TECHNICAL NOTE

AN EFFICIENT ALGORITHM FOR OUTPUT CODING IN PAL-BASED CPLDS

D. Kania

Institute of Electronics, Silesian Technical University
ul. Akademicka 16, 44-100 Gliwice, Poland, kdariusz@zeus.polsl.gliwice.pl

(Received: March 16, 2001 – Accepted: May 15, 2002)

Abstract One of the approaches used to partition inputs consists in modifying and limiting the input set using an external transcoder. This method is strictly related to output coding. This paper presents an optimal output coding in PAL-based programmable transcoders. The algorithm can be used to implement circuits in PAL-based CPLDs.

Key Words Partitioning, Coding, Decomposition, Programmable Logic Devices

1. INTRODUCTION

One of the methods of reducing the number of required module inputs is input coding [1,3]. The issue of coding becomes essential if a programmable PAL structure is to be used as an external transcoder [2]. PAL-based programmable chips have a limited number of terms connected to their OR gates. If a binary coding is used, due to uneven term allocation, only 2k+1 different words can be coded (k is the number of terms). It is of course possible to expand the number of terms and increase the number of code words by using additional outputs. However, there is a need for a different coding which will evenly use the products connected to the OR gates. What kind of coding can use the terms evenly? How many different words can be coded using m-outputs, if k-products are connected to every output? The present paper is an attempt to answer these questions.

2. THEORETICAL BACKGROUND

Let y be defined over the set I={I11,...,I1,I0} (Column A; Table 1). Let us try to find a partition of the function arguments using an 8-input transcoder.

Let \( w_k(I_{s-1},...,I_{p+1},I_p) \) be the number of different words formed by the input variables \( I_{s-1},...,I_{p+1},I_p \). If a set \( X_1 \) exists such that \( X_1 \subseteq I \) and \( w_{X_1} < 2^{(X_1^{-1})} \), then it is possible to limit the number of inputs by using an external transcoder. If we assume that the transcoder has \( n_t \)-inputs, then our search for \( X_1 \) will of course be limited to such subsets for which \( X_1 \leq n_t \).

In our case \( n_t=8 \), so our search will begin with those subsets for which \( X_1=8 \). A subset \( X_1=\left\{ I_{11},I_{10},I_{18},I_{17},I_{15},I_{14},I_{12},I_{11} \right\} \) meeting all the conditions does indeed exist; it is therefore possible to propose the partitioning presented in Table 1 and Figure 1.

We shall now try to implement the transcoder in a programmable structure, in which 3 terms are connected to each output structure (e.g. a CPLD macrocell of the MAX 5000 series of devices by Altera). In what follows it will be assumed, for
simplicity, that the outputs of the programmable transcoder are high active. Under this assumption, a "1" in a code word means that a term was used. It is not possible to implement the transcoder directly in such macrocells. It is necessary to expand the number of terms, which leads to using additional outputs. The resulting transcoder structure and the corresponding file are presented in Figure 2.

Can the individual words be coded better, using fewer outputs?

### 3. OUTPUT CODING IN PAL-BASED PROGRAMMABLE STRUCTURES

Programmable circuits have limited internal resources.

Let \( k \) be the number of code words that can be obtained in a given programmable transcoder, \( m \) - the number of transcoder outputs and \( k \) - the number of terms connected to each output. Optimal coding means using a minimal number of terms while keeping this number evenly distributed among the individual outputs. Let us consider what maximum number \( k_{\text{c}} \) of code words can be obtained using a circuit with given parameters \((m,k)\). To use the terms optimally, we should first use all the combinations "0 of \( m \)", "1 of \( m \)”, "2 of \( m \)” etc. active outputs. Each of the blocks "i of \( m \)” where \( 0 \leq i \leq m \), makes use of a constant number of terms connected to each output (Table 2).

#### TABLE 1. PLA Files Defining The Function \( y \) Before \((y.pla)\) and After Argument Partitioning.

<table>
<thead>
<tr>
<th>( y.pla ) file</th>
<th>( A )</th>
<th>transcoder</th>
<th>( B )</th>
<th>( y ) function circuits</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>.i 12 .o 1 .lib 111</td>
<td>.i 8 .o 4</td>
<td>.lib 111</td>
<td>.i 8 .o 1 .lib 19</td>
<td>.i 8 .o 1 .lib 111.</td>
<td></td>
</tr>
<tr>
<td>.ilb I11,I10,I9,I8,I7,I6,I5,I4,I3,I2,I1,0</td>
<td>.ilb I11,I10,I9,I7,I5,I4,I2,I1</td>
<td>.ilb I11, I9, I3, 0,a3,a2,a1,a0</td>
<td>.ilb I11, I9, I3, 0,a3,a2,a1,a0</td>
<td>.ilb I11, I9, I3, 0,a3,a2,a1,a0</td>
<td></td>
</tr>
<tr>
<td>.ob y</td>
<td>.ob a3,a2,a1,a0</td>
<td>.ob y</td>
<td>.ob y</td>
<td>.ob y</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10110100001 1</td>
<td>00000000 0001</td>
<td>00000001 0010</td>
<td>000000001 1 001010110 1</td>
<td>00000001 1 001010110 1</td>
<td></td>
</tr>
<tr>
<td>001000000000 1 10011100000 1</td>
<td>00000101 0011</td>
<td>00000101 0010</td>
<td>11010001 1 01010111 1</td>
<td>11100100 1 11010000 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
<tr>
<td>000000000000 1 10111111000 1</td>
<td>000000001 0010</td>
<td>00000101 0010</td>
<td>01100001 1 10100111 1</td>
<td>10000000 1 00111010 1</td>
<td></td>
</tr>
</tbody>
</table>

\[ X_1 = \{I_{11}, I_{10}, I_8, I_7, I_5, I_4, I_2, I_1\} \]

\[ X_2 = \{a_3, a_2, a_1, a_0\} \]

**Figure 1.** The example of partitioning the inputs using an external transcoder.
number j such that \( \sum_{i=0}^{j}(m-1) = k \) and \( w_k(X1) < \sum_{i=0}^{j+1}(m) \),
then the coding algorithm is relatively simple. It consists in choosing an arbitrary set of combinations “i of m”; this set will have \( w_k(X1) + 1 \) elements, where \( 0 \leq i \leq j+1 \). The algorithm becomes more complicated if no number j meets the condition \( \sum_{i=0}^{j+1}(m-1) = k \). In this case, we can present the concept of optimal coding differently. In the first step we find a value of j, which meets the inequalities \( \sum_{i=0}^{j+1}(m-1) < k < \sum_{i=0}^{j}(m-1) \). Now a part of the required number of words can be obtained using all the combinations “i of m”, where \( 0 \leq i \leq j \).
This step yields \( \sum_{i=0}^{j}(m) \) different combinations.
When these have been coded, the transcoder still has a certain number \( k_r = k - \sum_{i=0}^{j}(m-1) \) of unused terms at each output. These terms can be used to code more words. With \( k_r \) unused terms connected to every OR gate we can code a maximum number of “j+1 of m” combinations. The total number of terms used by every combination is j+1. With m outputs and \( k_r \) unused terms at every output we can code a maximum of \( \left( \frac{mk_r}{j+1} \right) \) additional words. The total number of possible words is therefore \( \sum_{i=0}^{j}(m) + \left[ \frac{mk_r}{j+1} \right] \). The above allows us to state that
\[
k_i = \sum_{i=0}^{j}(m) + \left[ \frac{mk_r - \sum_{i=0}^{j-1}(m-1)}{j+1} \right] \tag{1}
\]
where j is a number satisfying the inequality
\[
\sum_{i=0}^{j+1}(m-1) < k \leq \sum_{i=0}^{j}(m-1) \tag{2}
\]
Assume that we want to code \( w_k(X1) = 10 \) words using a PAL-based programmable transcoder with \( k=3 \) products per output Table 1 and Figure 1. It is possible to code \( w_k(X1) \) words if the coefficient of the transcoder is \( k_r > w_k(X1) \). From Equation 1, we can determine the number of outputs necessary to code a given number of words. Knowing the values of m and k we can determine the parameter j meeting Equation 2. The creation
of code words occurs in two stages. In the first stage, all "i of m" combinations are generated, where \(0 \leq i \leq j\). In the second stage, the remaining words are coded as "j+1 of m" combinations. The first stage requires no special explanations. The purpose of the second stage is to find a maximum number of "j+1 of m" combinations when \(k_r\) terms per output are available. The optimal PAL-based programmable transcoder structure is presented in Figure 3.

4. CONCLUSIONS

The word-coding algorithm presented above was implemented in a program Decomp assisting the decomposition of combinational circuits. In the present paper the algorithm was described for structures with an equal number of terms connected to each output and with high-active outputs. It is, however, also directly applicable to output coding in circuits with varied number of terms and programmable output type (e.g. 22V10).

5. REFERENCES