Chapter 4

Encoding of Symbolic Outputs

The output encoding problem entails finding binary codes for symbolic outputs in a switching function so as to minimize the area or an estimate of the area of the encoded and optimized logic function. In this chapter, we will focus on describing output encoding algorithms that target two-level or Programmable Logic Array (PLA) implementations of logic. As in input encoding targeting two-level logic, the optimization step that follows encoding is one of two-level Boolean minimization.

We begin with an example. An arbitrary output encoding of the function shown in Figure 4.1(b), is shown in Figure 4.2(a). The

![Symbolic covers](image)

**Figure 4.1: Symbolic covers**
symbolic values \{out1, out2, out3, out4, out5\} have been assigned the binary codes \{001, 010, 011, 100, 101\}. The encoded cover is now a multiple-output logic function. These algorithms exploit the sharing between the different outputs so as to produce a minimum cover. It is easy to see that an encoding such as the one in Figure 4.2(b), where each symbolic value corresponds to a separate output, can have no sharing between the outputs. Optimizing the function of Figure 4.2(b) would produce a function with a number of product terms equal to the total number of product terms produced by \textit{disjointly} minimizing each of the on-sets of the symbolic values of Figure 4.1(b). This cardinality is typically far from the minimum cardinality achievable via an encoding that maximally exploits sharing relationships.

In this chapter, we will first describe heuristic techniques that attempt to generate various types of \textit{output constraints} which when satisfied result in maximal sharing during the two-level minimization step. De Micheli in [71] showed that exploiting dominance relationships between the codes assigned to different values of a symbolic output results in some reduction in the overall cover cardinality. This work in described in Section 4.1. Disjunctive relations were shown to be an additional sharing mechanism by Devadas and Newton in [30], and the notion of generalized prime implicants (GPIs) was introduced. GPIs provide a systematic means to exactly minimize the product-term count in the encoded, two-level implementation, using a strategy similar to

\begin{figure}[h]
\centering
\begin{tabular}{cccc}
\hline
0001 & 001 & 0001 & 10000 \\
00-0 & 010 & 00-0 & 01000 \\
0011 & 010 & 0011 & 01000 \\
0100 & 011 & 0100 & 00100 \\
1000 & 011 & 1000 & 00100 \\
1011 & 100 & 1011 & 00010 \\
1111 & 101 & 1111 & 00001 \\
\hline
\end{tabular}
\caption{Possible encodings of the symbolic output}
\end{figure}