The second part of this book, Chapters 6 to 8, will present several transformational techniques to system synthesis. Typically, such techniques take a behavioral VHDL specification as input and map it first into a naive structural implementation. The implementation is then modified by iterative application of semantic-preserving transformations. These transformations do not change the level of abstraction, rather they explore the design space, looking for near-optimal solutions. When the transformation procedure terminates and a satisfactory solution is found, the design will be mapped into a lower-level description and the synthesis process continues to refine the design into a final implementation.

This chapter will address several basic issues of the transformational techniques. In particular, it will describe a design representation model which is used during the transformation process, the mapping of VHDL to this design representation, some basic transformations, and the selection of transformations during high-level synthesis.

6.1 Design Representation

Central to a transformational approach to synthesis is to have a design representation model that can be used to capture the intermediate results accurately to facilitate analysis of the current solution [Pen87]. This model must be able to
represent designs with different degrees of completion. That is, it should be
able to capture a fairly abstract design with a lot of unspecified information, for
example, a pure behavioral specification. At the same time, it should be able to
describe a very detailed implementation with physical parameters which is
produced by the synthesis process. Thus, it is not necessary or always possible
to have just one design representation model; a lot of synthesis systems actually
use several different representation models during different stages of the
synthesis process.

In most design environments, however, it is very useful to have a unified
design representation which can be used to represent the design at different
degree of completeness. The Extended Timed Petri Net (ETPN) representation
model is such a unified design representation [Pen87, PeKu93]. ETPN consists
of two separate but related parts: the control part and the data path. The data
path is represented as a directed graph with nodes and arcs. The nodes are used
to capture data manipulation and storage units. The arcs represent the connec­
tions of the nodes. The control part, on the other hand, is represented as a timed
Petri net with restricted transition firing rules [PeKu93]. A very simple
example of ETPN is shown in Figure 6.1. Figure 6.1b shows the data path
where each data path node is depicted as a rectangle with a label indicating the
basic operation of the node. The arcs of the data path represent the data flow
between the nodes. Flow of data from one node to another is controlled by the
control signals coming from the control part. The control relation is indicated

![Figure 6.1: An example of ETPN.](image-url)