Redundant States in Sequential Circuits

Removal of redundant states is important because

- **Cost**: the number of memory elements is directly related to the number of states
- **Complexity**: the more states the circuit contains, the more complex the design and implementation becomes
- **Aids failure analysis**: diagnostic routines are often predicated on the assumption that no redundant states exist
✓ State minimization

- DEFINITION: Two states $S_i$ and $S_j$ are said to be equivalent if and only if for every input sequence, the same output sequence will be produced regardless of whether $S_i$ or $S_j$ are the initial states.

IF STATES $S_i$ AND $S_j$ ARE EQUIVALENT, THEN THEIR CORRESPONDING K-SUCCESSORS (FOR ALL K) ARE THE SAME OR ARE ALSO EQUIVALENT.
Equivalent States

- States $S_1, S_2, \ldots, S_j$ of a completely specified sequential circuit are said to be equivalent if and only if, for every possible input sequence, the same output sequence is produced by the circuit regardless of whether $S_1, S_2, \ldots, S_j$ is the initial state.

- Let $S_i$ and $S_j$ be states of a completely specified sequential circuit. Let $S_k$ and $S_l$ be the next states of $S_i$ and $S_j$, respectively for input $I_p$.

  $S_i$ and $S_j$ are equivalent if and only if for every possible $I_p$ the following conditions are satisfied:
  - the outputs produced by $S_i$ and $S_j$ are the same;
  - the next states $S_k$ and $S_l$ are equivalent.

States SZ and SY are equivalent and are combined to one state by pointing all arrows that go to SY to state SZ and removing SY with its all arrows.
Equivalence and Compatible Relations

- **Equivalence relation**: Let $R$ be a relation on a set $S$. $R$ is an equivalence relation on $S$ if and only if it is reflexive, symmetric, and transitive.

- An equivalence relation on a set partitions the set into disjoint equivalence classes.

- **Theorem**: State equivalence in a sequential circuit is an equivalence relation on the set of states.
Finding Equivalent States By Inspection

(a) 
\[
\begin{array}{c|cc}
 x & 0 & 1 \\
 \hline
 A & B/0 & C/1 \\
 B & C/0 & A/1 \\
 C & D/1 & B/0 \\
 D & C/0 & A/1 \\
\end{array}
\]

(b) 
\[
\begin{array}{c|cc}
 x & 0 & 1 \\
 \hline
 A & B/0 & C/1 \\
 B & C/0 & A/1 \\
 C & D/1 & B/0 \\
 D & D/0 & A/1 \\
\end{array}
\]

(c) 
\[
\begin{array}{c|cc}
 x & 0 & 1 \\
 \hline
 A & B/0 & C/1 \\
 B & B/0 & A/1 \\
 C & D/1 & B/0 \\
 D & D/0 & A/1 \\
\end{array}
\]

(d) 
\[
\begin{array}{c|cc}
 x & 0 & 1 \\
 \hline
 A & B/0 & C/1 \\
 B & B/0 & A/1 \\
 C & B/1 & B/0 \\
\end{array}
\]

(e) 
\[
\begin{array}{c|cc}
 x & 0 & 1 \\
 \hline
 A & B/0 & C/1 \\
 B & D/0 & A/1 \\
 C & D/1 & B/0 \\
 D & B/0 & A/1 \\
\end{array}
\]
A partition consists of one or more blocks, where each block comprises a subset of states that may be equivalent, but the states in a given block are definitely not equivalent to the states in the other blocks.

The partitioning method initially assumes that all states are equivalent and then proceeds to determine those state which are not equivalent by analyzing each states k-successors.
PROCEDURE:

1) all states belong to the initial partition $p_1$
2) $p_1$ is partitioned in blocks such that the states in each block generate the same output.
3) continue to perform new partitions by testing whether the $k$-successors of the states in each block are contained in one block. Those states whose $k$-successors are in different blocks cannot be in one block.
4) procedure ends when a new partition is the same as the previous partition
## Finding Equivalent States by Partitioning

<table>
<thead>
<tr>
<th>Partition blocks</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition $P_0$</td>
<td>(ABCDE)</td>
</tr>
<tr>
<td>Output for $x = 0$</td>
<td>11100</td>
</tr>
<tr>
<td>Output for $x = 1$</td>
<td>00011</td>
</tr>
<tr>
<td>Separate (ABC) and (DE)</td>
<td></td>
</tr>
<tr>
<td>Separate (ABC) and (DE)</td>
<td></td>
</tr>
<tr>
<td>Partition $P_1$</td>
<td>(ABC)</td>
</tr>
<tr>
<td>Next state for $x = 0$</td>
<td>CCB</td>
</tr>
<tr>
<td>Next state for $x = 1$</td>
<td>BEE</td>
</tr>
<tr>
<td>Separate (A) and (BC)</td>
<td></td>
</tr>
<tr>
<td>Partition $P_2$</td>
<td>(A)</td>
</tr>
<tr>
<td>Next state for $x = 0$</td>
<td>C</td>
</tr>
<tr>
<td>Next state for $x = 1$</td>
<td>B</td>
</tr>
<tr>
<td>Separate (D) and (E)</td>
<td></td>
</tr>
<tr>
<td>Partition $P_3$</td>
<td>(A)</td>
</tr>
<tr>
<td>Next state for $x = 0$</td>
<td>C</td>
</tr>
<tr>
<td>Next state for $x = 1$</td>
<td>B</td>
</tr>
<tr>
<td>Partition $P_4 = P_3$</td>
<td>(A)</td>
</tr>
</tbody>
</table>

States $B$ and $C$ are equivalent.
**Equivalent States: Implication Table**

- **completely specified state table**

  If \( c \) and \( d \) are equivalent, then
  \( a \) and \( b \) are equivalent \((a, b) \Leftrightarrow (c, d)\)

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X=0</td>
<td>X=1</td>
</tr>
<tr>
<td>( a )</td>
<td>( c )</td>
<td>( b )</td>
</tr>
<tr>
<td>( b )</td>
<td>( d )</td>
<td>( a )</td>
</tr>
<tr>
<td>( c )</td>
<td>( a )</td>
<td>( d )</td>
</tr>
<tr>
<td>( d )</td>
<td>( b )</td>
<td>( d )</td>
</tr>
</tbody>
</table>

\((c, d) \Leftrightarrow (a, b)\)  \implies  \((c, d) \land (a, b)\)

both pairs are equivalent
### Implication Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x = 0$</td>
<td>$x = 1$</td>
</tr>
<tr>
<td>$x = 0$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$x = 1$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td>$x = 0$</td>
<td>$c$</td>
<td>$f$</td>
</tr>
<tr>
<td>$x = 1$</td>
<td>$d$</td>
<td>$d$</td>
</tr>
<tr>
<td>$x = 0$</td>
<td>$e$</td>
<td>$d$</td>
</tr>
<tr>
<td>$x = 1$</td>
<td>$f$</td>
<td>$b$</td>
</tr>
<tr>
<td>$x = 0$</td>
<td>$g$</td>
<td>$e$</td>
</tr>
</tbody>
</table>

- **the equivalent states**
  - $(a,b), (d,e), (d,g), (e,g)$
- **the reduced states**
  - $(a,b), (c), (d,e,g), (f)$
- **the state table:**

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x = 0$</td>
<td>$x = 1$</td>
</tr>
<tr>
<td>$x = 0$</td>
<td>$a$</td>
<td>$d$</td>
</tr>
<tr>
<td>$x = 1$</td>
<td>$b$</td>
<td>$f$</td>
</tr>
<tr>
<td>$x = 0$</td>
<td>$c$</td>
<td>$d$</td>
</tr>
<tr>
<td>$x = 1$</td>
<td>$d$</td>
<td>$c$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>$b$</th>
<th>$d, e\checkmark$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$d$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$e$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$f$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$g$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>
Incompletely specified circuits

✓ Finding compatible states by inspection
The state table may be incompletely specified: combinations of inputs or input sequences may never occur (Some next states and outputs are don’t care).

Multi-input primitive flow tables are always incompletely specified
  • Several synchronous circuits also have this property

Incompletely specified states are not “equivalent” as in completely specified circuits. Instead, we are going to find compatible states
Equivalent and Compatible States

- **completely specified state table** \( \Rightarrow \) **equivalence**
  
  If \( a \) and \( b \) are equivalent, and \( b \) and \( c \) are equivalent then also \( a \) and \( c \) are equivalent:
  
  \((a, b), (b, c) \Rightarrow (a, c)\)

- **uncompletely specified state table** \( \Rightarrow \) **compatibility**
  
  If \( a \) and \( b \) are compatible, and \( b \) and \( c \) are compatible not necessarily \( a \) and \( c \) are compatible:

**Compatibility relation:** let \( R \) be a relation on a set \( S \). \( R \) is a compatibility relation on \( S \) if and only if it is reflexive and symmetric. A compatibility relation on a set partitions the set into compatibility classes. They are typically not disjoint.
The partitioning minimization procedure which was applied to completely specified state tables can also be applied to incompletely specified state tables.

To perform the partitioning process, we can assume that the unspecified outputs have a specific value.

The partitioning method is equally applicable to Mealy type FSMs in the same way as for Moore-type FSMs.
Partition minimization method

Example of incompletely specified FSM.

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Output $z$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$w = 0$</td>
<td>$w = 1$</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
<td>G</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>G</td>
<td>F</td>
<td>-</td>
</tr>
</tbody>
</table>

Let consider both unspecified outputs = 0

$P_1 = (ABCDEFG)$

$P_2 = (ABD)(CEF)$

$P_3 = (AB)(D)(G)(CEF)$


$P_5 = P_4$

Let consider both unspecified outputs = 1

$P_1 = (ABCDEFG)$

$P_2 = (AD)(BCEF)(G)$

$P_3 = (AD)(B)(CEG)(F)$

$P_4 = (AD)(B)(CEG)(F)$
Compatible Pairs (DG sequential circuit)

✓ Implication tables are used to find compatible states.
  • We can adjust the dashes to fit any desired condition.
  • Must have no conflict in the output values to be merged.

(a) Primitive flow table
(b) Implication table

The compatible pairs are:
(a, b)
(a, c)
(a, d)
(b, e)
(b, f)
(c, d)
(e, f)
Merger diagrams

- States are represented as dot in a circle
- Lines connect states couples compatible
- Maximal sets can be identified as those sets in which every states is connected to every other state by a line segment
Maximal Compatibles

✓ A group of compatibles that contains all the possible combinations of compatible states.

✓ n-state compatible \( \Rightarrow \) n-sided fully connected polygon.
  - All its diagonals connected.
  - Not all maximal compatibles are necessary
  - An isolated dot: a state that is not compatible to any other state

• a line: a compatible pair

• a triangle: a compatible with three states

• an n-state compatible: an n-sided polygon with all its diagonals connected
Closed Covering Condition

- The condition that must be satisfied is that the set of chosen compatibles must:
  - Cover all states.
  - Be closed: the closure condition is satisfied if there are no implied states or if the implied states are included within a set.

- In the example of the DG sequential circuit, the maximal compatibles are:
  
  \[(a, b) (a, c, d), (b, e, f)\]

- If we remove \( (a, b) \), we get a set of two compatibles:
  
  \[(a, c, d), (b, e, f)\]

  - All the six states are included in this set.
  - There are no implied states for \((a,c); (a,d);(c,d);(b,e);(b,f)\) and \((e,f)\) [you can check the implication table]. The closer condition is satisfied.

The original primitive flow table can be merged into two rows, one for each of the compatibles.
Closed Covering Condition (Example)

- From the aside implication table, we have the following compatible pairs: 
  \( (a, b) \) \( (a, d) \) \( (b, c) \) \( (c, d) \) \( (c, e) \) \( (d, e) \)
- From the merger diagram, we determine the maximal compatibles: 
  \( (a, b) \) \( (a, d) \) \( (b, c) \) \( (c, d, e) \)
- If we choose the two compatibles: \( (a, b) \) \( (c, d, e) \)
  - All the 5 states are included in this set.
  - The implied states for \( (a, b) \) are \( (b, c) \). But \( (b, c) \) are not included in the chosen set. This set is not closed.
  - A set of compatibles that will satisfy the closed covering condition is \( (a, d) \) \( (b, c) \) \( (c, d, e) \)
  - Note that: the same state can be repeated more than once

<table>
<thead>
<tr>
<th>Compatibles</th>
<th>( (a, b) )</th>
<th>( (a, d) )</th>
<th>( (b, c) )</th>
<th>( (c, d, e) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied states</td>
<td>( (b, c) )</td>
<td>( (b, c) )</td>
<td>( (d, e) )</td>
<td>( (a, d) )</td>
</tr>
</tbody>
</table>

Closure table
Minimization of compatible classes

Select a set of compatibility classes so that the following conditions are satisfied:

- **Completeness**: all states of the original machine must be covered
- **Consistency**: the chosen set of compatibility classes must be closed
- **Minimality**: the smallest number of compatibility classes is used
Bounding the number of states

- Unfortunately the process of selecting the set of compatibility classes that meets the three conditions must be done by trial and error.

- Let $U$ be the upper bound on the number of states needed in the minimized circuit. Then $U = \text{minimum} (\text{NSMC}, \text{NSOC})$
  - where NSMC = the number of sets of maximal compatibles
  - and NSOC = the number of states in the original circuit

- Let $L$ be the lower bound on the number of states needed in the minimized circuit. Then $L = \text{maximum}(\text{NSMI}_1, \text{NSMI}_2, \ldots, \text{NSMI}_i)$
  - where NSMI$_i$ = the number of states in the $i^{th}$ group of the set of maximal incompatibles of the original circuit.
State Reduction Algorithm

✓ Step 1 -- find the maximal compatibles
✓ Step 2 -- find the maximal incompatibles
✓ Step 3 -- Find the upper and lower bounds on the number of states needed
✓ Step 4 -- Find a set of compatibility classes that is complete, consistent, and minimal
✓ Step 5 -- Produce the minimum state table
Example -- State reduction problem

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A/-</td>
<td>-/-</td>
</tr>
<tr>
<td>B</td>
<td>C/1</td>
<td>B/0</td>
</tr>
<tr>
<td>C</td>
<td>D/0</td>
<td>-/1</td>
</tr>
<tr>
<td>D</td>
<td>-/-</td>
<td>B/-</td>
</tr>
<tr>
<td>E</td>
<td>A/0</td>
<td>C/1</td>
</tr>
</tbody>
</table>

Closure table

Compatibles pairs

Incompatibles pairs
(BC)(BE)(DE)

Maximum Compatibles
(ABD)(ACD)(ACE)

U = 3

Maximum Incompatibles
(BE)(BC)(ED)

L = 2

A' = (ABD)
B' = (ACE)

Closure table

Reduced state table
Unique State Assignments

### NUMBER OF STATE ASSIGNMENTS

<table>
<thead>
<tr>
<th>$N_S$</th>
<th>$N_{FF}$</th>
<th>$N_{SA}$</th>
<th>$N_{UA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>6,720</td>
<td>140</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>20,160</td>
<td>420</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>40,320</td>
<td>840</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>40,320</td>
<td>840</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>$4.15 \times 10^9$</td>
<td>10,810,800</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>$2.91 \times 10^{10}$</td>
<td>75,675,600</td>
</tr>
</tbody>
</table>

\[
N_{UA} = \frac{(2^{N_{FF}} - 1)!}{(2^{N_{FF}} - N_S)!N_{FF}!}
\]

#### Present state and Assignments

<table>
<thead>
<tr>
<th>Present state</th>
<th>(x)</th>
<th>Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>0</td>
<td>(A/0) (B/0)</td>
</tr>
<tr>
<td>(B)</td>
<td>0</td>
<td>(A/0) (C/0)</td>
</tr>
<tr>
<td>(C)</td>
<td>0</td>
<td>(C/0) (D/0)</td>
</tr>
<tr>
<td>(D)</td>
<td>1</td>
<td>(C/1) (A/0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>States</th>
<th>1 (y_1 y_2)</th>
<th>2 (y_1 y_2)</th>
<th>3 (y_1 y_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>00</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>(B)</td>
<td>01</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>(C)</td>
<td>11</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>(D)</td>
<td>10</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

\[
D_1 = y_1 x + y_2 x \\
D_2 = y_1 x + y_1 x \\
D_1 = y_2 x + y_2 x \\
D_2 = y_2 x + y_1 x
\]
Another state table reduction problem

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B/1</td>
<td>D/0</td>
</tr>
<tr>
<td>B</td>
<td>--</td>
<td>B/0</td>
</tr>
<tr>
<td>C</td>
<td>E/0</td>
<td>D/0</td>
</tr>
<tr>
<td>D</td>
<td>B/1</td>
<td>A/0</td>
</tr>
<tr>
<td>E</td>
<td>--</td>
<td>C/1</td>
</tr>
<tr>
<td>F</td>
<td>--</td>
<td>E/1</td>
</tr>
</tbody>
</table>

(a)

Compatibles pairs
$(AB)(AD)(BC)(BD)$

U = 4

(b)

Incompatibles pairs

L = 4

(c)

Maximum Compatibles
$(ABD)(C)(E)(F)$

U = 4

(d)

Note that the resultant state table has considerable flexibility. This property will serve to simplify the hardware realization.

(e)

(f)

Reduced state table
Yet another state reduction problem

Maximum Compatibles
(ABC)(ACD)(ADE)
U=3

Maximum Incompatibles
(BD)(BE)(CE)
L=2

Closure table
Maximum Compatible

Closure table

Reduced state table
# Generating Maximal Composables and Incompatibles

## Maximum Composables

\[(AEGH)(BCG)(CDG)\]
\[(CEG)(CFG)\]
\[U = 5\]

## Maximum Incompatibles

\[(ABDF)(AC)(BDEF)\]
\[(CH)(BDFH)\]
\[L = 4\]
Reduced state table

✓ Closure table: treat maximum compatibles as states and find their sets of next states

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
(AEGH) & AGH & CDG \\
(BCG)   & BG  & AEG \\
(CDG)   & CG  & CEG \\
(CEG)   & AG  & CEG \\
(CFG)   & DG  & AEG \\
\end{array}
\]

\[
\begin{array}{ccc}
A' & A'/1 & C'/1 \\
B' & B'/- & A'/0 \\
C' & B', C', D', E'/1 & D'/0 \\
D' & A'/1 & D'/0 \\
E' & C'/- & A'/0 \\
\end{array}
\]

Closure table  Reduced state table

✓ All maximal compatibles are used as states of the reduced machine. Hence, the final five states are:

\[
\begin{align*}
A' &= (AEGH), & D' &= (CEG) \\
B' &= (BCG), & E' &= (CFG) \\
C' &= (CDG) \\
\end{align*}
\]
State Assignment

✓ Primary Objective of Synchronous Networks
  • Simplification of Logic and Improvement of Performance
  • Improvement of Testability
  • Minimization of Power Consumption.

✓ Primary Objective of Asynchronous Networks
  • Prevention of Critical Races
  • Simplification of Logic
Race-Free State Assignment

- **Objective:** choose a proper binary state assignment to prevent critical races
- **Only one variable can change** at any given time when a state transition occurs
- **States between which transitions occur will be given adjacent assignments**
  - Two binary values are said to be adjacent if they differ in only one variable
- **To ensure that a transition table has no critical races, every possible state transition should be checked**
  - A tedious work when the flow table is large
  - Only 3-row and 4-row examples are demonstrated
Three states require two binary variables (in the flow table outputs are omitted for simplicity)

Representation by a transition diagram

a and c are not adjacent in such an assignment!

• Impossible to make all states adjacent if only 3 states are used
3-Row Flow-Table Example

✓ A race-free assignment can be obtained if we add an extra row to the flow table
✓ Only provide a race-free transition between the stable states
✓ The transition from a to c must now go through d
  \[00 \Rightarrow 10 \Rightarrow 11\] (no race condition)
✓ Note that no stable state can be introduced in row d

<table>
<thead>
<tr>
<th></th>
<th>(x_1x_2)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(a)</td>
<td>b</td>
<td>d</td>
<td>(a)</td>
</tr>
<tr>
<td>b</td>
<td>(a)</td>
<td>b</td>
<td>((b))</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>c</td>
<td>((c))</td>
<td>((c))</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>–</td>
<td>c</td>
<td>–</td>
</tr>
</tbody>
</table>

\[d = 10 \Rightarrow c = 11\]

\[a = 00 \Rightarrow b = 01\]

<table>
<thead>
<tr>
<th></th>
<th>(x_1x_2)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(a)</td>
<td>(00)</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>(00)</td>
<td>01</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>c</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>d</td>
<td>00</td>
<td>–</td>
<td>11</td>
<td>–</td>
</tr>
</tbody>
</table>
Shared-Row Method

- The shared row is not assigned to any specific stable state
- Used to convert a critical race into a cycle that goes through adjacent transitions between two stable states
- May require more extra rows
State adjacencies for assignments

**General rules:**

**Rule 1.** States that have the same next states for a given input should be given logically adjacent assignments.

**Rule 2.** States that are the next states of a single present state, under logically adjacent inputs, should be given logically adjacent assignments.
4-Row Flow-Table Example

- A flow table with 4 states requires an assignment of two state variables.
- If there were no transitions in the diagonal direction (from a to c or from b to d), it would be possible to find adjacent assignment for the remaining 4 transitions.

- In general in order to satisfy the adjacency requirement, at least 3 binary variables are needed.
4-Row Flow-Table Example

• The following state assignment map is suitable for any table.
  • a, b, c, and d are the original states.
  • e, f, and g are extra states.
  • States placed in adjacent squares in the map will have adjacent assignments.
  • Please note that state variable order in figures is $y_3y_1y_2$. 

![Diagram](image)
4-Row Flow-Table Example

To produce cycles:

- The transition from a to d must be directed through the extra state e.
- The transition from c to a must be directed through the extra state g.
- The transition from d to c must be directed through the extra state f.

Although the flow table has 7 rows, there are only 4 stable states.
Multiple Row Method

- Multiple-row method is easier. May not as efficient as in above shared-row method.
- Each stable state is duplicated with exactly the same output. Behaviors are still the same.
- While choosing the next states, choose the adjacent one among the two possibilities.

(a) Binary assignment

(b) Flow table
### Don't Care Assignment

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td>-</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>d</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>

#### Needed Transitions

- Column 00: $d \rightarrow a$
- Column 01: $a \rightarrow b, c \rightarrow d$
- Column 11: $b \rightarrow c$
- Column 10: $a \rightarrow c, b \rightarrow d$

- All possible transitions between pairs of rows are needed
- Fill in don't care to eliminate races
- Direct transitions for columns 01, 10 (No don't care)
Don't Care Assignment

- All possible transitions between pairs of rows are needed
- Fill in don't care to eliminate races
- Direct transitions for columns 01, 10 (No don't care)

In column 00: \(d \xrightarrow{\text{a}}\) changed to \(d \xrightarrow{\text{c}}\xrightarrow{\text{a}}\).
In column 11: \(b \xrightarrow{\text{c}}\) changed to \(b \xrightarrow{\text{a}}\xrightarrow{\text{c}}\).
State Assignment

✓ Universal Assignment for 8-Row Tables
  • Require 4 state variables.
  • Slow due to several successive changes needed.

✓ Universal Assignment as the Last Resort.
  • Try to use smaller number of state variables.
  • Try to take advantage of don’t care.
  • Allow several state variables change simultaneously.
    ▪ Make all races noncritical.
    ▪ Faster.
Transition changes

<table>
<thead>
<tr>
<th></th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>C, 1</td>
<td>B, 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>C, 1</td>
<td>B, 0</td>
<td>A, 1</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>C, 1</td>
<td>A, -</td>
<td>C, 0</td>
<td>D, 0</td>
</tr>
<tr>
<td>d</td>
<td>E, 0</td>
<td>D, 1</td>
<td>C, 0</td>
<td>D, 0</td>
</tr>
<tr>
<td>e</td>
<td>E, 0</td>
<td>D, 1</td>
<td>F, -</td>
<td>E, 1</td>
</tr>
<tr>
<td>f</td>
<td>D, -</td>
<td>D, 1</td>
<td>B, -</td>
<td>F, 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>C, 1</td>
<td>B, 0</td>
<td>A, 1</td>
<td>C, -</td>
</tr>
<tr>
<td>b</td>
<td>A', 1</td>
<td>B, 0</td>
<td>A, 1</td>
<td>A, -</td>
</tr>
<tr>
<td>c</td>
<td>C, 1</td>
<td>A, -</td>
<td>C, 0</td>
<td>D, 0</td>
</tr>
<tr>
<td>d</td>
<td>E, 0</td>
<td>D, 1</td>
<td>C, 0</td>
<td>D, 0</td>
</tr>
<tr>
<td>e</td>
<td>E, 0</td>
<td>D, 1</td>
<td>F, -</td>
<td>E, 1</td>
</tr>
<tr>
<td>f</td>
<td>E', -</td>
<td>D, 1</td>
<td>B, -</td>
<td>F, 0</td>
</tr>
</tbody>
</table>

Graphs and transition diagrams illustrating the transitions.
### Extra states

<table>
<thead>
<tr>
<th></th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
<th>I₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>C,1</td>
<td>B,0</td>
<td>A,1</td>
<td>C,-</td>
</tr>
<tr>
<td>b</td>
<td>C,1</td>
<td>B,0</td>
<td>A,1</td>
<td>A,-</td>
</tr>
<tr>
<td>c</td>
<td>C,1</td>
<td>A,-</td>
<td>C,0</td>
<td>D,0</td>
</tr>
<tr>
<td>d</td>
<td>E,0</td>
<td>D,1</td>
<td>C,0</td>
<td>D,0</td>
</tr>
<tr>
<td>e</td>
<td>E,0</td>
<td>D,1</td>
<td>H,-</td>
<td>E,1</td>
</tr>
<tr>
<td>f</td>
<td>D,-</td>
<td>D,1</td>
<td>B,-</td>
<td>F,0</td>
</tr>
<tr>
<td>g</td>
<td>-,-</td>
<td>B,-</td>
<td>A,-</td>
<td>-,-</td>
</tr>
<tr>
<td>h</td>
<td>-,-</td>
<td>-,-</td>
<td>F,-</td>
<td>-,-</td>
</tr>
</tbody>
</table>

![Diagram of states and transitions](image)

![Transition graph](image)
Summary

- Shared row method
- Multiple row method
- Don’t care assignment
- Universal states assignment
- Transition changes
- Other approaches

1 VP Nelson et al., not presented here
Hazards

✓ A timing problem arises due to gate and wiring delays
✓ **Hazards**: Unwanted switching transients at the network output, caused by input changes and due to different paths through the network from input to output that may have different propagation delays
✓ Hazards occur in combinational and asynchronous circuits:
  • In combination circuits, they may cause a temporarily false output value.
  • In asynchronous circuits, they may result in a transition to a wrong stable state.
✓ **Asynchronous Sequential Circuits**:
  • **Hypothesis**:
    ▪ Operated in fundamental mode with only one input changing at any time
  • **Objectives**:
    ▪ Free of critical races
    ▪ Free of hazards
Hazards in combinational circuits

✓ Static 1-hazard (sum of products)
  • The remedy
    ▪ the circuit moves from one product term to another
    ▪ additional redundant gate
Hazards in sequential circuits

✓ An asynchronous example:

\[ Y = x_1 x_2 + x'_2 y \]

(a) Logic diagram

(b) Transition table

(c) Map for \( Y \)

\[ 111 \rightarrow 110 \]
\[ 111 \rightarrow 010 \]
Remove Hazards with Latches

- The implementation of the asynchronous circuits with SR latches can remove static hazards.

- Note that a sum-of-products implementation is automatically free of static 0-hazards and a product-of-sums implementation is free of static 1-hazards.

- SR Latch (NOR type)
  - Allow 1-hazard (a momentary 0 has no effect)
  - The network realizing S and R must be free of 0-hazards.

- S’R’ Latch (NAND type)
  - Allow 0-hazard (a momentary 1 has no effect)
  - The network realizing S and R must be free of 1-hazards.
Hazard-Free Realization

✓ **S-R Latch (NOR type):**

S-R Flip-Flop Driven by 2-level AND-OR Networks

Equivalent Network Structure (in general faster)
Example

✓ Consider a SR-latch with the following Boolean functions for S and R
  
  \[
  S = AB + CD \\
  R = A'C'
  \]

✓ If we want to use a NAND latch we must complement the value for S and R
  
  \[
  S = (AB + CD)' = (AB)'(CD)' \\
  R = (A'C')'
  \]

✓ The Boolean function for output is
  
  \[
  Q = (Q'S)' = [Q' (AB)'(CD)']'
  \]

✓ The output is generated with two levels of NAND gates:
  
  A product-of-sums implementation is automatically free of static 1-hazards.

If output Q is equal to 1, then Q’ is equal to 0. If two of the three inputs go momentarily to 1, the NAND gate associated with output Q will remain at 1 because Q’ is maintained at 0.
Essential Hazards

✔ Besides static and dynamic hazards, another type of hazard in asynchronous circuits is called: Essential Hazard

✔ It is caused by unequal delays along two or more paths that originate from the same input

✔ Cannot be corrected by adding redundant gates

✔ Can only be corrected by adjusting the amount of delay in the affected path
  • Each feedback path should be examined carefully!!
Recommended Design Procedure:

1. State the design specifications.
2. Derive a Primitive Flow Table.
3. Reduce the Flow Table by merging rows.
4. Make a race-free binary state assignment.
5. Obtain the transition table and output map.
6. Obtain the logic diagram using SR latches.
1) **Design Specifications:**

It is necessary to design a negative-edge-triggered T flip-flop. The circuit has two inputs T (toggle) and C (clock) and one output Q. The output state is complemented if T=1 and the clock changes from 1 to 0 (negative-edge-triggering). Otherwise, under all input condition, the output remains unchanged.

- A Negative-Edge-Triggered T FF
- Two inputs: T, C
- Flip-Flop changes state when T = 1 and C changes from 1 to 0
- Q remains constant under all other conditions
- T and C do not change simultaneously
Design Example

2) Primitive Flow Table

<table>
<thead>
<tr>
<th>State</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T$</td>
<td>$C$</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$e$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$g$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$h$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>$T$</th>
<th>$C$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$f$,</td>
<td>$a$</td>
<td>$b$,</td>
</tr>
<tr>
<td>$b$</td>
<td>$g$,</td>
<td>$c$</td>
<td>$b$,</td>
</tr>
<tr>
<td>$c$</td>
<td>$h$,</td>
<td>$c$</td>
<td>$d$,</td>
</tr>
<tr>
<td>$d$</td>
<td>$e$,</td>
<td>$a$,</td>
<td>$d$,</td>
</tr>
<tr>
<td>$e$</td>
<td>$f$,</td>
<td>$d$,</td>
<td>$d$,</td>
</tr>
<tr>
<td>$f$</td>
<td>$e$,</td>
<td>$a$,</td>
<td>$a$,</td>
</tr>
<tr>
<td>$g$</td>
<td>$g$,</td>
<td>$h$,</td>
<td>$h$,</td>
</tr>
<tr>
<td>$h$</td>
<td>$g$,</td>
<td>$c$,</td>
<td>$c$,</td>
</tr>
</tbody>
</table>
Design Example

3) Merging of the Flow Table

Implication Table

<table>
<thead>
<tr>
<th></th>
<th>a, c</th>
<th>b, d</th>
<th>c, e</th>
<th>d, g</th>
<th>e, h</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>b</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>c</td>
<td>b, d</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>d</td>
<td>b, d</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>e</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>f</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>g</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>h</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Compatible pairs:

(a, f) (b, g) (b, h) (c, h) (d, e) (d, f) (e, f) (g, h)

Merger Diagram

The maximal compatibles pairs are:

(a, f) (b, g) (b, h) (c, h) (d, e, f)

U = 4
Maximal Incompatibles

L=4
In this particular example, the minimal collection of compatibles is also the maximal compatibles set that satisfy also the closed condition:

\[(a, f) \quad (b, g, h) \quad (c, h) \quad (d, e, f)\]
4) State Assignment and Transition Table

- No diagonal lines in the transition diagram: No need to add extra states (race-free binary state assignment!)
5) Logic Diagram

(a) $S_1 = y_2 \text{ } TC + y'_2 \text{ } T'C'$

(b) $R_1 = y_2 \text{ } T'C' + y'_2 \text{ } TC$

(c) $S_2 = y'_1 \text{ } TC'$

(d) $R_2 = y_1 \text{ } TC'$