State Assignment

Assigning Bits to States
- How and why it is done
- Assigning on a Karnaugh map
- Reasons For Choosing A Particular Assignments
  - Assignments that minimize logic

Guidelines
- a) Do something with unused states
- b) It doesn't matter where you start
- c) When you cuddle, it doesn't matter on which side

Heuristics
1. Parents of a child (for same input) should cuddle.
2) Sisters should cuddle
3. If rules 2) and 3) are inconclusive, states with the same output should cuddle.

Examples
Sequential Circuits

State Assignment

An Unassigned State Graph

An Unassigned State Table

The States Have Names (or Letters)
They Have Not Yet Been Given Bit Patterns

Choose An Assignment.
For 4 States We Need 2 Bits Per State.
Each Bit Is Stored In One Flip-flop.

One Assignment
Rst = 00
By = 01
K = 11
Go = 10

A 2nd Assignment
Rst = 00
By = 01
K = 11
Go = 10

A 3rd Assignment
Rst = 00
By = 01
K = 01
Go = 10

A 4th Assignment
Rst = 11
By = 01
K = 00
Go = 10

A 5th Assignment
Rst = 10
By = 00
K = 01
Go = 11

A 6th Assignment
Rst = 00
By = 01
K = 11
Go = 10

A 7th Assignment
Rst = 00
By = 01
K = 11
Go = 10

There are 24 two-bit assignments
Later we see only 3 give different circuits.

Sequential Circuits

State Assignment

When one first designs a finite-state machine, one uses symbolic states like S0, S1 and S2, or better, Rst, WaitOn1, and Got10.

Eventually one must equate these with the bits on the outputs of flip-flops. This is called state assignment.
States Assignment; Inserting into State Table

A State Table After Assignment

The symbolic states in the original state table, are filled out with the bit patterns decided upon during state assignment.
Defining the State Assignment on a K-map

Assigning 8 states on a K-map

<table>
<thead>
<tr>
<th>( Q_2 )</th>
<th>( Q_1Q_0 )</th>
<th>state</th>
<th>symbol assignm't</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00 Rst</td>
<td>R</td>
<td>000</td>
</tr>
<tr>
<td>0</td>
<td>01 S</td>
<td>P</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>11 L</td>
<td>T</td>
<td>011</td>
</tr>
<tr>
<td>1</td>
<td>00 M</td>
<td>Qd</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>10 N</td>
<td>Qd</td>
<td>111</td>
</tr>
<tr>
<td>1</td>
<td>11 N</td>
<td>N</td>
<td>110</td>
</tr>
</tbody>
</table>

Assigning 16 states on a K-map

<table>
<thead>
<tr>
<th>( Q_3Q_2Q_1Q_0 )</th>
<th>state</th>
<th>symbol assignm't</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 Rst</td>
<td>R</td>
<td>0000</td>
</tr>
<tr>
<td>01 S</td>
<td>A</td>
<td>0001</td>
</tr>
<tr>
<td>01 C</td>
<td>B</td>
<td>0011</td>
</tr>
<tr>
<td>0011 D</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>11 M</td>
<td>F</td>
<td>0100</td>
</tr>
<tr>
<td>1101 G</td>
<td>0111</td>
<td></td>
</tr>
<tr>
<td>1111 U</td>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>1110 etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assigning States on a K-Map

This is essentially the same as listing the state assignments, but it takes less writing and lists them in the right order to make the state table.
REASONS FOR CHOOSING A PARTICULAR ASSIGNMENT

1) No Reason; Choose Arbitrary Bit Patterns (Done Above)

2) Choose To Reduce Combinational Logic. This tends to reduce the circuit’s:
   - cost
   - delay
   - power consumption

3) Choose to reduce number of AND terms (for PLAs).

STATE ASSIGNMENTS THAT MINIMIZE LOGIC

- No Good Assignment Method To Minimize Logic.
- Only Sure Way: Try all \((2^N)!\) assignments

1 N Flip-flops. For \(N=2, 3,\) and 4, symmetry will reduce this to 3, 840 and 54,486 respectively.

Reasons for Choosing a State Assignments

An arbitrary choice is easy, and will always work.
However it may require a lot of extra logic.
Assigning to reduce logic, depends on implementation.

- For logic built with gates, one wants to reducing the number and width (number of inputs) of gates.
- For PLAs one wants to reduce the number of AND terms. The width does not matter.
- For some array logic (Field Programmable Gate Arrays) one sometimes has more flip-flops that one needs. It is possible then to reduce the gate count be using extra flip-flops.

1 Some techniques are one-hot, one-cold, M-of-N, ( ) and shift-register machines (Lala, Prag.K., Practical Digital Logic Design and Testing, Prentice Hall, 1996, pp.186-189 and pp 287-309).
State Assignments That Minimize Logic

- Objective is fewer gates
  It costs less to build.
  Fewer gates usually give a faster circuit.
  Fewer gates use less power.

- No Good Method
  A Good Designer Can Often Beat Computer Programs
  \( (2^N)! \) possible assignments
  Remove equivalent assignments get \( (2^N-1)!/N \)
  Which gives huge numbers
  Which is why assignment is so hard.

Rules for State Assignment:

Three rules (heuristics or inexact algorithms)

1. Parents of a child (for same input) should cuddle.
2. Sisters should cuddle.
3. If outputs are complicated, states with the same output should cuddle.

Three guidelines

a) Do something with unused states.
b) It doesn't matter where you start.
c) When you cuddle, it doesn't matter on which side.

<table>
<thead>
<tr>
<th>flip flops</th>
<th>Assignments total</th>
<th>Assignments unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>40320</td>
<td>840</td>
</tr>
<tr>
<td>4</td>
<td>(2.09\times10^{13})</td>
<td>54,486,432,000</td>
</tr>
<tr>
<td>5</td>
<td>(2.63\times10^{35})</td>
<td>6.85x10^{31}</td>
</tr>
<tr>
<td>6</td>
<td>(1.27\times10^{89})</td>
<td>2.75x10^{84}</td>
</tr>
<tr>
<td>7</td>
<td>(3.86\times10^{215})</td>
<td>5.97x10^{209}</td>
</tr>
<tr>
<td>N</td>
<td>((2^N)!)</td>
<td>((2^N-1)!/N)</td>
</tr>
</tbody>
</table>

State Assignments

Many people have worked on state assignment methods.
But no one has devised method that gives close to minimum logic.

Number of State Assignments

For \(N\) flip-flops there are \(2^N\) states. Thus the first state has \(2^N\) assignment choices, the second \(2^N-1\) etc. giving \((2^N)!\) assignments.

However if one inverts all the Q bits, (or just one flip-flop Q) the logic estimate is still the same since inverters are not included in a gate estimation. This gives \((2^N-1)!\) assignments.

Also if one interchanges columns in the state table, this is the same as interchanging variable names, and the logic equations are the same. If these symmetries are removed one gets \((2^N-1)!/N!\) states.

If there are \(D\) don’t care states one gets
\n\(\frac{(2^N-1)!}{(D! \ N!)}\) assignments.

It is a hard problem

With 2 flip-flops one can easily try all three the choices
With 7 flip-flops, checking all unique assignments, would take \(10^{193}\) years at one calculation/nanosecond

Rule (Heuristic) 3

A complicated output is when the output logic equations look messy and the next state equations look much simpler.

---

Guideline a: Unused States

Rules for State Assignment

What to do with unused (don’t care) states

**Don’t care**

There are two types of “d” here:

- In the right-hand table, it means substitute any state
- In the left and centre state graph, it means “go to this state for both 1 and 0.”

**Fail Safe**

If, because of power-line glitches, electromagnetic interference, or a static-electricity discharge, one gets into an unused state, the machine will behave in a way that minimizes the damage. This could be either:

1. Go to a specific state which minimizes the damage. This is often the reset state. This allows the machine to continue automatically.
2. Stay in the unused state and do nothing. This means someone must come, see what caused the problem, and manually do something.
Gideline b: It Doesn’t Matter Where You Start

Rules For State Assignment

Objective is fewer gates

Guideline b:

It doesn’t matter much where you start
For convenience make the reset state 000.

Circuit with Rst = 000

Same Circuit except inverters make Rst = 111

Sort of Proof
Circuits differ only by extra inverters
Inverters are not part of gate count.
Also NAND/NOR conversions will cancel most extra inverters.

Gideline b: It Doesn’t Matter Where You Start

It Does Not Matter Where You Start
Example showing how DeMorgan can be used to cancel most extra inverters.

The difference in complexity is so small, that there is no advantage in trying different assignments for a starting state
Since we usually start a state assignment by choosing a reset state, make it Rst = 00... This allows one to use the Reset pin available on most flip flops.
Rule 1: Parents of the Same Child Cuddle

Rule 1 (Strongest Rule)

Parents of the same child (for same input) should cuddle up

Example State Table

<table>
<thead>
<tr>
<th>State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>R</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>G</td>
<td>E</td>
<td>K</td>
</tr>
<tr>
<td>H</td>
<td>R</td>
<td>K</td>
</tr>
<tr>
<td>J</td>
<td>d</td>
<td>K</td>
</tr>
</tbody>
</table>

(State with the same next state (for the same input) should have adjacent assignment.)

Parents

- Possible assignment for R-E
  - Child
  - X=0
  - B
  - 00
  - 01
  - E

- Possible assignment for C-G
  - X=0
  - C
  - 00
  - 01
  - G

List of parents

- [(R,E), (G, K, H, J)]

Use this assignment

State table is now in K-map order

Rule 1: Parents of the Same Child for the Same Input

Changing the state table to K-map form

Arrange the top of the state table according to the left column of the Karnaugh map.

Arrange the bottom of the state table following the right column of the Karnaugh map.

Move the bottom of the state table over to the right of the top part of the table, being careful to interchange the x=0 and x=1 next-state columns.

The result is a symbolic K-map of the logic.

The Significance of the Placement

Notice in the next states that B-B, and D-D form a two square cluster, and C-G-K-J forms a 4-square cluster. These will turn out to be very easy to circle on the final Karnaugh maps.

Shareable AND Terms

This rule is good for making shareable AND terms. These are especially useful for PALs and PLAs which have limited AND terms.
Rule 1: Explanation

Why Rule 1 Works (a low priority explanation)

Groups of Zeros

One circles groups of “1”s to get Σ of Π equations.
It is also useful to have a group of adjacent “0”s as an area free of “1”s which gives simpler logic.

Generalization of Rule 1

The “Into” Rule

If a number of parent states branch into a single child for the same input, group these state assignments into a single circle on the K-map.

With multiple inputs like x and y, if y is the same for both transitions, but x is different, grouping the states will help, but not as much as if they were both the same.

\[ \text{Input conditions } xy \text{ should be the same for all the inputs.} \]

But one (or some) variable(s) the same will may help a little.

---

Rule 2: Sisters Should Cuddle

Rule 2 (Medium Rule)

Sisters Should Cuddle Up
(they will have different inputs)

Original State Table

<table>
<thead>
<tr>
<th>STATE</th>
<th>NEXT STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>R</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>K</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>G</td>
<td>E</td>
<td>K</td>
</tr>
<tr>
<td>H</td>
<td>K</td>
<td>H</td>
</tr>
<tr>
<td>J</td>
<td>d</td>
<td>K</td>
</tr>
</tbody>
</table>

State Table arranged as per K-Map

<table>
<thead>
<tr>
<th>STATE</th>
<th>NEXT STATE</th>
<th>NEXT STATE</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>B</td>
<td>C</td>
<td>R</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>H</td>
<td>R</td>
<td>K</td>
<td>E</td>
</tr>
</tbody>
</table>

Possible Placement for B-C

<table>
<thead>
<tr>
<th>STATE</th>
<th>NEXT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>01</td>
<td>B</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

List of sisters
(B,C) (G,H) (E,R) (B,E) (E,K) (R,K)

Rule 2 (a low priority explanation)

Rule 2 was used because it makes a “catchy” rule.

The more usual statement is
“Children of the same parent should be adjacent on the K-map.”

Generalization of Rule 2

The “Out Of” Rule

If a single parent states branches into several children, clearly for different inputs, group the sister state assignments into a single circle on the K-map.

For best results, make part of the state code match the input.

Assignment for sisters

Group the child states together on the K-map. Try to make part of the state code match the input.

---

Rule 2: Explanation

Why Rule 2 Works (This is a low priority slide)

Sisters should be adjacent on a K-map

State Table arranged as per K-Map

<table>
<thead>
<tr>
<th>STATE</th>
<th>NEXTSTATE</th>
<th>NEXTSTATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>H</td>
<td>R</td>
<td>K</td>
</tr>
</tbody>
</table>

Placement Using Rule 2

- Take the State Table in K-map order
- Fill in next state assignments where rule was used.

State Table / K-Map

<table>
<thead>
<tr>
<th>STATE</th>
<th>NEXTSTATE</th>
<th>NEXTSTATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>R=000</td>
<td>101</td>
<td>100C</td>
</tr>
<tr>
<td>E=001</td>
<td>101</td>
<td>001E</td>
</tr>
<tr>
<td>K=011</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>H=010</td>
<td>R</td>
<td>K</td>
</tr>
</tbody>
</table>

- There are only groups of two.
- (They may be groups of “1”s or “0”s)

Two maps out of three

This rule gives us two out of three maps which have adjacent “1”s or adjacent “0”s. In the example only one group two of “1”s is circled for each application of rule 2.

Rule 2 aligns “1”s horizontally, rule 1 aligns them vertically. by combining the rules one can get larger groups that can be circled.

Larger State Tables

This rule may seem rather weak with only 2 to 4 state variables. Groupings of two on 3 and 4 variable Karnaugh maps are very common, and there is not much reuse of AND terms.

However with N variables, N-1 of the maps will group either “1”s or “0”s. Half of these on average will be groups of “1”s which will use an AND gate which can be shared.

Also, if there are two inputs, there are 4 next states, and 4 sister states to group.
Rule 3: Last Resort, Cuddle States With Same Output

Rules for State Assignment

Rule 3 (Weak Rule) usually.

States With the Same Output Should Cuddle

Same Beautiful State Table

<table>
<thead>
<tr>
<th>STATE</th>
<th>NEXT STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>R</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>G</td>
<td>E</td>
<td>K</td>
</tr>
<tr>
<td>H</td>
<td>R</td>
<td>K</td>
</tr>
<tr>
<td>J</td>
<td>d</td>
<td>K</td>
</tr>
</tbody>
</table>

Possible Placement

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>R</td>
<td>C</td>
</tr>
<tr>
<td>01</td>
<td>E</td>
<td>B</td>
</tr>
<tr>
<td>11</td>
<td>K</td>
<td>J</td>
</tr>
<tr>
<td>10</td>
<td>H</td>
<td>G</td>
</tr>
</tbody>
</table>

This rule only simplifies the output map.

Usually the outputs are less complicated than the next states.

Give the other rules priority.

It is usually obvious if this becomes the important rule.

Possible Placement

Output Table arranged as per K-Map

<table>
<thead>
<tr>
<th>STATE</th>
<th>OUTPUT</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
<td>J</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
<td>G</td>
</tr>
</tbody>
</table>

Rule 3: Last Resort, Cuddle States With Same Output

Rule 3 and Rule 4

Heuristic 3 (a low priority explanation)

Often the output logic is much simpler than the state changing logic. Suppose 2 out of 16 states have an output. Then the most complex possible Moore output logic, ignoring inverters, is shown. The state changing logic is normally much more complex.

Try to simplify the state-changing logic first.

It is usually obvious if the output logic is important.

Also simplifying the state changing logic will often simplify the output logic.

In general, leave rule 3 till last.

Heuristic 4 (a low priority explanation)

This table and two-node graph illustrate another rule which may help. It should have a priority of about 2.5.

States which are each others parent (and child) for the same input should be adjacent.

27. PROBLEM

Draw three Karnaugh maps for the example of Rule 4, showing how the logic can simplify if C and R are adjacent.

Explain in general, for a 16 state machine:

What the maximum number of “1”s that can be circled because of this rule, on any one map?

This circle will be possible on how many maps? Give a minimum and maximum number.

What the maximum number of “0”s that can be circled because of this rule, on any one map?

This circle will be possible on how many maps? Give a minimum and maximum number.

(Graph on right) Assignment Graph for state tables and circuits at bottom of next page.
Guideline c:: Cuddle on Any Side (But Don’t Change Lovers)

Guideline c:::

It Doesn’t Matter on Which Side You Cuddle
Only Whom You Cuddle With

Variables switched

Q0 Q1 Q2
0 0 0
0 1 0
1 0 1
1 1 0

Q0 Q1 Q2
0 0 0
0 1 0
1 0 1
1 1 0

Check, the colors of your neighbors is always the same
Once you decide which states are adjacent, all assignments give the same logic circuit.

Five of six equivalent assignments with R=000. All adjacencies are maintained (Assignment graph on page 25)

28.** Problem:** Show the 6th equivalent assignment on a K-map.
Applying the Rules

Using All the Rules at Once

Make an Assignment Graph Showing all the Rules.

<table>
<thead>
<tr>
<th>State Table</th>
<th>Rule 1 Coloring</th>
<th>State Table</th>
<th>Rule 2 Coloring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td><strong>Next State X=0</strong></td>
<td><strong>State</strong></td>
<td><strong>Next State X=1</strong></td>
</tr>
<tr>
<td>R</td>
<td>B</td>
<td>R</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>G</td>
<td>B</td>
<td>G</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>E</td>
<td>B</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>G</td>
<td>K</td>
<td>G</td>
<td>K</td>
</tr>
<tr>
<td>H</td>
<td>R</td>
<td>H</td>
<td>R</td>
</tr>
<tr>
<td>J</td>
<td>d</td>
<td>J</td>
<td>d</td>
</tr>
</tbody>
</table>

Assign R=000 (guideline a)

Fill in map from assignment graph

Rule 1 used 2+4 times
Rule 2 used 5 times
Rule 3 is shown only if there is no other line.

Diagonal lines will have to be rearranged again if their assignment are to be adjacent on the K-map.

With rule 1 is completed, place and rearrange states for rule 2. Again try to avoid diagonal lines.

29. PROBLEM

Take the assignment graph above and do another state assignment different from the two given in the notes.

a) Put the state symbols, R, B, C ... on a map, showing heavy light and dashed lines according to the rules used for placement.

Follow the convention that R should be $Q_2Q_1Q_0=000$.

b) Make the combination state-table/K-map showing the assignment. It should have letters for the present state columns and bits in the next-state column. Don’t forget to interchange the $Q_2X$ columns where appropriate so K-map order will be preserved.

c) Make the 3 Karnaugh maps and circle them.

d) Draw the circuit without input letters or inverting circles in order to get a gate count. Share AND gates if it will reduce the number of gates.

e) Give a gate count, a gate-input count and a letter count for the logic equations. Recall that a letter count is the number of letters on the right hand side of the equations.

If you have $A=Q_2Q_1$, $D_2= A+ Q_2Q_0$, $D_1= A + Q_1$, $D_0=Q_0$ you have 8 letters.
Applying the Rules

Using All The Rules at Once.

Rule 1 is the most important.

Use that first. It may give enough information to make all the assignments.

Hint: Just as on a Karnaugh map, three states can never be close to each. However if there is a don’t care state, one can assign that as a fourth state so that the the other three can cuddle with the “d” state.

Rule 2 is used:

- When Rule 1 does not assign all the states.
- To break ties when Rule 1 can do an assignment two ways.

Rule 3 is often not needed

Use Rule 3 when Rules 1 and 2 do not give enough information.

30. • Problem

Show that the state assignment on the right can implement the problem on the slides with 10 gates and under 20 letters using one common AND gate.
Using All the Rules at Once

Another Assignment

State Table / K-Map

Assign \( R=000 \) (guideline a)

Complete map from assignment graph

Fill in table placing states in map order

Using All the Rules at Once

Other Assignments

Another good state assignment.

The best assignment found to date. Note the letter count is the sum of the letters on the right hand side of all the expressions. This is the solution to a previous problem.

Conclusion

- The rules help, but do not work too hard trying to match all the rules.
- An assignment that does not match quite as many rules may work just as well or better.
- There is no good method known to get the best, or even close to the best, state assignment.\(^1\)

\(^1\) Nelson, Nagle, Carroll and Irwin, *Digital Circuit Analysis & Design*, Prentice Hall, 1995, have a slightly better, but more complex method using the staircase (implicant) table.

© John Knight
Applying the Rules; An Alternate Assignment

Using All the Rules at Once (Cont)
Maps and Circuit

Using All the Rules at Once

31. PROBLEM (CIRCUIT ABOVE)
Why is the $Q_2Q_1Q_0$ term not worth sharing?

32. PROBLEM
On the right is a solution to Prob 23 in the Product State Machine Section. It has the minimum number of states.
a) Determine which states should have adjacent assignments and tell under which rule.
b) Make an assignment graph. Show heavy connections for rule 1 and lighter ones for rule 2.
c) Complete a state assignment starting with $R=000, A=100, B=001$.
   (Hint: a “d” state can make a threesome into a foursome)
d) Draw the Karnaugh maps and circle them.
e) Count the number of letters needed.

33. PROBLEM Repeat PROBLEM 32 using a state graph with an extra state. See page 37.

34. PROBLEM
Summary and Conclusions

Summary:
1. Parents of a child (for same input) should cuddle.
2. Sisters should cuddle.
3. If rules 2) and 3) are inconclusive, states with the same output should cuddle.
   a) Decide what to do with unused states.
   b) It doesn't matter where you start.
      Put Reset at 000...
   c) When you cuddle, it doesn't matter on which side.

Conclusions:
The Rules are useful but far from perfect.

Using All the Rules at Once

Show that the assignment below is the assignment on Slide 14 made by interchanging $Q_1 \leftrightarrow Q_0$ and rearranging the State-table/K-map back into Karnaugh map order. Further show the logic is exactly the same as on Slide 15 and hence this is an equivalent state assignment for logic minimization purposes.

<table>
<thead>
<tr>
<th>ASSIGNMENT</th>
<th>STATE</th>
<th>NEXT STATE</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC H K J</td>
<td>$R=000$</td>
<td>$B=110$</td>
<td>$C=100$</td>
</tr>
<tr>
<td>RC H K J</td>
<td>$R=001$</td>
<td>$B=010$</td>
<td>$C=010$</td>
</tr>
<tr>
<td>RC H K J</td>
<td>$R=011$</td>
<td>$B=110$</td>
<td>$C=101$</td>
</tr>
<tr>
<td>RC H K J</td>
<td>$R=111$</td>
<td>$B=110$</td>
<td>$C=101$</td>
</tr>
</tbody>
</table>

Some other assignments for the assignment graph on page 28.
Using All the Rules at Once

---

**Assignment Graph** for above circuits

**State Graph** for Problem 33