, India, in 1989. He is currently working towards the completion of requirements for the Master of Science degree in Electrical Engineering at the Virginia Polytechnic Institute and State University, Blacksburg, Virginia.