Concurrent Resynthesis for Network Optimization

Kuang-Chien Chen
Fujitsu America Inc., USA

Masahiro Fujita
Fujitsu Laboratories Ltd., Japan

Abstract

We present an algorithm C-RENO (Concurrent Resynthesis for Network Optimization) for the optimization of multi-level combinational networks. In C-RENO, a given network is optimized for global optimality by resynthesizing a set of gates concurrently, using other existing gates in the network. C-RENO uses the generalized ratio-set algorithm which enables us to resynthesize complex gates instead of only simple gates (e.g., NAND and NOR), exploring larger reconfiguration space. High quality networks are derived by C-RENO even if no network don’t-cares is used, and results which are not achievable by other methods have been obtained.

1. Introduction

Many logic synthesis systems have been developed in recent years and all of them incorporate optimization algorithms for deriving high quality networks. A common feature in those optimization algorithms is the use of don’t-care conditions for area and/or timing optimization, but different algorithms use different don’t-cares (e.g., satisfiability don’t-cares [1,3] or permissible functions [5,11,13]) and different data structures for function manipulation.

In [8], a network optimization algorithm RENO (REsynthesis for Network Optimization) has been proposed. In RENO, a network is minimized by resynthesizing each gate optimally using other existing gates in the network. It was shown that due to the high reconfiguration ability of the gate resynthesis process, good results are obtained even if no network don’t-cares is used.

Almost all previous approaches including RENO minimize a network by minimizing only one gate at a time, thus obtaining only local optimal results. In this paper, we shall present the C-RENO (Concurrent REsynthesis for Network Optimization) algorithm which is a significant extension of the RENO algorithm. In C-RENO, a set of gates which are compatible to each other (to be explained later) are minimized by a concurrent resynthesis process that leads to more global optimal results. Good common cube factors which are useful for realizing multiple gates are also extracted in this process for further area reduction. The ability to minimize multiple gates concurrently offers many possibilities in the design of minimization algorithms.

We shall briefly discuss the generalized ratio-set algorithm which is used in RENO for resynthesizing a single gate [8]. Then, the C-RENO algorithm for multi-gate minimization is discussed. C-RENO has been implemented using the shared-OBDD data structure [10], and we shall compare it with RENO and the caspf_simplify algorithm in MIS 2.2 [13].

2. Notations and definitions

We consider a loop-free combinational network \( N \) realizing an \( m \)-output function \( F = \{ f_1, f_2, \ldots, f_m \} \) of \( n \) variables \( X = \{ x_1, x_2, \ldots, x_n \} \). \( N \) has \( n \) input terminals, \( m \) output terminals and \( R \) gates \( V = \{ v_1, v_2, \ldots, v_R \} \). For convenience, an input terminal is also denoted by \( v_i \) in later discussions when there is no confusion. Each function \( f_i \) can be incompletely specified, and \( f_{ON}^i \), \( f_{DC}^i \), and \( f_{OFF}^i \) denote the ON-set, DC-set and OFF-set of \( f_i \), respectively. The function realized by gate \( v_i \) with respect to external inputs is denoted by \( f_X(v_i) \).

Definition 1: Let \( f \) and \( g \) be two single-output functions of the same number of variables. The ratio (or generalized cofactor) of \( f \) with respect to \( g \) is \( \emptyset \) if \( f \cdot g = \emptyset \). Otherwise, the ratio is \( f \cup \emptyset \).

Definition 2: A function \( f \) is called a permissible function (abbreviated as PF) for a gate \( v_i \) if the specified functions realized at all output terminals do not change by replacing the function at \( v_i \) by \( f \) [1]. Usually more than one PF exist for \( v_i \), and they are called a set of permissible functions, abbreviated as SPF. SPF of \( v_i \) can be expressed by an incompletely specified function with its ON-, DC- and OFF-sets, denoted by \( G_{ON}(v_i) \), \( G_{DC}(v_i) \) and \( G_{OFF}(v_i) \) respectively.

Two types of SPF’s, i.e., the maximum SPF (abbreviated as MSPF and denoted by \( G_{M} \)) and compatible SPF (abbreviated as CSPF and denoted by \( G_{C} \)), were defined in the Transduction method [11], and minimization algorithms based on MSPF and CSPF have been developed [5,7,11].

3. The generalized ratio-set algorithm

In the gate resynthesis procedures to be discussed, a key question is how a given function is covered by a set of other functions. In other words, given a function \( f \) and a set of functions \( G = \{ g_1, g_2, \ldots, g_k \} \), we want to find out how \( h \) can be covered by the functions in \( G \) (we assume that \( h \) can always be covered by the union of all the functions in \( G \), which is true in the resynthesis procedures to be discussed later). This problem can be solved by the OBDD-based generalized ratio-set algorithm shown in Figure 1.

In Figure 1, the ratio for each function \( g_i \) with respect to \( h \) (i.e., \( \{B_i\} \)) is derived. Then, since \( h \) can be covered by the functions in \( G \), the union of the \( B_i \)'s is tautology. Then, the problem of finding out how \( h \) is covered by the functions in \( G \) becomes the problem of finding out which \( B_i \)'s in obdd_list can be chosen to form tautology, and this task is achieved by Procedure OBDD_tautology_checking. Details of this procedure can be found in [8].
generalized_ratio_set(G, h) {
  /*
   * G: a set of functions \{ g_1, g_2, \cdots, g_k \}.
   * h: a function to be covered by the functions in G.
   */
  /* Initialize */
  obdd_list = essen_fun = table = \emptyset;
  /* Derive the ratios of functions in G w.r.t. h */
  foreach function g_i in G {
    if (\{g_i; h\} \neq \emptyset) {
      B_i = \emptyset;
      obdd_list[k++] = B_i;
    }
  }
  OBDD_tautology_checking(obdd_list, essen_fun, table);
  return table;
}

Figure 1. The OBDD-based generalized ratio-set algorithm.

4. Resynthesis of a single gate

Suppose we are given a multi-level network \( N \) which in general consists of complex gates that express arbitrary functions. Let us consider the problem of resynthesizing a single gate \( v_i \) in \( N \) by using other existing gates in the network. Our goal is to synthesize the function of gate \( v_i \) with a set of good cubes, where each cube consists of literals corresponding to the gates in the network.

Definition 3: Let \( v_i \) be a gate whose function with respect to its immediate predecessors, denoted by \( f_x(v_i) \), consists of cubes \( \{ c_1, c_2, \cdots, c_k \} \). The literals in those cubes are called gate literals (in general these literals can be either complemented or non-complemented, although we shall use complemented literals in later discussions). Let \( c_j \) be a cube in \( f_x(v_i) \), where \( c_j = \overline{v}_{i_j} \overline{v}_{i_{j+1}} \cdots \overline{v}_{i_t} \), for \( t = 1, \cdots, k \). Then the function of \( c_j \) with respect to external inputs is denoted by \( f_x(c_j) \) and is defined as:

\[
  f_x(c_j) = f_x(v_{i_1}) f_x(v_{i_2}) \cdots f_x(v_{i_t}).
\]

Definition 4: Let \( G(v_i) = \{ G^{ON}(v_i), G^{DC}(v_i), G^{OFF}(v_i) \} \) be a SPF of a gate \( v_i \). If \( c_j \neq \overline{v}_{i_j} \overline{v}_{i_{j+1}} \cdots \overline{v}_{i_t} \), then \( c_j \) is a cube consisting of gate literals (\( c_j \) is not necessarily a cube in \( f_x(v_i) \)), then \( c_j \) is called a candidate cube for resynthesizing \( v_i \) if \( f_x(c_j) G^{OFF}(v_i) \subseteq \emptyset \) and \( v_i \notin S(v_i) \), for \( j = t+1, \cdots, t \), where \( S(v_i) \) is the set of successor gates of \( v_i \).

Lemma 1: Let \( v_i \) be a gate where \( f_x(v_i) = \{ c_1, c_2, \cdots, c_k \} \) and \( G(v_i) = \{ G^{ON}(v_i), G^{DC}(v_i), G^{OFF}(v_i) \} \) be a SPF of \( v_i \), then:

(i) \( G^{ON}(v_i) \subseteq f_x(v_i) \) and \( G^{OFF}(v_i) \subseteq f_x(v_i) \).

(ii) \( f_x(c_j) G^{OFF}(v_i) = \emptyset \), for \( t = 1, \cdots, k \).

Property (ii) of Lemma 1 shows that each cube in \( f_x(v_i) \) is a candidate cube for resynthesizing \( v_i \), which is trivially true. In order to resynthesize \( v_i \) by a better set of cubes, however, we need to consider other candidate cubes.

Theorem 1: Let \( G(v_i) = \{ G^{ON}(v_i), G^{DC}(v_i), G^{OFF}(v_i) \} \) be a SPF of a gate \( v_i \). If gate \( v_i \) satisfies

(i) \( v_i \notin S(v_i) \), and

(ii) \( f_x(v_i) G^{OFF}(v_i) = \emptyset \),

then the cube consisting of the single gate literal \( v_i \) and all the cubes in \( f_x(v_i) \) are candidate cubes for resynthesizing \( v_i \).

A set of candidate cubes can be found quickly by Theorem 1. These cubes are good candidates for resynthesizing \( v_i \) since they are also used in realizing other gates. If some candidate cubes are used in resynthesis, then common cube factors can be extracted. We should also note the following:

1. Even if a gate \( v_i \) does not satisfy \( f_x(v_i) G^{OFF}(v_i) = \emptyset \), a cube \( c_j \) in \( f_x(v_i) \) may still be a candidate cube.

2. Both \( f_x(v_i) \) and \( f_x(v_i) \) should be checked for a gate \( v_i \), so that we can utilize cubes in both \( f_x(v_i) \) and \( f_x(v_i) \) for resynthesis.

Procedure single_gate_resynthesis is shown in Figure 2. In this procedure, we first find a set of candidate cubes for resynthesizing \( v_i \) by Theorem 1. Then, the generalized ratio-set algorithm is used to derive a covering table showing how \( G^{ON}(v_i) \) is covered by the candidate cubes. After assigning costs to each column, a minimum cost column-cover is found and cubes selected are used to form a new gate \( v_i \). Notice that when we resynthesize a single gate \( v_i \), the SPF (i.e., \( G(v_i) \)) can be either CSPF or MSPP of \( v_i \), or it can be simply the original function realized by \( v_i \) (i.e., SPF = \( f_x(v_i), \emptyset, f_x(v_i) \)). Details can be found in [8].

single_gate_resynthesis(N, v_i, G(v_i)) {
  /*
   * v_i: a gate in N.
   * G(v_i) = \{ G^{ON}(v_i), G^{DC}(v_i), G^{OFF}(v_i) \}: a SPF of v_i.
   * Resynthesis of a single gate v_i using existing candidate cubes in N. The result is a new gate v_i.
   */
  /* Initialize */
  CCF = \emptyset;
  /* Find candidate cubes and their cube functions. */
  foreach column c_i in table {
    Assign costs to the columns of the table.
    Each column represents a candidate cube in CC.
    column_cost[i] = cost_of_candidate_cube(c_i);
  }
  /* Find how G^{ON}(v_i) is covered by candidate cubes. */
  table = generalized_ratio_set(G^{ON}(v_i));
  /* Assign costs to the columns of the table. */
  each column i in table {
    column_cost[i] = cost_of_candidate_cube(c_i);
  }
  /* Find a minimum cost cover corresponding to a set of cubes that realize G^{ON}(v_i) optimally. */
  min_cover = minimum_cover(table, column_cost);
  v_i = form_gates_from_cover(CC, min_cover);
  return v_i;
}

Figure 2. Procedure for the resynthesis of a single gate.
4. C-RENO: Minimization by concurrent resynthesis of multiple gates

Algorithms which minimize one gate at a time can only obtain sub-optimal results. In this section we discuss an algorithm for resynthesizing multiple gates concurrently using the generalized ratio-set algorithm.

Definition 5: A set of gates \( V = \{v_1, v_2, \ldots, v_k\} \) in a network is a compatible set of gates if for every pair of gates \( v_i \) and \( v_j \) in \( V \), \( v_i \notin S(v_j) \) and \( v_j \notin S(v_i) \).

The problem of concurrent resynthesis of a set of gates for area minimization is as follows: given a set of compatible gates \( V = \{v_1, v_2, \ldots, v_k\} \) and their SPF's which are compatible to each other. Synthesize a set of new gates \( V' = \{v_1', v_2', \ldots, v_k'\} \), such that for each \( v_j \in V \), either \( G_{ON}(v_j) \) or \( G_{OFF}(v_j) \) is realized by gate \( v_j \), and the total number of literals in these new gates is minimum. Notice that:

1. If a cube appear in more than one new gate, then it is preferable to extract it as a common factor to further reduce the total number of literals. This should be considered when we assign costs to the candidate cubes. Because new intermediate gates could be created from common cubes, the number of gates obtained by resynthesis could be larger than \( k \).
2. We only consider the resynthesis of a compatible set of gates, so that a compatible set of SPF's can be derived. For a compatible set of SPF's, we can use either CSPF's or simply the functions realized by these gates.
3. If the OFF-set of the SPF of a gate \( v_j \) is resynthesized and \( v_j \) is replaced by the new gate \( v_j' \), we need to reverse the polarity of the signals fanout out from \( v_j' \).

It is not difficult to see that separately resynthesize each gate in \( V \) optimally is not a solution to this problem. This is because for a set of \( k \) gates, we have \( 2^k \) choices of resynthesizing their ON- or OFF- sets. Also, we should use cubes which are good for multiple gates in order to extract common factors.

The concurrent_resynthesis_of_compatible_gates algorithm is shown in Figure 3. The major steps are explained below:

1. **Phase selection:** for each gate \( v_j \) in \( V \), we need to determine whether \( G_{ON}(v_j) \) (positive phase) or \( G_{OFF}(v_j) \) (negative phase) is synthesized. In Figure 3, this is determined heuristically by comparing the numbers of cubes in \( f_j(v_j) \) and \( f_j(v_j) \). In other words, if \( f_j(v_j) \) has more cubes than the \( f_j(v_j) \), then \( G_{OFF}(v_j) \) is synthesized. Otherwise, \( G_{ON}(v_j) \) is synthesized.

2. **Derivation of an multi-gate covering table:** for each gate \( v_j \) in \( V \), we first find a set of candidate cubes (CC) for covering \( G_{ON}(v_j) \) by Theorem 1. Then, a covering table (table) for \( v_j \) is derived by the generalized ratio-set algorithm. The multi-gate covering table (table_all) is then built by joining the covering tables for all the gates in \( V \).

3. **Cube cost assignment:** a cost is assigned to each candidate cube represented by a column in the multi-gate covering table. Since a cube which is used to realize more than one gate can be extracted as a common factor, the cost of a candidate cube depends not only on the number of literals it has, but also on the number of times it can be used to resynthesize the new gates.

4. **Minimum covering:** after the multi-gate covering table is derived, a minimum cost column cover (min_cover) is obtained. The cubes being selected are used to form the set of new gates \( V' \), and those cubes which appear in more than one gate are extracted as common factors (CF). Finally, gates in \( V' \) are replaced by the set of new gates if the total number of literals is reduced.

```cpp
concurrent_resynthesis_of_compatible_gates(N, V) {
    /*
    V: a compatible set of gates \{v_1, v_2, \ldots, v_k\}.
    This procedure minimizes the gates in V by concurrent resynthesis.
    */
    /* Initialize */
    CC_all = table_all = \emptyset;
    /* Phase selection: */
    for each gate \( v_j \) in V, determine whether \( G_{ON}(v_j) \) or \( G_{OFF}(v_j) \) is to be synthesized.
    /*
    foreach gate \( v_j \) in V {
        if(num_cube(v_j->ON) > num_cube(v_j->OFF)) {
            phase[i] = negative;
            exchange \( G_{ON}(v_j) \) and \( G_{OFF}(v_j) \);
        } else {
            phase[i] = positive;
        }
    }
    */
    /* Derivation of the multi-gate covering table: */
    for each gate \( v_j \) in V, find out how it is covered by the candidate cubes. Build the multi-gate covering table for all gates in V.
    /*
    foreach gate \( v_j \) in V {
        CC = Theorem_1(N, v_j, G(v_j));
        CCF = calculate_cube_functions(CC, \emptyset);
        table = generalized_ratio_set(CCF, \emptyset);
        CC_all = CC_all U CC;
        table_all = table_all U table;
    }
    */
    /* Cube cost assignment: */
    assign costs to the columns of table_all. Take into account that cubes which are used in more than one gate can be extracted as common factors.
    /*
    foreach column \( i \) in covering_table {
        column_cost[i] = cube_cost(CC_all, table_all, i);
    }
    */
    /* Minimum covering: */
    find a minimum cost column cover. Form new intermediate gates for common cube factors and also a new gate \( v_j \) for each gate \( v_j \) in V.
    /*
    min_cover = minimum_cover(table_all, column_cost);
    V' = form_gates_from_cover(CC_all, min_cover);
    CF = extract_common_cube_factors(V');
    */
    /* Replace the original gates if cost is reduced */
    if(cost(V') + cost(CF) < cost(V)) {
        insert each gate in CF into N;
        gate_replacement(V, V', phase);
    }
}
```

Figure 3. Concurrent resynthesis of compatible gates.
The entire C-RENO algorithm is shown in Figure 4. In C-RENO, starting from the outputs of the network, gates on each level are treated as a compatible set of gates (there could be other ways to choose a compatible set of gates), and they are resynthesized concurrently by procedure concurrent_resynthesis_of_compatible_gates. A compatible set of don't-cares can be used in this resynthesis process, although in the current implementation no don't-care is used.

```
C-RENO(N, nodc_or_cspf) {
    // nodc_or_cspf: use no don't-cares or CSPF?
    foreach level i from output side of N {
        V = gates on the i-th level;
        if(nodc_or_cspf = CSPF)
            derive CSPF's for the gates in V;
        concurrent_resynthesis_of_compatible_gates(N, V);
    }
}
```

Figure 4. Algorithm C-RENO.

### 5. Experimental Results

Algorithm C-RENO has been implemented using the shared-OBDD data structures. Table 1 shows the results for several benchmark functions and they are compared with the cspf_simplify algorithm in MIS 2.2 and the RENO algorithm. The initial networks are multi-level benchmarks which are mapped to Library-2 of IWLS'89 by the technology-mapping procedure in MIS 2.2. For Algorithms RENO and C-RENO, the OBDD variable ordering heuristic in [9] is used. The following results have been observed:

1. Due to the global reconfiguration ability of C-RENO, good results are obtained even if no don't-cares are used. The area reduction obtained by C-RENO without using don't-cares is better than the results obtained by cspf_simplify in many cases. It is expected that further improvement can be achieved by using an appropriate don't-care set (e.g., CSPF) and iterative applications of C-RENO.

2. Compared with RENO which resynthesizes one gate at a time, better results are obtained by C-RENO for some functions (e.g., 9symml, alu4, and apex7), showing that concurrent resynthesis can further improve the results by finding a more global optimal solution. Also, the execution time for C-RENO is faster than RENO in most cases. However, because the procedure shown in Figure 3 uses a very simple-minded phase selection heuristic, C-RENO cannot obtain better results than RENO in several cases.

### Table 1. Comparison of Algorithm C-RENO with RENO and cspf_simplify

<table>
<thead>
<tr>
<th>Network</th>
<th>Initial</th>
<th>MIS 2.2</th>
<th>RENO (no-DC)</th>
<th>C-RENO (no-DC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>m</td>
<td>Literals</td>
<td>Time</td>
</tr>
<tr>
<td>9symml</td>
<td>9</td>
<td>1</td>
<td>456</td>
<td>375</td>
</tr>
<tr>
<td>alu2</td>
<td>10</td>
<td>6</td>
<td>786</td>
<td>728</td>
</tr>
<tr>
<td>alu4</td>
<td>14</td>
<td>8</td>
<td>1477</td>
<td>1376</td>
</tr>
<tr>
<td>apex7</td>
<td>49</td>
<td>37</td>
<td>429</td>
<td>386</td>
</tr>
<tr>
<td>b1</td>
<td>3</td>
<td>4</td>
<td>21</td>
<td>16</td>
</tr>
<tr>
<td>c8</td>
<td>28</td>
<td>18</td>
<td>282</td>
<td>210</td>
</tr>
<tr>
<td>cht</td>
<td>47</td>
<td>36</td>
<td>352</td>
<td>337</td>
</tr>
<tr>
<td>cm138a</td>
<td>6</td>
<td>8</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>cm120a</td>
<td>21</td>
<td>1</td>
<td>94</td>
<td>80</td>
</tr>
<tr>
<td>cm151a</td>
<td>12</td>
<td>2</td>
<td>45</td>
<td>38</td>
</tr>
<tr>
<td>cm152a</td>
<td>11</td>
<td>1</td>
<td>39</td>
<td>29</td>
</tr>
<tr>
<td>cm162a</td>
<td>14</td>
<td>5</td>
<td>73</td>
<td>70</td>
</tr>
<tr>
<td>cm163a</td>
<td>16</td>
<td>5</td>
<td>70</td>
<td>67</td>
</tr>
<tr>
<td>cm42a</td>
<td>4</td>
<td>10</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>cm82a</td>
<td>5</td>
<td>3</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>cm85a</td>
<td>11</td>
<td>3</td>
<td>66</td>
<td>61</td>
</tr>
<tr>
<td>cmb</td>
<td>16</td>
<td>4</td>
<td>78</td>
<td>66</td>
</tr>
<tr>
<td>example2</td>
<td>85</td>
<td>66</td>
<td>568</td>
<td>553</td>
</tr>
<tr>
<td>f51m</td>
<td>8</td>
<td>8</td>
<td>253</td>
<td>200</td>
</tr>
<tr>
<td>frg1</td>
<td>28</td>
<td>3</td>
<td>219</td>
<td>216</td>
</tr>
<tr>
<td>lal</td>
<td>26</td>
<td>19</td>
<td>288</td>
<td>240</td>
</tr>
<tr>
<td>majority</td>
<td>15</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>mux</td>
<td>21</td>
<td>1</td>
<td>155</td>
<td>133</td>
</tr>
<tr>
<td>parity</td>
<td>16</td>
<td>1</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>pc1</td>
<td>19</td>
<td>9</td>
<td>127</td>
<td>124</td>
</tr>
<tr>
<td>pcle</td>
<td>27</td>
<td>17</td>
<td>134</td>
<td>131</td>
</tr>
<tr>
<td>set</td>
<td>19</td>
<td>15</td>
<td>228</td>
<td>183</td>
</tr>
</tbody>
</table>

n: number of inputs, m: number of outputs. Run time is obtained in SPARCstation 1.
References


6. Conclusion and Future Plan

We have presented C-RENO for concurrent multi-gate optimization based on the generalized ratio-set algorithm. We feel that many possibilities and interesting problems arise when multiple gates are minimized concurrently. More work needs to be done on the phase selection problem and efficiency need to be improved for large networks. Extension of this approach for Boolean relation minimization is also being investigated.

Table 2. Effect of Phase selection in C-RENO

<table>
<thead>
<tr>
<th>Network</th>
<th>Initial</th>
<th>MIS 2 2</th>
<th>RENO (no-DC)</th>
<th>C-RENO (no-DC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>n</td>
<td>m</td>
<td>Literals</td>
<td>Literals Time</td>
</tr>
<tr>
<td>9ymml</td>
<td>9</td>
<td>1</td>
<td>456</td>
<td>375 22.1</td>
</tr>
<tr>
<td>alu2</td>
<td>10</td>
<td>6</td>
<td>786</td>
<td>728 225.0</td>
</tr>
<tr>
<td>alu4</td>
<td>14</td>
<td>8</td>
<td>1477</td>
<td>1376 1328.9</td>
</tr>
<tr>
<td>apex7</td>
<td>49</td>
<td>37</td>
<td>429</td>
<td>386 21.3</td>
</tr>
<tr>
<td>b1</td>
<td>3</td>
<td>4</td>
<td>21</td>
<td>16 0.3</td>
</tr>
<tr>
<td>c8</td>
<td>28</td>
<td>18</td>
<td>282</td>
<td>210 5.8</td>
</tr>
<tr>
<td>ch1</td>
<td>47</td>
<td>36</td>
<td>352</td>
<td>337 7.8</td>
</tr>
<tr>
<td>cm138a</td>
<td>6</td>
<td>8</td>
<td>35</td>
<td>35 1.2</td>
</tr>
<tr>
<td>cm150a</td>
<td>21</td>
<td>1</td>
<td>94</td>
<td>80 2.5</td>
</tr>
<tr>
<td>cm151a</td>
<td>12</td>
<td>2</td>
<td>45</td>
<td>38 1.0</td>
</tr>
<tr>
<td>cm152a</td>
<td>11</td>
<td>1</td>
<td>39</td>
<td>29 0.4</td>
</tr>
<tr>
<td>cm162a</td>
<td>14</td>
<td>5</td>
<td>73</td>
<td>70 1.4</td>
</tr>
<tr>
<td>cm163a</td>
<td>16</td>
<td>5</td>
<td>70</td>
<td>67 1.0</td>
</tr>
<tr>
<td>cm42a</td>
<td>4</td>
<td>10</td>
<td>35</td>
<td>35 0.8</td>
</tr>
<tr>
<td>cm52a</td>
<td>5</td>
<td>3</td>
<td>32</td>
<td>32 0.6</td>
</tr>
<tr>
<td>cm55a</td>
<td>11</td>
<td>3</td>
<td>66</td>
<td>61 1.2</td>
</tr>
<tr>
<td>cmb</td>
<td>16</td>
<td>4</td>
<td>78</td>
<td>66 1.1</td>
</tr>
<tr>
<td>example2</td>
<td>85</td>
<td>66</td>
<td>568</td>
<td>553 19.3</td>
</tr>
<tr>
<td>f51in</td>
<td>8</td>
<td>8</td>
<td>253</td>
<td>200 8.4</td>
</tr>
<tr>
<td>frgl</td>
<td>28</td>
<td>3</td>
<td>219</td>
<td>216 4.7</td>
</tr>
<tr>
<td>lai</td>
<td>26</td>
<td>19</td>
<td>288</td>
<td>240 5.9</td>
</tr>
<tr>
<td>majority</td>
<td>5</td>
<td>1</td>
<td>16</td>
<td>16 0.2</td>
</tr>
<tr>
<td>mutex</td>
<td>21</td>
<td>15</td>
<td>133</td>
<td>133 4.5</td>
</tr>
<tr>
<td>parity</td>
<td>16</td>
<td>1</td>
<td>60</td>
<td>60 0.5</td>
</tr>
<tr>
<td>pcbe</td>
<td>19</td>
<td>9</td>
<td>127</td>
<td>124 2.2</td>
</tr>
<tr>
<td>pcler8</td>
<td>27</td>
<td>17</td>
<td>134</td>
<td>141 4.5</td>
</tr>
<tr>
<td>set</td>
<td>19</td>
<td>15</td>
<td>228</td>
<td>183 5.0</td>
</tr>
</tbody>
</table>