Chapter 3

Algorithm for Cell Clustering

Based on our motivation in Chapter 2, we propose a design flow to determine how to cluster cells to share sleep transistors. First, we perform the cell characterization to compute the parameters of each cell and the allowed maximum current to flow through a sleep transistor under a specified performance degradation. Then, we model the relations of gate transition-time as a *relation graph* taking topology and functionality into consideration. Finally, clustering of gates is formulated as a clique partitioning problem. Figure 3.1 shows the design flow. The detailed description of each step is described in the following sections.

3.1 Cell Characterization

The objective of the cell characterization is to determine the size of sleep transistors (the maximum current flowing through sleep transistors under a

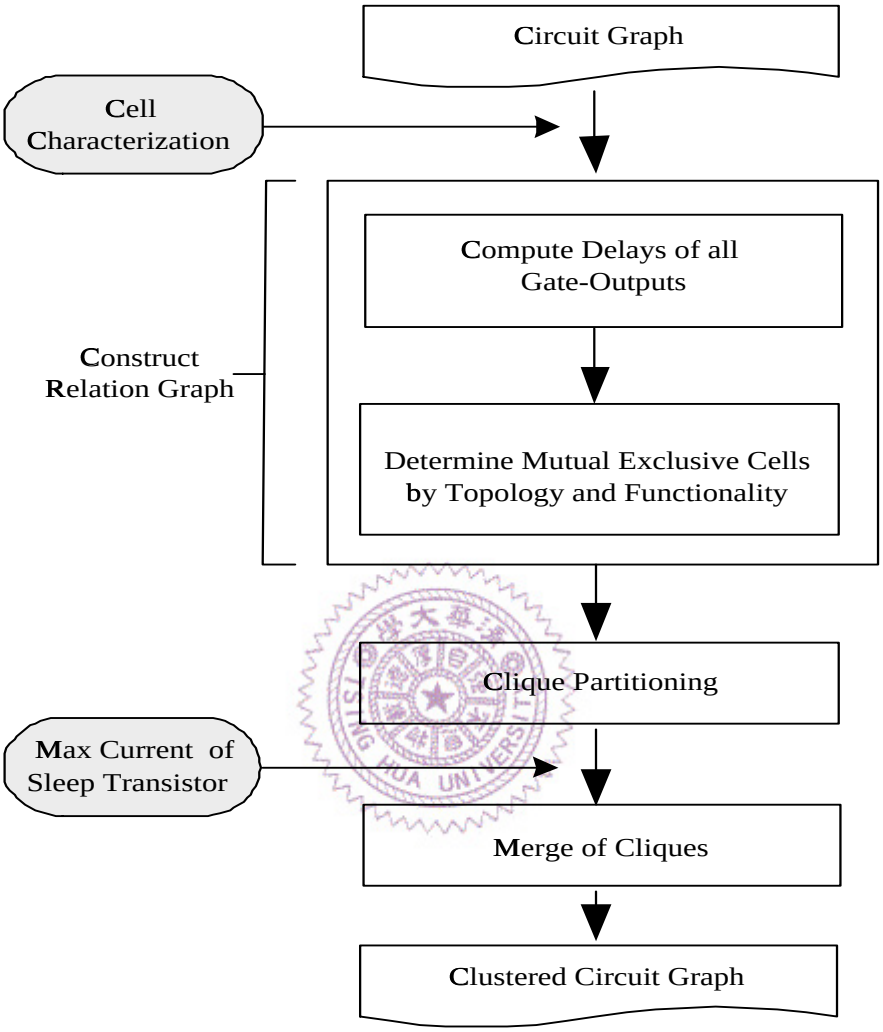

Figure 3.1: Design flow of the cell clustering algorithm

specified timing constraint) and to compute the parameters of cells such as cell delay, discharge current value, and discharge time.

The size of sleep transistors are calculated by analyzing the equations presented in [10]. The delay of a single gate at the absence of a sleep transistor can be formulated as Equation (3.1), where V_{dd}=1.8V, α is the velocity saturation index and V_{tL} is the low threshold voltage set to 350mV using TSMC 0.18μm CMOS technology.

$$\tau_d = \frac{C_{load} V_{dd}}{(V_{dd} - V_{tL})^{\alpha}} \tag{3.1}$$

On the other hand, sleep transistors in MTCMOS circuits will degrade the performance. Let the delay of a single gate be expressed as Equation (3.2), where V_x is the potential of the virtual ground.

$$\tau_d^{sleep} = \frac{C_{load} V_{dd}}{(V_{dd} - V_x - V_{tL})^{\alpha}} \tag{3.2}$$

Assuming the circuit in MTCMOS tolerate a 5% performance degradation formulated as Equation (3.3). Then, after setting α to 1 and substituting Equation (3.3) by Equation (3.1) and Equation (3.2), V_x can be formulated as in Equation (3.4) and its value is computed as 0.0725V.

$$\frac{\tau_d}{\tau_d^{sleep}} = 95\% \tag{3.3}$$

$$V_x = 0.05(V_{dd} - V_{tL}) \tag{3.4}$$

Then we do a SPICE simulation under TSMC spice model of 0.18 μm technology. By setting V_{tH} to be 500mV, the maximum current of one sleep transistor is then computed as 432uA under $(\frac{W}{L})_{sleep} \approx 10$. This is the value that the maximum current of the circuit flows through sleep transistors in MTCMOS in order to maintain the performance of the circuit.

At this same time, to characterize cells, for each gate in the library, we construct its SPICE model using TSMC spice model of 0.18 μm technology for simulation. For each gate, all input combinations are applied and the worst case delay is taken as the propagation delay. Note that the fan-out of a gate is also considered and a load of $6fF$ is applied as one fan-out of each gate. Then the delay of a 1-output inverter is defined as one time unit and the delay of all gates are computed accordingly. In the meantime, the peak current value and time at which the discharge occurs are monitored. We also take the worst case discharge current value as the peak current value and record the time unit at which discharge occurs. The discharge current of each gate takes a triangular shape. However, to simplify the computation, we assume the maximum discharge current occurs for one time unit of the propagation delay.

3.2 Construct Relation Graph

To reduce the number of sleep transistors, we want to cluster as many gates as possible for one sleep transistor and the current flowing from each cluster does not exceed the maximum allowed current of the sleep transistor. Previous work proposed to cluster gates that make transition at different time (topology information). We found that more gates can be clustered if the functionality of the circuit is also taken into consideration as presented in our motivation. In this section, we will first present how to compute the transition time of each gate. Then, based on this timing information, we will construct the relations of all gates taking both timing (topology) and functionality into consideration. Before we present our algorithm, we first define a *relation graph* $G_r(V, E)$, which represent the discharge relation among gates taking into consideration both topology and functionality. A vertex $v_i \in V$ stands for one gate and an edge $(v_i, v_j) \in E$ means that v_i and v_j do not make transition at the same time.

3.2.1 Compute Delays of all Gate-Outputs

We adopt a skill similar to IMAX [11], which is input-pattern independent and estimates an upper bound envelope of all possible current waveforms. The algorithm proceeds with a Breath-First Search from primary

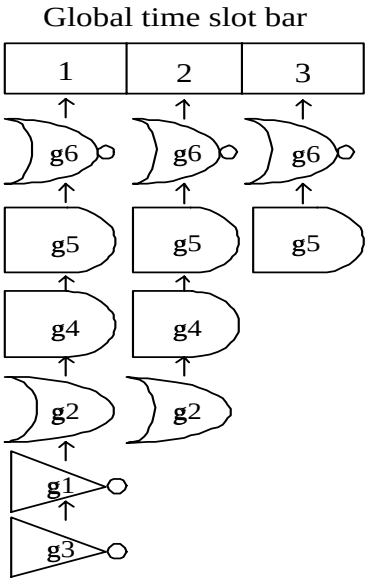

Figure 3.2: Global time slot bar

inputs to primary outputs. At each node, we compute the possible time slots that a gate may switch. There is a global time slot bar. After the possible switching times are obtained, the gate is linked to the time slot of the global time slot bar if the gate switches at that particular time slot. After the search, we get a global time slot bar which has the information about the upper bound of possible switching gates at each time slot in one cycle. During the computation, we also record the path information. Take the circuit in Figure 2.3 as an example. Assuming unit delay is used for all gates when the discharge of gates occurs. For example, to compute the possible switching times of $g5$. Let the transition times of $g4$ be computed as time

slots one and two. Then, the transition times of $g5$ are computed through primary input b, which is time slot one, and through $g4$, which are time slots two and three. Therefore, there are three possible discharge time slots (one, two and three) for the output of gate $g5$. As shown in Figure 3.2, $g5$ is linked to the time slots one, two, and three of the global time slot bar. Similarly, we compute the transition times of the all gates and find that there are three time slots in one cycle for this circuit. All gates are linked to the global time slot bar as shown in Figure 3.2.

3.2.2 Determine Mutual Exclusive Cells by Topology and Functionality

Recall that in our *relation graph*, each edge $\in E$ stands for a mutually exclusive relation in terms of gate output transition time. To build this graph, initially, we assume the *relation graph* is a complete graph. That is, all gates are mutually exclusive. Then, we use the topology and functionality information to decide if an edge is deleted one by one.

The checking procedure starts with the first time slot of the global time slot bar. For each time slot, there are a set of gates linked to this slot. In terms of circuit topology, it means that these gates will switch simultaneously and the edges connecting these gates in the *relation graph* should be deleted. However, due to the dependency relations of signals (multiple fan-out of a

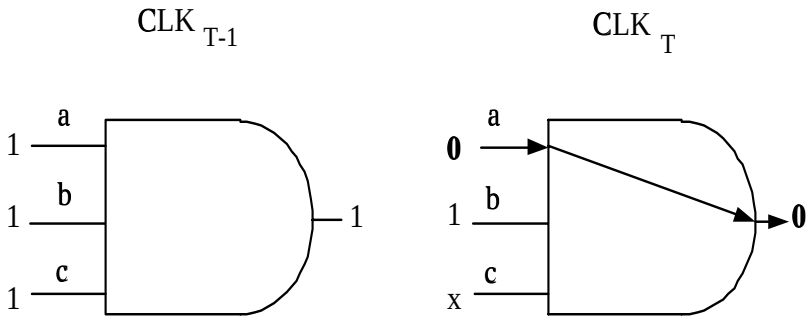

Figure 3.3: AND gate discharge scenario

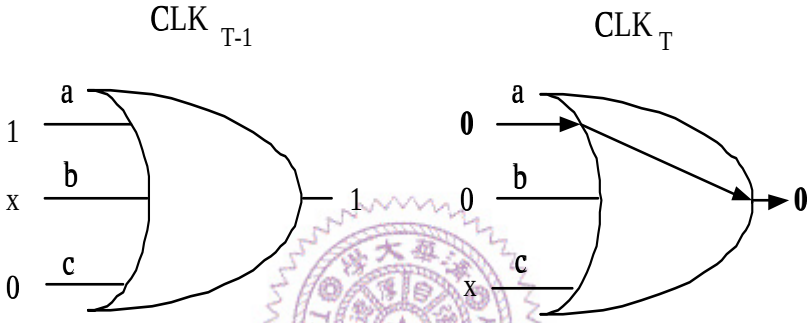

Figure 3.4: OR gate discharge scenario

gate, reconverged paths..), two possible simultaneous switching gates in the global time slot bar are not sure to discharge at the same time in terms of functionality.

To check if two gates in the same time slot are indeed switch simultaneously, we take advantage of the ATPG-like skill to check the input cones of two gates. During the functionality checking, two successive clock cycles

rather than one clock cycle was considered in order to reflect the transition of signals. If we find any signal conflict during the functionality checking, it means there is no input vector to make the checked two gates discharge simultaneously at this particular time slot. In other words, these two gates are mutually exclusive at this time slot and we will not delete their relation edge in the *relation graph*.

The details of the functionality check are described as follows. We know that once one input signal which is a controlling input of the gate is determined, the output signal of the gate is determined regardless of other inputs. On the other hand, no non-controlling inputs of the gate can determine the output signal until the last non-controlling input of the gate arrives. Based on these implications, we will develop our functionality check algorithm to check pairs of gates which linked to the same time slot.

Before we describe our implication rule, we first define some terms. We say that the input signal which makes the output transition is an *on-input*. Inputs that arrive before the *on-input* are called *early-side-inputs*. Inputs that arrive later than *on-input* are called *late-side-inputs*.

We will first demonstrate our implication by a discharged AND gate and a discharged OR gate. In Figure 3.3, assume the input a is the *on-input*, b is the *early-side-input* and c is the *late-side-input* of the AND gate. The signal

transition of $1 \rightarrow 0$ on a from the $(T-1)_{th}$ clock cycle to the T_{th} clock cycle will make a output signal transition of $1 \rightarrow 0$. In this case, we will get an implication that the input b must be 1 in the $(T-1)_{th}$ clock cycle because the output signal in the $(T-1)_{th}$ clock cycle is 1, and b must stay 1 in the T_{th} clock cycle because b can not yet make the output signal transition. Similarly, it also implies that the input c must be 1 in the $(T-1)_{th}$ clock cycle but is don't-care in the T_{th} clock cycle.

In Figure 3.4, we also assume the input a is the *on-input* , b is the *early-side-input* and c is the *late-side-input* of the OR gate. The *on-input* a makes a $1 \rightarrow 0$ transition of output signal in the T_{th} clock cycle. We have an implication that the early input b in the $(T-1)_{th}$ clock cycle is don't-care because the signal a (1) is the controlling input of OR gate in the $(T-1)_{th}$ clock cycle. However, b must be non-controlling value (0) in the T_{th} clock cycle because b can not decide the output signal of OR gate. It also implies that the *late-side-input* c is not arrived in the T_{th} clock cycle where a makes a transition. Hence, c must be 0 in the $(T-1)_{th}$ clock cycle and c is don't-care in the T_{th} clock cycle. Using the same techniques, we develop implication rules for four output transition scenario of AND gate and OR gate in Table 3.1 and Table 3.2, repectively. The implication rules of gates of different functionalities can be easily developed in the same fashion.

Table 3.1: Implication rule of AND gate

AND gate							
Output		On Iutput		Early Side Input		Late Side Input	
CLK_{T-1}	CLK_T	CLK_{T-1}	CLK_T	CLK_{T-1}	CLK_T	CLK_{T-1}	CLK_T
0	0	x	x	x	x	x	x
1	1	1	1	1	1	1	1
1	0	1	0	1	1	1	x
0	1	0	1	x	1	1	x

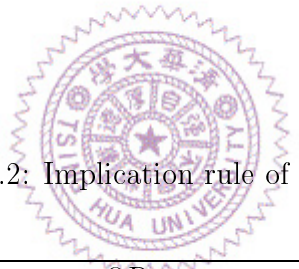

Table 3.2: Implication rule of OR gate

OR gate							
Output		On Iutput		Early Side Input		Late Side Input	
CLK_{T-1}	CLK_T	CLK_{T-1}	CLK_T	CLK_{T-1}	CLK_T	CLK_{T-1}	CLK_T
0	0	0	0	0	0	0	0
1	1	x	x	x	x	x	x
1	0	1	0	x	0	0	x
0	1	0	1	0	0	0	x

Now, we use one example to illustrate how to use this implication rules to check if two gates are indeed discharged simultaneously. Take the circuit in Figure 3.5 as an example. We are to check if the gate $g1$ and $g2$ are discharged at the seventh time slot in the T_{th} clock cycle. The *on-input* paths of them are $(d - g4 - j - g3 - n - g1)$ and $(a - g8 - f - g7 - p - g2)$. According to our implication rule, for $g1$ to discharge, signal transitions $1 \rightarrow 0$, $0 \rightarrow 1$, and $1 \rightarrow 0$ from the $(T-1)_{th}$ to the T_{th} clock cycle are assigned, repectively, to edges n, j, and d on the *on-input* path of $g1$. Since edge k is an *early-side-input* for $g3$, it is implied that edge k has the signal transition of $x \rightarrow 1$ from the $(T-1)_{th}$ to the T_{th} clock cycle. Similarly, since the arrival time of m is 3 unit which is earlier than that of the *on-input* of $g1$, 4, it is implied that m sustains value 1 from the $(T-1)_{th}$ to the T_{th} clock cycle. Consequently, the input h and i to the $g5$ must sustain 0 from the $(T-1)_{th}$ to the T_{th} clock cycle and the primary input c must be 1 from the $(T-1)_{th}$ to the T_{th} clock cycle due to the inverse functionality of $g6$. Similarly, for $g2$ to discharge, $1 \rightarrow 0$, $1 \rightarrow 0$, and $0 \rightarrow 1$ from the $(T-1)_{th}$ to the T_{th} clock cycle are assigned, respectively, to edges p, f, and a on the *on-input* path of $g2$. Since the edge o is an *early-side-input* for $g2$, it is implied that edge o must sustain value 1 from the $(T-1)_{th}$ to the T_{th} clock cycle. Since the arrival time of l is 3 unit which is later than that of the *on-input* of $g7$, 1, it is implied that l

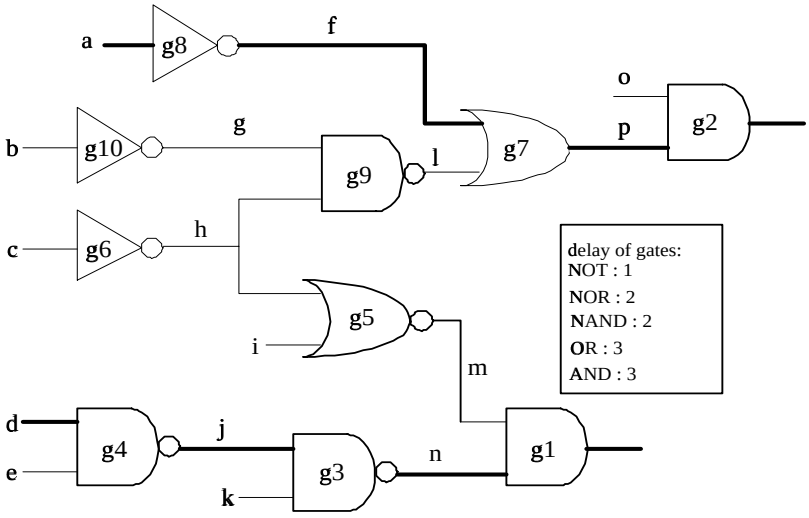

Figure 3.5: Implication example

has the signal transition of $0 \rightarrow x$ from the $(T-1)_{th}$ to the T_{th} clock cycle. Consequently, the input g and h to the $g9$ must have the signal transition of $1 \rightarrow x$ from the $(T-1)_{th}$ to the T_{th} clock cycle. Obviously, there is a conflict on edge h that one path requires h sustain signal $0 \rightarrow 0$ and the other path the signal transition of $1 \rightarrow x$. That is, no input vectors can make $g1$ and $g2$ discharge simultaneously at this time slot. So, we can ensure that $g1$ and $g2$ are mutually exclusive at this time slot.

Note that during functionality checking, the conflicts not only occur in the intersection of input cones of two gates but also in the input cone of one gate alone.

3.3 Clique Partitioning

Note that all gates in a clique of our *relation graph* are mutually exclusive and only the maximum current among them is computed as the discharge current of the clique. Therefore, after the new *relation graph* is constructed, we want to partition our *relation graph* to as fewer cliques as possible. The heuristic algorithm in [12] is utilized to achieve our objective. The heuristic algorithm proceeds by combining vertices in the *relation graph* step by step. At each step, a pair of vertices with the largest number of common neighbors is selected. Then these two vertices are combined to from a new vertex and the *relation graph* is updated. This algorithm stops when there is no pair of vertices to combine and each vertex stands for one clique finally. The algorithm is shown in Figure 3.6.

3.4 Merge of Cliques

After clique partitioning, we have many cliques and each clique has one discharge current. To use as fewer sleep transistors as possible, we can merge some cliques to share one sleep transistor as long as the sum of their currents is not over the maximum discharge current of a sleep transistor. To that end, we develop algorithm to merge cliques. The problem can be formulated as the Bin Packing problem where the capacity of the bin is the maximum allowed

```
1 Algorithm Clique Partitioning Algorithm()
2 Input : Relation Graph G = (V, E)
3 Output : Clustered Relation Graph
4
5   While(E ≠ ∅)
6       find (v_i, v_j) ∈ E with largest set of common neighbors;
7       s ← v_i ∪ v_j;
8       V ← V ∪ s;
9       delete edges linked v_i or v_j;
10      add edges connecting between s and neighbors of s;
```

Figure 3.6: Clique partitioning

current of a sleep transistor and that of an object (a clique) is the maximum current of the clique. We will merge cliques by the best-fit decreasing greedy Bin Packing algorithm. First, we sort the cliques according to their currents. Cliques in the ordered list are then merged to the bin to share one sleep transistor until the merge of the next candidate clique results in the overflow of the maximum current of one sleep transistor. We continue to merge cliques from the next candidate clique in the same way until all cliques are processed. The algorithm is shown in Figure 3.7.

```
1 Algorithm Merge of Cliques Algorithm()
2 Input : Cliques List
3 Output : Merges of Cliques
4
5   sort cliques list in decreasing order;
6   add new bin, B, and set B = ∅;
7   set capacity C_B of B to 0;
8   for each clique in cliques list
9       if(C_B + capacity of clique ≤ maximum allowed current)
10          B ← B ∪ clique;
11          C_B = C_B + capacity of clique;
12      else
13          add new bin, B, and set B = ∅;
14          set capacity C_B of B to 0;
15          B ← B ∪ clique;
16          C_B = C_B + capacity of clique;
```

Figure 3.7: Merge of cliques