Co-Optimization of Memory BIST Grouping, Test Scheduling, and Logic Placement

Andrew B. Kahng†‡ and Ilgweon Kang‡
UC San Diego ECE† and CSE‡ Departments, La Jolla, CA 92093
{abk, igkang}@ucsd.edu

Abstract—Built-in self-test (BIST) is a well-known design technique in which part of a circuit is used to test the circuit itself. BIST plays an important role for embedded memories, which do not have pins or pads exposed toward the periphery of the chip for testing with automatic test equipment. With the rapidly increasing number of embedded memories in modern SOCs (up to hundreds of memories in each hard macro of the SOC), product designers incur substantial costs of test time (subject to possible power constraints) and BIST logic physical resources (area, routing, power). However, only limited previous work addresses the physical design optimization of BIST logic; notably, Chien et al. [7] optimize BIST design with respect to test time, routing length, and area. In our work, we propose a new three-step heuristic approach to minimize test time as well as test physical layout resources, subject to given upper bounds on power consumption. A key contribution is an integer linear programming ILP framework that determines optimal test time for a given cluster of memories using either one or two BIST controllers, subject to test power limits and with full comprehension of available serialization and parallelization. Our heuristic approach integrates (i) generation of a hypergraph over the memories, with test time-aware weighting of hyperedges, along with top-down, FM-style min-cut partitioning; (ii) solution of an ILP that comprehends parallel and serial testing to optimize test scheduling per BIST controller; and (iii) placement of BIST logic to minimize routing and buffering costs. When evaluated on hard macros from a recent industrial 28nm networking SOC, our heuristic solutions reduce test time estimates by up to 11.57% with strictly fewer BIST controllers per hard macro, compared to the industrial solutions.

Our main contributions can be summarized as follows.

- We propose a weighted hypergraph construction that allows use of top-down min-cut partitioning of memories into clusters that have good physical design and test scheduling attributes.
- We propose an ILP that comprehends parallel and serial testing of a given group of memories as it finds a minimum-test time solution with one or two BIST controllers.
- We use the above two elements, along with bottleneck matching to find BIST logic placement locations, in a heuristic that simultaneously reduces both BIST logic and test time costs in hard macros from a recent 28nm networking SOC.

In the remainder of this paper, Section II briefly reviews related works in the areas of test scheduling and memory BIST. Section III describes our ILP formulation to minimize test time taking advantage of available serialization and parallelization. Section IV presents our heuristic approach, and Section V gives experimental results with industrial testcases. Section VI describes directions of ongoing work and concludes the paper.

II. RELATED WORKS

In this section, we broadly classify related literature as dealing with (1) test scheduling and (2) BIST controller optimizations.

A. Test Scheduling

Test time reduction has long been a basic goal of DFT research, since test time is directly related to test cost. Parallel (simultaneous) testing reduces test time but is constrained by power and bandwidth (pin count) limits. Works such as that of Yao et al. [20], formulate and solve the test scheduling problem to minimize total test time while satisfying such constraints. Iyengar et al. [12] [13] adapt a rectangle packing problem formulation to test scheduling; they co-optimize test access mechanism (TAM) architecture and test wrapper, while designating a group of tests. Zou et al. [21] formulate SOC test scheduling as two-dimensional bin packing under given pin constraints, and simulated annealing is used to search for a heuristic optimum test schedule by perturbations to an initial solution.


Wang et al. [19] develop a test scheduling algorithm based on elements of the March algorithm for memory BIST; the objective is to minimize overall test time under a power constraint.

Unlike previous works, we study the minimization of total test time in the context of a mixture of serial and parallel testing, with multiple memory BIST controllers, by considering physical information...
of memories.\footnote{The mixture of serial and parallel testing induces what may be thought of as a partial level-oriented strip packing problem. (In the two-dimensional strip packing problem, (rectangular) items are packed into an “open-ended” rectangle of given height and infinite width, and the objective is to minimize width while packing all of the items into the rectangle )\cite{2}.) To our knowledge, the DFT literature has not yet considered this partial level-oriented strip packing problem of as a partial level-oriented strip packing problem. Since we use logical constraints to define parallel and serial testing. Devanathan et al.\cite{9} propose a physically-aware memory BIST datapath synthesis framework, wherein a hybridized synthesis approach achieves correct-by-construction, area-efficient memory BIST solutions. Devanathan et al. demonstrate the benefits from strategic approaches to physically-aware BIST in\cite{11} and built-in self-repair (BISR) design optimization methods in\cite{10}: such techniques mitigate the difficulties of physical design closure such as congestion and timing closure, even as the numbers of memory instances and BIST controllers in complex SOCs continue to increase. The authors of\cite{10} \cite{11} also note that their methods enable designers to apply more effective tests and reduce verification cycle times.

Chien et al.\cite{7} propose a memory BIST design optimization method to minimize test time, wire length and total area while considering several practical design constraints. To our knowledge,\cite{7} is the first published work considering aspects of physical design for memory BIST controllers. The authors adopt an integer linear programming (ILP) formulation for the assignment of memories to controllers. They then apply legalization and refinement steps to meet user-specified constraints and to further improve the quality of their solution. Although\cite{7} is the previous work that is closest to ours, we observe that it makes a number of simplifications that we avoid, e.g., (i) all memory instances in a BIST cluster are tested in parallel (leading to an unrealistic test time estimate); and (ii) only one cluster is tested at a time (preventing exploitation of parallel testing with multiple BIST controllers).\footnote{This being said, the test cost and the area/power overheads of memory BIST are rapidly drawing more attention to this topic.}

III. ILP FORMULATION

We develop an integer linear program (ILP) to solve the memory test scheduling problem when using multiple BIST controllers. Note that our ILP formulation is very different from those of\cite{5} \cite{6} \cite{8} \cite{15} since we use logical constraints to define parallel and serial testing. Table I defines notations used in our discussion. The objective is to minimize total test time, i.e.,

\[
\text{minimize } \max_{m_i} T_{E_i}
\]

where \( T_E = T_S + T_D \), and \( T_S \geq 0 \). We assume that a memory has test time proportional to its depth\cite{17} and test power proportional to the square root of its size. Based on our studies, we see that allowing both serial and parallel testing of memories can reduce test time as illustrated in Figure 1.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
Term & Meaning \\
\hline
\hline
\( M \) & Set of memory instances \\
\hline
\( m_i \) & \( i^{\text{th}} \) memory instance, where \( 1 \leq i \leq |M| \) \\
\hline
\( s_i \) & Size of word in \( m_i \) \\
\hline
\( y_j \) & Number of words in \( m_j \) \\
\hline
\( B \) & Set of memory BIST controllers \\
\hline
\( b_k \) & \( k^{\text{th}} \) memory BIST controller, where \( 1 \leq k \leq |B| \) \\
\hline
\( P \) & Set of partitions, (\text{k-way partitioning}) \\
\hline
\( p_i \) & \( i^{\text{th}} \) partition \\
\hline
\( D_i(p_i) \) & Diameter of the \( i^{\text{th}} \) partition \( p_i \) \\
\hline
\( T_S \) & Test start time of \( m_i \) \\
\hline
\( T_{E_i} \) & Test end time of \( m_i \) \\
\hline
\( D_i \) & Test duration of \( m_i \) \\
\hline
\( \tau_q \) & Instantaneous time \\
\hline
\( E(m_i) \) & Test power of \( m_i \) \\
\hline
\( E(t_q) \) & Total test power at time \( t_q \) \\
\hline
\( E_{\text{MAX}} \) & Upper bound on total test power \\
\hline
\( U_i(t_q) \) & Indicator whether \( m_i \) is under testing at time \( t_q \) \\
\hline
\( V_i(t_q) \) & Indicator whether \( t_q \geq T_S \) \\
\hline
\( W_i(t_q) \) & Indicator whether \( t_q \leq T_{E_i} \) \\
\hline
\( B_{k,i} \) & Indicator whether \( m_i \) is tested with BIST controller \( b_k \) \\
\hline
\( L_{k,i,j} \) & Indicator whether \( m_i \) and \( m_j \) belong to the same BIST controller \( b_k \) \\
\hline
\( L_{k,i,j} \) & Indicator whether \( m_i \) and \( m_j \) are tested in parallel with the same BIST controller \( b_k \) \\
\hline
\( F_{k,i,j} \) & Indicator whether \( m_i \) is tested before starting test of \( m_j \) with the same BIST controller \( b_k \) \\
\hline
\( Q_{k,i,j} \) & Indicator whether \( t_S \leq T_S \) \\
\hline
\( N_l \) & Large integer \\
\hline
\( N_{LL} \) & Large integer \( N_L \gg N_{LL} \) \\
\hline
\( \epsilon \) & Positive and very small real number, \( 0 < \epsilon \ll 1 \) \\
\hline
\end{tabular}
\end{table}

The ILP constraints are as follows.

Maximum power constraint. We use \( E_{\text{MAX}} \) to denote an upper bound on maximum available test power. The instantaneous testing power \( E(t_q) \) cannot exceed \( E_{\text{MAX}} \), as indicated by constraint (2).

\[
E(t_q) = \sum_{m_i \in M} U_i(t_q) \cdot E(m_i)
\]

where \( U_i(t_q) = \begin{cases} 1, & T_S \leq t_q \leq T_{E_i} \\ 0, & \text{otherwise} \end{cases} \forall m_i, \sum_{\forall t_q} U_i(t_q) \geq 1 \quad (3)
\]

BIST assignment constraint. We use the constraint (5) to ensure that each memory is uniquely assigned to a BIST controller for testing.

Fig. 1: Example with nine memories and two BIST controllers showing test time reduction when both serial and parallel testing are allowed. The left figure shows test time when only parallel testing is allowed. The right figure shows the reduced test time by allowing both serial and parallel testing.
$B_{kj}$ indicates whether $m_i$ is assigned to BIST controller $b_k$ for testing.

$$\sum_{b} B_{kj} = 1, \ m_i \in M$$

where $B_{kj} = \begin{cases} 1, & \text{if } m_i \text{ assigned to } b_k \\ 0, & \text{otherwise} \end{cases}$ (5)

Scheduling constraint. We define three indicator variables $I_{k,i,j}$, $F_{k,i,j}$, and $L_{k,i,j}$ to constrain the order of testing between two memories $m_i$ and $m_j$ that are assigned to the same controller $b_k$. These ensure that $m_i$ and $m_j$ are tested either in series or in parallel.

- $I_{k,i,j}$ indicates whether $m_i$ and $m_j$ share the same BIST controller $b_k$, and has a value of zero when this is true, as shown in Equation (6). If $I_{k,i,j} = 0$ (i.e., $m_i$ and $m_j$ share the same BIST controller),
  - $F_{k,i,j}$ indicates whether $m_i$ is tested before $m_j$ when tested serially; or
  - $L_{k,i,j}$ indicates whether $m_i$ and $m_j$ are tested in parallel, as shown in Equations (6)–(8). When $I_{k,i,j} = 1$, there is no scheduling relationship between $m_i$ and $m_j$.

$$I_{k,i,j} = \begin{cases} 0, & (B_{k,j} = 1) \wedge (B_{k,i} = 1) \\ 1, & \text{otherwise} \end{cases}$$

(6)

$$F_{k,i,j} = \begin{cases} 1, & (B_{k,j} = B_{k,i} = 1) \wedge (T_{k,j} \leq T_{k,i}) \\ 0, & \text{otherwise} \end{cases}$$

(7)

$$(I_{k,i,j} + F_{k,i,j} + L_{k,i,j}) = 1 \ , \text{ where } i \neq j$$

(8)

IV. Co-optimization of Test Scheduling and Memory BIST Logic Placement

We now describe our heuristic methodology for the co-optimization of test scheduling and memory BIST logic placement. Modern semiconductor chips contain hundreds of embedded memories scattered across the entire die. These memories can have various widths and depths, and can belong to different clock and logic hierarchies. Both the number and complexity of memory instances make the test scheduling problem extremely hard. We utilize a “divide-and-conquer” approach to develop a three-step heuristic method that (1) initially partitions all memories based on physical information using MLPart [4] [24]; (2) solves the test scheduling problem using an ILP formulation, followed by additional partitioning for better test time optimization; and (3) places memory BIST logic for each partition to minimize wirelength between memory BIST logic and memories. The goals of our heuristic approach are (1) minimization of test time, (2) reduction of number of partitions (i.e., number of BIST controllers), and (3) minimization of wirelength between each BIST and memories. We have developed a solver that uses command-line options as shown in Table II. Algorithms 1–3 outline our heuristic modeling approach.

<table>
<thead>
<tr>
<th>TABLE II: MBIST solver command-line options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-numMaxP</td>
</tr>
<tr>
<td>-numMinP</td>
</tr>
<tr>
<td>-maxMemP</td>
</tr>
<tr>
<td>-minMemP</td>
</tr>
<tr>
<td>-maxD</td>
</tr>
<tr>
<td>-conP</td>
</tr>
<tr>
<td>-gridS</td>
</tr>
<tr>
<td>-t</td>
</tr>
<tr>
<td>-longD</td>
</tr>
<tr>
<td>-shortD</td>
</tr>
</tbody>
</table>

A. Memory Partitioning

Memory partitioning is the “divide” step in our heuristic approach. We divide memory instances into $k$ partitions using MLPart [4] [24], a min-cut hypergraph partitioner based on the multilevel Fiduccia-Mattheyses hypergraph partitioning [24] algorithm. The input to MLPart is a hypergraph $G$, where each node in $G$ corresponds to a memory in the design (Algorithm 2). We define edge weights based on parameters such as memory shape, depth, power, location, etc. We expect that partitioning memories that have the same shape or depth

into one group leads to higher opportunity to minimize test time. This is because memories with the same depth can be tested in parallel and memories with the same power can be tested in serial, which minimizes idle space in test time and power. In addition, we assign larger weights to edges when memories are closer.

Algorithm 1 Memory Partitioning

**Procedure Partitioning(M)**

Input: $M$, numMaxP, numMinP, maxMemP, maxD, $K_{[3-6]}$

Output: $P^{out}$

1: for $n = numMaxP$ to numMinP do
2: $P^{r}$ ← single partition of $M$;
3: for $k = 1$ to $n - 1$ do
4: $G \leftarrow null$; $p_i \leftarrow 0$;
5: if $\max_{p_r \in P^{r}} (|p_r|) > maxMemP$ then
6: $p_i \leftarrow \arg \max_{p_r \in P^{r}} (|p_r|)$;
7: else if $\max_{p_r \in P^{r}} maxD > maxD$ then
8: $p_i \leftarrow \arg \max_{p_r \in P^{r}} (maxD)$;
9: else
10: $p_i \leftarrow \arg \max_{p_r \in P^{r}} (|p_r|)$; $p_j \leftarrow \arg \max_{p_r \in P^{r}} (maxD)$(p);
11: $p_i \leftarrow \arg \min_{p_r \in P^{r}} (p_i, cut, p_j, cut)$;
12: end if
13: for $i = 1$ to $n - 1$ do
14: $G \leftarrow GenerateHypergraph(p_i, r, G)$;
15: end for
16: $\{p_i, p_j\} \leftarrow MLPart(G)$;
17: $P^{r+1} \leftarrow \arg \max_{p_r \in P^{r}} (|p_r|)$; $B^{r+1} \leftarrow \{p_i\}$;
18: end for
19: $D_{max} = \max_{p_r \in P^{r}} (maxD)$; $P^{r+1} = P^{r+1}$
20: if $D_{max} > maxD$ then
21: if $numMaxP = numMinP$ then
22: return $P^{numMinP}$;
23: else
24: return $P^{numMaxP}$;
25: end if
26: end if
27: end for
28: return $P^{numMinP}$;

Algorithm 2 Construct Weighted Hypergraph (for $r^{th}$ criterion)

**Procedure GenerateHypergraph(p, r, G_m)**

Input: $p$, criterion index $r$, (hyper)edge weight $K_r$, hypergraph $G_m$

Output: $G$

1: $G \leftarrow G_m$;
2: for all $m_i \in p$, $i = 0$ to $|p| - 1$ do
3: $v_i \leftarrow mapping(m_i)$; // node $v_i$ corresponds to $m_i$ in partition $p$
4: add node $v_i$ to $G$;
5: visited($v_i$) ← false;
6: end for
7: for $i = 0$ to $|p| - 1$ do
8: $V_{conn} \leftarrow \emptyset$; $e \leftarrow null$;
9: $V_{conn} \leftarrow \{v_i\}$; // $v_i$ is reference node
10: visited($v_i$) ← true;
11: for $j = 0$ to $|p| - 1$ do
12: if $i \neq j$ then
13: if (visited($v_i$) == false) || ($r \geq 4$) then
14: if $v_i$ and $v_j$ satisfy criterion crit, then
15: $V_{conn} \leftarrow V_{conn} \cup \{v_j\}$; // $V_{conn}$ is set of nodes that satisfy crit w.r.t. $v_i$
16: visited($v_i$) ← true;
17: end if
18: end if
19: end if
20: end if
21: if $|V_{conn}| \geq 2$ then
22: $e \leftarrow$ connect all $r \in V_{conn}$ as (hyper)edge;
23: weight(e) ← $K_r$;
24: add (hyper)edge $e$ to $G$;
25: end if
26: end for
27: return $G$;
... which in turn leads to less test time. Therefore we minimize the number of partitions as long as the diameter of all partitions is \( \leq \max D \). Fewer partitions result in a larger solution space for scheduling, in...
C. Memory BIST Logic Placement

In the memory BIST logic placement step, we first define grids that cover the entire design. Any grid square that does not intersect memories is a possible location for BIST logic placement. We calculate the diameter from a grid square to all memories in a partition, and use this as a cost parameter. By calculating this cost parameter for all grid squares and all partitions, we generate a two-dimensional cost matrix for each grid square and memory partition. We then use this cost matrix to formulate and solve a min-weight maximum-matching problem in a bipartite graph, which is efficiently solvable using the Hungarian algorithm [23]. The resulting matching heuristically addresses timing criticality in paths between BIST logic and memories.

V. VALIDATION AND EXPERIMENTAL RESULTS

Our heuristic implementation is developed in C++ and compiled with g++ 4.8.0. All experiments are run on a 2.5GHz Intel Xeon E5-2640 Linux workstation with 128GB memory and 12 hyperthreaded CPU cores. In the partitioning step, we apply MLPart [24] on hypergraphs generated using Algorithm 2 above. In the scheduling step, we use CPLEX 12.5.1 [22] as our ILP solver to schedule testing of memories in each partition. Last, we solve the min-weight maximum matching problem in a bipartite graph [23] to assign BIST logic placement locations to partitions. (To our understanding, the turnaround time of our heuristic is not critical, and resynthesis of memory BIST logic after memory grouping takes only a few hours [26].) Table II presents command-line options in our implementation. In all of our experiments, we set 200 as the power constraint since the maximum $E(m_i)$ in testcases is $150 < E(m_i) < 200$.

To validate our heuristic methodology, we use six industrial testcases, each derived from a separate hard macro in a recent 28nm networking SOC product. Parameters of these testcases are given in Table IV. The number of memories in each testcase ranges from 124 to 160 and the number of partitions ranges from 7 to 13. Maximum and minimum number of memories, and maximum diameters without BIST logic, are also presented in Table IV.

| TCs | $|M|$ | $|P|$ | MAX | MIN | D ($\mu m$) |
|-----|------|------|-----|-----|------------|
| TC1 | 143  | 15   | 26  | 1   | 3900       |
| TC2 | 150  | 11   | 28  | 2   | 4500       |
| TC3 | 124  | 8    | 22  | 8   | 2200       |
| TC4 | 160  | 13   | 30  | 1   | 1340       |
| TC5 | 137  | 7    | 26  | 11  | 3200       |
| TC6 | 148  | 12   | 25  | 1   | 4100       |

Table V compares industrial results and our results. We achieve up to 11.57% improvement in estimated test time, strictly smaller number of partitions (i.e., number of memory BIST controllers), and reduced maximum diameter with respect to BIST logic placement location, compared to industrial results. Considering that test time is directly related to test cost and that fewer number of memory BIST logic leads to smaller die area, we believe this is a significant improvement. Furthermore, smaller maximum diameter of each memory partition (as shown in Figure 3) indicates better timing, which allows at-speed testing with smaller gate sizes and higher-$V_T$ cell instances.

VI. CONCLUSIONS

In this work, we propose a heuristic methodology to co-optimize partitioning, test scheduling and memory BIST logic placement to minimize test time. Our heuristic approach generates hypergraphs over memories with test time-aware weighting of hyperedges, along with top-down, FM-style min-cut partitioning. Our ILP formulation comprehends parallel and serial testing for test time optimization with respect to power constraints. Further, we place the BIST logic to minimize the maximum diameter for each BIST group, which minimizes routing and buffering costs and improves timing. On hard macros from a recent industrial 28nm networking SOC, our results achieve up to 11.57% reduction in test time compared to the industrial solutions, using strictly fewer BIST controllers.

Our ongoing work pursues three main directions. (1) First, recall that we construct the weighted hypergraph instance for top-down partitioning independently of any map of placement density or routing congestion. We currently do not evaluate our memory partitioning and BIST logic placement solutions after placement and routing, and signoff timing analysis. To bridge this gap, we seek to integrate our partitioning and BIST logic placement optimizations into a (production) physical implementation flow. (2) Second, our need to...
TABLE V: Comparison between industrial solution and our solution. (P = partition, B = BIST controller, MAX = max_{i\leq k}|p_i|, MIN = min_{i\leq k}|p_i|, D = maximum diameter with BIST logic, TT = test time, TTR = test time reduction, and Power = peak power.)

<table>
<thead>
<tr>
<th>Industrial Solution</th>
<th>Our Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D (\mu m)</td>
</tr>
<tr>
<td>TC1</td>
<td>13</td>
</tr>
<tr>
<td>TC2</td>
<td>11</td>
</tr>
<tr>
<td>TC3</td>
<td>8</td>
</tr>
<tr>
<td>TC4</td>
<td>13</td>
</tr>
<tr>
<td>TC5</td>
<td>7</td>
</tr>
<tr>
<td>TC6</td>
<td>12</td>
</tr>
</tbody>
</table>

apply SolveMBISTILP(2) to optimally schedule the testing of a large cluster of memories using two BIST controllers means that the hypergraph construction at some point leads min-cut partitioning “away” from good memory clusters. Thus, we seek improved hypergraph construction and weighting such that top-down mincut partitioning more directly produces a multi-way clustering that achieves minimum test time with k BIST controllers. (3) Third, recall that an initial motivation for this work is the disconnect between front-end DFT teams and back-end PD teams. We plan to enable the use of our tool by a PD team in a production SOC design environment to validate the accuracy and schedule impact of (i) early feedback on timing and need for LVT devices in the BIST logic, (ii) understanding of feasible memory groupings in light of test schedule and power constraints.

ACKNOWLEDGMENTS

We thank Dr. Jae-Gon Lee and Ms. Nancy MacDonald for valuable feedback and discussions of problem formulations.

REFERENCES


APPENDIX

LOGICAL CONSTRAINT HANDLING IN ILP

We describe the handling of logical constraints using indicator variables and very large numbers [3] [16] [18]. This method is extended for all indicators used in our formulation. To describe $U_i(t_q)$ in Equation (3), we define two more indicators, $V_i(t_q)$ and $W_i(t_q)$ as shown in Equations (9)–(11). A pair of inequalities in Equation (9) shows that $V_i(t_q) = 1$ when $T_{S_i} \leq t_q$. Likewise, Equation (10) shows that $W_i(t_q) = 1$ when $T_{E_i} > t_q$. $U_i(t_q) = 1$ if $t_q \geq T_{S_i}$ and $t_q < T_{E_i}$. In other words, when $V_i(t_q) = 1$ and $W_i(t_q) = 1$, we have $U_i(t_q) = 1$. Equation (11) shows the relation between $V_i(t_q)$, $W_i(t_q)$ and $U_i(t_q)$. Note that $V_i(t_q)$ and $W_i(t_q)$ can never be zero at the same time.

$t_q - T_{S_i} + \epsilon \leq n_i \cdot V_i(t_q)$

$T_{S_i} - t_q \leq \epsilon \cdot n_i \cdot (1 - V_i(t_q))$

$V_i(t_q) = \begin{cases} 1, & T_{S_i} \leq t_q \\ 0, & \text{otherwise} \end{cases}$

$t_q - T_{E_i} + \epsilon \leq n_i \cdot W_i(t_q)$

$T_{E_i} - t_q \leq \epsilon \cdot n_i \cdot W_i(t_q)$

$W_i(t_q) = \begin{cases} 1, & t_q < T_{E_i} \\ 0, & \text{otherwise} \end{cases}$

$V_i(t_q) + W_i(t_q) - 1 = U_i(t_q)$

Equations (12)–(14) show how we define $F_{k,i,j}$ and $L_{k,i,j}$ with $I_{k,i,j}$, $Q_{k,i,j}$, and $N_{LL}$. Whenever $I_{k,i,j} = 1$, inequalities are always true by virtue of $N_{LL}$, which means that $m_i$ and $m_j$ are irrelevant.