A Simple Yet Effective Merging Scheme for Prescribed-Skew Clock Routing

Rishi Chaturvedi and Jiang Hu

Department of Electrical Engineering
Texas A&M University, College Station, TX-77843, USA
{rishi, jianghu}@ee.tamu.edu

Abstract

In order to achieve multi-GHz operation frequency for VLSI design, clock networks need to be designed in a very elaborated manner and be able to deliver prescribed useful skews rather than merely zero-skew. Although traditional zero-skew clock routing methods can be extended directly to prescribed skews, they tend to result in excessive wirelength as the differences among delay-targets for clock sinks are neglected. In this paper, we propose the maximum delay-target and minimum merging-cost merging scheme for prescribed-skew clock routing. This scheme is simple yet surprisingly effective on wirelength reduction. Experimental results on benchmark circuits show that our merging scheme yields 53%-61% wirelength reduction compared to traditional clock routing methods.

1. Introduction

In synchronous VLSI circuits, since the pace of almost every data transfer is coordinated by clock signals, the quality of clock networks has vital influence to the circuit timing performance. Moreover, as clock network is perhaps the largest interconnect net, a clock network with excessive wirelength may lead to severe problems on power consumption, power/ground supply noise, process variations and thermal issues. Thus, clock network with elaborated timing characteristics and minimal wirelength is crucial for multi-GHz VLSI designs.

A clock network is an interconnect system that delivers a clock signal from a driver to a set of sinks which is normally a set of storage elements such as flip-flops. A clock network design usually starts with specifying delay-targets from the driver to each sink. Since clock skew, which is the difference of delay between clock sinks, is more important than the delay itself, this specifying process is often called skew scheduling. The basic objective of skew scheduling is to minimize the clock period subject to setup-time and hold-time constraints [4,6]. After skew scheduling, the delay-targets or skew specifications are achieved through clock network routing. A common structure for clock network is a routing tree where the clock driver is the root node and the clock sinks are the leaf nodes. Without loss of generality, we can conceive the clock tree routing as a process that recursively merges a set of subtrees in a bottom-up fashion. Initially, each clock sink is a subtree and then the subtrees are merged in pairs. A pair of subtrees is merged to form a new subtree whose root is the merging node. This procedure proceeds till there is only one subtree left and this single subtree is connected to the driver directly. There are two major decision-makings in this clock tree routing process: (i) merging selection that tells which subtrees should be merged together; (ii) finding the locations of the merging nodes. The merging scheme can be extracted out and performed in advance to construct an abstract tree. The internal nodes in the abstract tree correspond to the merging nodes without specifying locations. The remaining task of deciding the locations for the internal/merging nodes is called layout embedding. Abstract tree construction and layout embedding can be performed either separately or in an integrated manner. Examples of abstract tree and embedding are shown in Figure 1. A good survey on clock network design is provided in [7].

The decision-makings on merging selection and layout embedding depend on both the optimization objectives and circuit properties such as clock sink load distributions. Most of previous works on clock network design attempt to minimize clock skew or obtain zero skew, because the skew is a lower bound for clock period time [1]. In this scenario, a more precise definition of skew is the maximum delay difference among all clock sinks. An early influential zero-skew routing technique is H-tree [1], which is a simple and regular structure. The H-tree technique is designed for and limited to evenly distributed clock sinks with similar load capacitances; nevertheless, it is still popular for top-level clock network design. Later on, top-down recursive partitioning [8] and bottom-up
When delay-targets for four sinks $t_4 > t_3 > t_2 > t_1$, traditional merging scheme may result in abstract tree in (a) and embedding in (b) with wire snakings. A different abstract tree in (c) and its layout embedding in (d) may yield less wirelength.

recursive matching method [9] are proposed for general sink load distributions. However, both methods emphasize on load balancing without evaluating actual delay. In [11], Tsay introduced an Elmore delay based layout embedding technique that can achieve exact zero skew for any given abstract tree. In order to further reduce wirelength, the DME (Deferred Merge Embedding) algorithm was developed in [2] according to the observation that there are multiple locations for a merging node to satisfy skew specifications. Instead of committing a merging node to particular location immediately, DME identifies and maintains merging segment for each merging node in a bottom-up tree traversal. After merging segments for all merging nodes are found, a top-down tree traversal is conducted to choose one location on each merging segment such that the total wirelength is minimized. Both Tsay’s embedding and DME embedding technique can be applied to achieve any non-zero skew as well. For any given abstract tree and Elmore delay model, DME is a very mature layout embedding technique to obtain any skew specifications with minimal wirelength and becomes a basis for many subsequent clock routing works [3, 5, 10, 12].

For merging schemes, a widely accepted conclusion is that a subtree should be merged with its nearest neighboring subtree to save wirelength. Even for this common implicit rule, there are several variations of the merging schemes. For early VLSI technologies, interconnect delay is dominated by capacitive load, thus many previous merging schemes [2, 8, 9] sought for a balanced abstract tree to facilitate zero skew. However, Edahiro noted in [5] that sometimes an unbalanced abstract tree might yield less wirelength even for zero-skew clock routing. This is due to the fact that distributed RC delay started to dominate interconnect delay and merely balancing capacitive load is not adequate. This is particularly true when sink distribution is far from uniform. In [5], the merging selection is integrated with DME embedding. At each step, Edahiro chose a subset (generally less than a half) of subtrees to be merged in pairs in contrast to choosing all subtrees in other works. The work of [5] reported so far the best wirelength for zero-skew routing.

It was observed long time ago that certain non-zero skew could be utilized to improve clock frequency [1, 6]. These are called useful skews to be distinguished from any unwanted non-zero skews. In this scenario, a skew refers to the delay difference between a certain sink pair. A few works [10, 12] integrate skew scheduling with clock routing to exploit the useful skews. Starting with a zero-skew routing tree, the work of [12] performs merging segment perturbation and gate sizing to minimize power consumption subject to setup-time and hold-time constraints for a fixed clock period time. In [10], an incremental scheduling algorithm is proposed and combined with the DME embedding for a given abstract tree.

In practical design flows, skew scheduling is often carried out individually ahead of clock routing. We call the useful skews specified through the scheduling algorithms as prescribed skews. Even though there are plenty of works on skew scheduling [4, 6] generating useful skew specifications, work on clock routing delivering the prescribed skews is hardly seen. Perhaps this is due to the misconception that existing zero-skew routing techniques can be applied to prescribed skew directly. Indeed, the layout embedding techniques originally designed for zero-skew [2,11] can be adopted directly to achieve prescribed skews. However, zero-skew driven merging schemes do not necessarily work well for prescribed-skew routing. In fact, we discover that huge wirelength is generated through traditional merging scheme in which only subtree spatial proximity is considered while delay-target differences due to prescribed skew are ignored. This is especially true when the differences among delay-targets are large so that a lot of wire snakings [11] are incurred. The example in Figure 1 illustrates that different merging schemes (abstract trees) may provide different wirelength for prescribed-skew routing.

In this work, we propose the maximum delay-target and minimum merging-cost ordered merging scheme and integrate it with DME embedding for general prescribed-skew clock routing. This merging scheme is simple yet
surprisingly effective on minimizing wirelength. The additional computation cost is very limited in practice. We compared our routing method with traditional clock routing method [5] on benchmark circuits and found that our merging scheme yields 53%-61% wirelength reduction.

The rest of this paper is organized as follows. The problem formulation is introduced in Section 2. Our proposed merging scheme is described in Section 3. Experimental results are shown in Section 4 and conclusions are given in Section 5.

2. Preliminary

Same as other clock routing works, we adopt the Elmore delay model for delay computation. The problem we will solve is formally stated as follows.

Minimum Wirelength Prescribed Skew Routing Tree Problem: Given a set of clock sinks $V = \{v_1, v_2, \ldots, v_n\}$, load capacitance $C_i$ for each sink $v_i \in V$, skew specifications $q_{i,j}$ for every pair of sinks $v_i, v_j \in V$, find a rooted Steiner tree with clock sinks as leaf nodes such that the total wirelength is minimized and the skew specification $q_{i,j} = d_{i,j} - d_j$ is satisfied for root-to-sink delay $d_i$ and $d_j$ of any sink pair $v_i, v_j \in V$.

The skew specifications can also be expressed through root-to-sink delay-target $t_i$ for each sink $v_i \in V$, as long as $q_{i,j} = t_i - t_j \forall v_i, v_j \in V$ is satisfied. It does not matter whether or not the delay $d_i$ of sink $v_i$ in a clock tree is equal to its delay-target $t_i$. The skew specifications can be satisfied whenever we can find a single constant $C$ such that $t_i = d_i + C$ is true for every sink $v_i \in V$. Please note that the ultimate goal is to achieve the skew specifications and the concept of delay-target is for the convenience of computation and description. The zero-skew requirement can be obtained by letting $t_1 = t_2 = \ldots = t_n$.

Now we generalize the concept of delay-target to include subtrees. Let $T_i$ denote a subtree rooted at node $v_i$. This subtree can be characterized by delay-target $t_i$ and downstream capacitance $C_i$ at its root $v_i$. If $v_i$ is a sink node, its delay-target $t_i$ is given. If $v_i$ is a merging node, its delay-target $t_i$ can be computed recursively as follows. If we merge subtrees $T_j$ and $T_k$ at merging node $v_i$ as shown in Figure 2(a), let the wirelength from $v_i$ to $v_j$ and $v_k$ be $l_{i,j}$ and $l_{i,k}$, respectively. Then the delay from $v_i$ to $v_j$ and $v_k$ are:

$$d_{i,j} = \frac{1}{2} r_{i,j} C_{j} + r l_{i,j} C_{j}$$

$$d_{i,k} = \frac{1}{2} r_{i,k} C_{k} + r l_{i,k} C_{k}$$

where $r$ and $c$ are wire resistance and capacitance per unit length, respectively. In order to meet skew constraints, these delays have to satisfy the following equality:

$$d_{i,j} - d_{i,k} = t_j - t_k$$

Then the delay-target $t_i$ can be obtained by rearranging the above equality as

$$t_i = t_j - d_{i,j} = t_k - d_{i,k}$$

Since the delay-targets are propagated bottom-up according to the above equation, the skew specifications can be enforced by only considering Equation (2) without checking delays at sink/leaf nodes. The downstream capacitance $C_i$ can be obtained directly as $C_i = C_j + C_k + c l_{i,j} + c l_{i,k}$.

The minimum wirelength required for the merging is the Manhattan distance $l_{j,k}$ between $v_j$ and $v_k$. The wirelength from $v_i$ to $v_j$ and $v_k$ need to satisfy $l_{j,k} = l_{i,j} + l_{i,k}$. When there is great difference between delay-targets, for example, when $t_j$ is much greater than $t_k$, we have to let $l_{i,k} = 0$ and let $l_{i,j} = l_{j,k}$ to ensure that the constraint of Equation (2) is met. The actual wirelength of $l_{i,j}$ can be obtained by solving the following equation.

$$\frac{1}{2} r_{i,j} C_{j} + r l_{i,j} C_{j} = t_j - t_k$$

The method of using wirelength greater than $l_{j,k}$ is called wire snaking [11] which is demonstrated in Figure 2(b).

Since the top-level framework of our prescribed-skew routing algorithm is very similar to Edahiro’s zero-skew routing algorithm in [5], we briefly review the basic version of Edahiro’s algorithm. Edahiro integrated a nearest-pair merging scheme with DME embedding into a two-phase approach. The first phase is a bottom-up recursive subtree merging process, which constructs the abstract tree and finds the merging segments [2] for each merging node. In each step, only one pair of subtrees (initially clock sinks), which have the minimum distance from each other, is selected to be merged. The nearest neighbor is found using Delaunay triangulation. After a merging, the two subtrees are replaced by the newly created subtree. This process is repeated recursively till there is only one subtree left. The second phase is a top-down tree traversal that identifies one location on each merging segment such that the total wirelength is
minimized. This phase is exactly the same as the top-down phase in DME algorithm [2]. Edahiro also presented a clustering based speed-up technique in the same paper [5].

3. Algorithm

Although the top-level framework of our algorithm is almost the same as Edahiro’s algorithm, which is reviewed in previous section, our merging scheme is designed for a more general prescribed-skel routing and is quite different from Edahiro’s algorithm or other previous works. Therefore, the description will be focused on our merging scheme.

Most of previous bottom-up merging schemes [5, 9] choose the subtree pair with the minimum distance between their roots and merge them first. Their attentions are only at subtree spatial proximities, since delay-targets are identical for all sinks in zero-skel routing. It is shown in the previous section that great difference between delay-targets may cause wire snakings, thus traditional merging schemes tend to result in excessive wirelength because of their neglect of the delay-target differences. We demonstrate this problem through the example in Figure 1. Assume that the given delay-targets are quite different from each other and they follow the inequality \( t_1 < t_2 < t_3 < t_4 \), especially \( t_3 \) and \( t_4 \) are much greater than \( t_1 \) and \( t_2 \). We merge \( v_1 \) to \( v_2 \) first, since their distance is the smallest among all sink pairs. Because \( t_2 \) is significantly greater than \( t_1 \), it is quite likely that a wire snaking occurs when we merge \( v_1 \) and \( v_2 \) to \( v_5 \) as shown in Figure 1(b). Similarly, \( v_3 \) is merged with \( v_4 \) at \( v_6 \). Since \( t_3 \) and \( t_4 \) are much greater than \( t_1 \) and \( t_2 \), it is quite possible that \( t_6 \) is much greater than \( t_5 \) and another wire snaking results from merging subtree \( T_5 \) and \( T_6 \) at \( v_7 \).

Since wire snaking is more likely to happen when the difference of delay-targets between two subtrees is large, it can be reduced if we choose a merging order that can reduce the delay-target differences among all subtrees. According to Equation (3), the delay-target of the newly created subtree is always smaller than the delay-targets of the two subtrees it is merged from. Therefore, if we choose to merge the subtree with the maximum delay-target first, the overall delay-target differences among subtrees will be reduced. We can analogize the set of subtrees as a group of runners. We give a higher priority to the runner who is lagging behind so that the first runner and the last runner are closer to each other. According to Equation (1), if \( C_j \) is much greater than \( C_k \), it is easier to achieve great \( d_{i,j} - d_{i,k} \) without wire snaking. When the maximum delay-target tree is merged first, the newly created subtree from this merging not only has a smaller delay-target but also has a greater load capacitance that makes the matching to other small delay-target subtrees easier. Therefore, the maximum delay-target ordered merging can reduce the chance of wire snaking by decreasing delay-target imbalance and increasing load imbalance that is coherent with the delay-target imbalance.

**Procedure:** FindSubtreesToBeMerged(\( T \))

<table>
<thead>
<tr>
<th>Input:</th>
<th>A set of subtrees ( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Two subtrees to be merged</td>
</tr>
</tbody>
</table>

1. \( T_i \leftarrow \) subtree with the maximum delay-target in \( T \)
2. \( \text{minCost} \leftarrow \infty \)
3. For each subtree \( T_j \in T \backslash T_i \)
4. \( \text{cost} \leftarrow \) merging cost between \( T_i \) and \( T_j \)
5. If \( \text{cost} < \text{minCost} \)
6. \( \text{minCost} \leftarrow \text{cost}, T_k \leftarrow T_j \)
7. Return \( T_i \) and \( T_k \)

Fig. 3. Algorithm of the merging selection scheme.

We further illustrate the advantage of this maximum delay-target ordered merging through the example in Figure 1. In Figure 1(d), we first merge \( v_3 \) and \( v_4 \) to obtain subtree \( T_6 \) rooted at \( v_6 \) as \( v_4 \) has the maximum delay-target. Since \( t_4 \) and \( t_3 \) are much greater than \( t_2 \) and \( t_1 \), it is very likely that \( t_6 \) is still greater than \( t_1 \) and \( t_2 \). Next, we merge \( v_6 \) with \( v_2 \) to node \( v_8 \) and denote this merging as \( v_6 + v_2 \sim v_8 \). We can compare this merging with \( v_6 + v_5 \sim v_7 \) in Figure 1(b), since both merging start from \( v_6 \). On one hand, there is less imbalance on delay-targets for merging \( v_6 + v_2 \sim v_8 \) since \( t_6 - t_2 < t_6 - t_5 \). On the other hand, as \( C_6 < C_5 \), the merging \( v_6 + v_2 \sim v_8 \) has greater imbalance on load capacitance which makes it easier to achieve imbalanced delay-targets without wire snaking. If we compare the merging \( v_1 + v_8 \sim v_9 \) in Figure 1(d) and the merging \( v_1 + v_2 \sim v_5 \) in Figure 1(b), same conclusion can be obtained. Therefore, the maximum delay-target first merging indeed reduces the chance of wire snaking.

Besides the maximum delay-target criterion, there is another major difference between our merging scheme and previous works. Previous works such as [5] evaluate every pair of subtrees and choose a pair according to a single minimum distance criterion. Our maximum delay-target criterion only selects a subtree, and we will apply another criterion to choose another subtree (we call it companion subtree) to be merged with the maximum delay-target subtree. If we pick the subtree, which is closest to the maximum delay-target tree as a companion, then the negligence on the delay-target difference between them may again result in wire snakings. If we pick the subtree with the closest delay-target, these two subtrees may be far apart from each other and the merging may cause large wirelength too. Therefore, a subtree needs to be merged
to another subtree that is not only nearby but also with similar delay-target. In other words, we need to play in a three-dimensional space of \((x, y, \text{delay}\_\text{target})\). However, distance and delay are different kind of quantities and it is hard to compare them directly. Certainly, we can use a weighted sum of distance and delay-target difference to guide the merging. But, this merging criterion is ad hoc since the weighting factor is somewhat arbitrary.

We introduce a merging cost to include the concern on distance and delay-targets in a unified form. This merging cost is simply the wirelength needed for the merging to satisfy the delay-target constraint (2). Therefore, the merging cost is same as the Manhattan distance between the roots of two subtrees if there is no wire snaking. Otherwise, the extra wirelength due to wire snaking is included in the merging cost in addition. Therefore we choose the companion subtree, which will lead to the minimum merging cost.

Evidently, our maximum delay-target and minimum merging-cost merging scheme is simple and easy to be implemented for practical applications. The algorithm description for this merging scheme is given in Figure 3. When integrated with the DME embedding as in [5], the merging will be performed \(n - 1\) times for \(n\) clock sinks. The complexity of merging selection is \(O(n)\) due to the loop of line 3-6 in Figure 3. Thus, the overall complexity of our prescribed-skew routing is \(O(n^2)\).

4. Experiment

We implemented prescribed-skew clock routing algorithm using our proposed merging scheme and extended [5], one of the best zero-skew routing algorithms, to prescribed skews for comparison. Since our merging scheme includes two major components: (i) the maximum delay-target subtree first and (ii) choosing companion subtree according to the minimum merging cost, we tried them separately and together to observe each individual effect. Therefore, we compared the following four different merging schemes:

- E-ZST: Extended Zero-Skew Tree routing in [5] using the minimum distance merging. The DME implementation is extended such that the prescribed skew can be achieved.
- MIC: The Minimum merging-Cost merging which is very similar to E-ZST except that the merging selection is based on the merging cost between two subtrees instead of the distance between them.
- MAT: Choose the Maximum delay-Target subtree first and find its companion subtree that is its nearest neighbor(with minimum distance).
- MAT-MIC: This is the complete version of our proposed merging scheme, which is a combination of previous two techniques. First choose the maximum delay-target subtree and then find its companion subtree which results in the minimum merging cost between them.

The prescribed-skew routing with above different merging schemes are implemented in C and experiments are performed on a PC with 1.7GHz Pentium 4 microprocessor and 512Mb memory. The benchmark circuits are prim1, prim2 and r1-r5 downloaded from the GSRC Bookshelf (http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/). The delay-targets are generated through running the BST [3] code with a global skew bound of 100ps and taking the non-zero skew results. The BST code is also downloaded from the GSRC Bookshelf.

The experimental results are shown in Table I. Since the prescribed-skew routings with these different merging schemes all deliver the same prescribed skews, we only report the total wirelength here. The percentage reduction on wirelength with respect to E-ZST are listed in column 5. We can see that either the maximum delay-target or the minimum merging-cost technique itself can make significant improvement on wirelength over the naive extension from previous zero-skew routing. The improvement from the merging-cost based criterion alone is from 24% to 51%. The effect from the maximum delay-target ordering is more remarkable and shows 39%-53% wirelength reduction. The combination of these two techniques, which is our proposed merging scheme, yields wirelength improvement of 53%-61%. The CPU time for each routing algorithm are shown in the rightmost column of Table I. Note that our merging scheme is not only effective but also fast for practical applications.

5. Conclusion and Future Work

Even though traditional zero-skew routing methods can be applied to achieve prescribed skews, they may bring huge wirelength overhead as the difference among sink delay-targets are ignored in their merging schemes. We proposed a new merging scheme based on the maximum delay-target and minimum merging-cost ordering and integrate it with DME embedding. Experimental results on benchmark circuits show that this simple technique is very effective on minimizing wirelength for prescribed-skew clock routing. Since buffers become very important for clock network design, we will investigate efficient algorithms generating buffered clock tree for prescribed skews in future.

6. Acknowledgement

The authors would like to thank Dr. G. Ellis, Dr. W. Shi and Dr. M. Zhao for valuable discussions.
TABLE I. Comparisons of prescribed-skew routing with different merging schemes. The wirelength reduction is with respect to the wirelength from E-ZST.

<table>
<thead>
<tr>
<th>Testcase</th>
<th>#sinks</th>
<th>Merging</th>
<th>Wirelength</th>
<th>Wire reduction</th>
<th>CPU(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prim1</td>
<td>269</td>
<td>E-ZST</td>
<td>271746</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIC</td>
<td>152636</td>
<td>44.0%</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI</td>
<td>161248</td>
<td>40.6%</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI-MIC</td>
<td>109056</td>
<td>59.9%</td>
<td>1</td>
</tr>
<tr>
<td>prim2</td>
<td>603</td>
<td>E-ZST</td>
<td>645248</td>
<td>-</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIC</td>
<td>490895</td>
<td>23.9%</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI</td>
<td>314455</td>
<td>51.3%</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI-MIC</td>
<td>259751</td>
<td>59.7%</td>
<td>1</td>
</tr>
<tr>
<td>r1</td>
<td>267</td>
<td>E-ZST</td>
<td>2477975</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIC</td>
<td>1515972</td>
<td>38.8%</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI</td>
<td>1491833</td>
<td>39.8%</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI-MIC</td>
<td>1160145</td>
<td>53.18%</td>
<td>1</td>
</tr>
<tr>
<td>r2</td>
<td>598</td>
<td>E-ZST</td>
<td>4935718</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIC</td>
<td>2936488</td>
<td>40.5%</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI</td>
<td>2656833</td>
<td>46.1%</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI-MIC</td>
<td>2243723</td>
<td>54.5%</td>
<td>1</td>
</tr>
<tr>
<td>r3</td>
<td>862</td>
<td>E-ZST</td>
<td>6847146</td>
<td>-</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIC</td>
<td>3927066</td>
<td>42.7%</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI</td>
<td>3242982</td>
<td>52.6%</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI-MIC</td>
<td>2851177</td>
<td>58.4%</td>
<td>1</td>
</tr>
<tr>
<td>r4</td>
<td>1903</td>
<td>E-ZST</td>
<td>15123351</td>
<td>-</td>
<td>462</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIC</td>
<td>7358685</td>
<td>51.3%</td>
<td>1250</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI</td>
<td>7283887</td>
<td>51.9%</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI-MIC</td>
<td>5819055</td>
<td>61.5%</td>
<td>4</td>
</tr>
<tr>
<td>r5</td>
<td>3101</td>
<td>E-ZST</td>
<td>23013115</td>
<td>-</td>
<td>2572</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIC</td>
<td>11076659</td>
<td>51.9%</td>
<td>6234</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI</td>
<td>10695010</td>
<td>53.5%</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAI-MIC</td>
<td>8810532</td>
<td>61.7%</td>
<td>11</td>
</tr>
</tbody>
</table>

References