Asynchronous Circuit Design

Huffman Circuit

INPUTS

OUTPUTS

STATE

Comb. Logic

Delay

Circuit Delay Model

- Uses the bounded gate and wire delay model.
- Environment must also be constrained:
  - Single-input change (SIC) - each input change must be separated by a minimum time interval.
  - SIC fundamental mode - the time interval is the maximum delay for circuit to stabilize.
  - MIC - allow multiple inputs to change.
  - MIC fundamental mode - waits for circuit to stabilize.
  - Extended burst mode - limited form of MIC operation.

Solving Covering Problems

- The last step of state minimization, state assignment, and logic synthesis is to solve a covering problem.
- A covering problem exists whenever you must select a set of choices with minimum cost which satisfy a set of constraints.
- Classic example: selection of the minimum number of prime implicants to cover all the minterms of a given function.

Formal Derivation of Covering Problem

- Each choice is represented with a Boolean variable $x_i$.
- $x_i = 1$ implies choice has been included in the solution.
- $x_i = 0$ implies choice has not been included in the solution.
- Covering problem is expressed as a product-of-sums, $F$.
- Each product (or clause) represents a constraint.
- Each clause is sum of choices that satisfy the constraint.
- Goal: find $x_i$'s which satisfy all constraints with minimum cost.

$$
cost = \min \sum_{i=1}^{t} w_i x_i \quad (1)$$
Asynchronous Circuit Design

Example Covering Problem

\[ f = x_1 \overline{x}_2 (\overline{x}_3 + x_4)(\overline{x}_3 + x_4 + x_5 + x_6)(\overline{x}_4 + x_5 + x_6) \]

\( (\overline{x}_5 + x_1 + x_6)(\overline{x}_6 + x_5) \)

Unate versus Binate

- **Unate covering problem** - choices appear only in their positive form (i.e., uncomplemented).
- **Binate covering problem** - choices appear in both positive and negative form (i.e., complemented).
- Algorithm presented here considers the more general case of the binate covering problem, but solution applies to both.

Constraint Matrix

- \( f \) is represented using a constraint matrix, \( A \).
- Includes a column for each \( x_i \) variable.
- Includes a row for every clause.
- Each entry of the matrix \( a_{ij} \) is:
  - ‘.’ if the variable \( x_i \) does not appear in the clause.
  - ‘0’ if the variable appears complemented, and
  - ‘1’ otherwise.
- \( j^{th} \) row of \( A \) is denoted \( a_j \).
- \( j^{th} \) column is denoted by \( A_j \).

Constraint Matrix Example

- \( f = x_1 \overline{x}_2 (\overline{x}_3 + x_4)(\overline{x}_3 + x_4 + x_5 + x_6)(\overline{x}_4 + x_5 + x_6) \)

\[
A = \begin{bmatrix}
1 & - & - & - & - & 1 \\
- & 0 & - & - & - & 2 \\
- & - & 0 & 1 & - & 3 \\
1 & - & 0 & 1 & 1 & 4 \\
0 & - & 1 & 1 & 1 & 5 \\
1 & - & - & 0 & - & 1 \\
- & - & - & 0 & 1 & 7
\end{bmatrix}
\]

Binate Covering Problem

- The binate covering problem is to find an assignment to \( x \) of minimum cost such that for every row \( a_j \) either
  - \( \exists j : (a_j = 1) \land (x_j = 1) \); or
  - \( \exists j : (a_j = 0) \land (x_j = 0) \).

BCP Algorithm

- \( bcp(A, x, b) \)
  - \( (A, x) \) - reduce \( (A, x) \);
  - \( L = lower\_bound(A, x) \);
  - if \( (L \geq \text{cost}(b)) \) then return \( b \);
  - (terminalCase(A)) then
    - if (A has no rows) return \( x \); else return \( b \);
    - \( c = \text{choose\_column}(A) \);
    - \( x_c = 1 \);
    - \( A^1 = \text{select\_column}(A, c) \);
    - \( x^1 = bcp(A^1, x, b) \);
  - if \( (\text{cost}(x^1) < \text{cost}(b)) \) then
    - \( b = x^1 \);
  - if \( (\text{cost}(b) = L) \) return \( b \);
  - \( x_0 = 0 \);
  - \( A^0 = \text{remove\_column}(A, c) \);
  - \( x^0 = bcp(A^0, x, b) \);
  - if \( (\text{cost}(x^0) < \text{cost}(b)) \) then \( b = x^0 \);
  - return \( b \);
### Essential Rows

A row $a_i$ of $A$ is **essential** when there exists exactly one $j$ such that $a_{ij}$ is not equal to '-'.

- This corresponds to a clause consisting of a single literal.
- If the literal is $x_j$ (i.e., $a_j = 1$), the variable is essential.
- If the literal is $\overline{x_j}$ (i.e., $a_j = 0$), the variable is unacceptable.
- The matrix $A$ is reduced with respect to the essential literal.
- This variable is set to the value of the literal, column is removed, and any row where the variable has the same value is removed.

### Row Dominance

A row $a_k$ dominates another row $a_i$ if it has all 1's and 0's of $a_i$.

- Row dominance reduces the set of rows, but does not affect the set of solutions.

### Essential Rows Example

Given function $f = x_1 x_2 (x_3 + x_4)(x_3 + x_4 + x_5 + x_6)(x_4 + x_5 + x_6)(x_5 + x_6)$, find essential rows and dominant rows.

#### Essential Rows

An essential row $a_i$ of $A$ is when there exists exactly one $j$ such that $a_{ij}$ is not equal to '-'.

All rows where variable has same value are removed.

#### Row Dominance

A row $a_k$ dominates another row $a_i$ if it has all 1's and 0's of $a_i$.

#### Essential Rows

A row $a_k$ dominates another row $a_i$ if it has all 1's and 0's of $a_i$.

### Reduce Algorithm

```plaintext
reduce (A, x)
   do
      A' = A;
      (A, x) = find_essential_rows(A, x);
      A = delete_dominating_rows(A, x);
      if (A \neq \emptyset) and A \neq A'
         A = delete_dominated_columns(A);
   return (A, x);
```

### Example Matrix

<table>
<thead>
<tr>
<th></th>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>x_4</th>
<th>x_5</th>
<th>x_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Function $f$

$f = \overline{x_3} x_4 (\overline{x_3} + x_4 + x_5 + x_6)(\overline{x_4} + x_5 + x_6)(\overline{x_5} + x_6)$

<table>
<thead>
<tr>
<th>x_2</th>
<th>x_3</th>
<th>x_4</th>
<th>x_5</th>
<th>x_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Matrix $A$

$x_1 = 1$, $x_2 = 0$
Asynchronous Circuit Design

Row Dominance Example

\[ f = (x_3 + x_4)(x_4 + x_5 + x_6)(x_4 + x_5 + x_6) \]

\[ A = \begin{bmatrix}
  0 & 1 & 0 & 1 & 1 & 1 \\
  0 & 1 & 1 & 1 & 1 & 5 \\
  -1 & 1 & 1 & - & - & 7 \\
  x_1 = 1, x_2 = 0
\end{bmatrix} \]

Column Dominance Example

\[ f = (x_3 + x_4)(x_4 + x_5 + x_6)(x_4 + x_5 + x_6) \]

\[ A = \begin{bmatrix}
  0 & 1 & - & - \\
  -1 & 1 & 1 & 5 \\
  - & - & 0 & 1 \\
  x_1 = 1, x_2 = 0
\end{bmatrix} \]

Checking Weights

- If weights are not equal, it is necessary to also check the weights of the columns before removing dominated columns.
- If weight of dominating column, \( w_j \), is greater than weight of dominated column, \( w_k \), then \( x_j \) should not be removed.
- Assume \( w_1 = 3, w_2 = 1, \) and \( w_3 = 1 \).

\[ A = \begin{bmatrix}
  1 & 1 & - \\
  - & - & 0 & 1 \\
  x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0
\end{bmatrix} \]
Asynchronous Circuit Design

27 / 1

Asynchronous Circuit Design

25 / 1

Asynchronous Circuit Design

30 / 1

Asynchronous Circuit Design

28 / 1

Asynchronous Circuit Design

26 / 1

Chris J. Myers (Lecture 5: Huffman Circuits)

Bounding

- If solved, cost of solution can be determined by Equation 1.
- Reduced matrix may have a cyclic core.
- Must test whether or not a good solution can be derived from partial solution found up to this point.
- Determine a lower bound, L, on the final cost, starting with the current partial solution.
- If $L$ is greater than or equal to the cost of the best solution found, the previous best solution is returned.

Bounding Example

Bounding Example

Bounding Example

Bounding Example

BCP Algorithm

\[ bcp(A, x, b) \]

\[
\begin{align*}
(A, x) & = \text{reduce}(A, x); \\
L & = \text{lower\_bound}(A, x); \\
& \text{if } (L \geq \text{cost}(b)) \text{ then return } (b); \\
& \text{if } (\text{terminalCase}(A)) \text{ then return } (x); \\
& \text{else return } (b); \\
& c = \text{choose\_column}(A); \\
& x_c = 1; A^1 = \text{select\_column}(A, c); x^1 = bcp(A^1, x, b) \\
& \text{if } (\text{cost}(x^1) < \text{cost}(b)) \text{ then } b = x^1; \\
& \text{if } (\text{cost}(b) = L) \text{ return } (b); \\
& x_c = 0; A^0 = \text{remove\_column}(A, c); x^0 = bcp(A^0, x, b) \\
& \text{if } (\text{cost}(x^0) < \text{cost}(b)) \text{ then } b = x^0; \\
& \text{return } (b);
\end{align*}
\]

Minimal Independent Set

- Finding exact lower bound is as difficult as solving the covering problem.
- Satisfactory heuristic method is to find a maximal independent set (MIS) of rows.
- Two rows are independent when it is not possible to satisfy both by setting a single variable to 1.
- Any row which contains a complemented variable is dependent on any other clause, so we must ignore these rows.

Lower Bound Algorithm

\[ \text{lower\_bound}(A, x) \]

\[
MIS = \emptyset \\
A = \text{delete\_rows\_with\_complemented\_variables}(A); \\
d \text{do } \\
\quad i = \text{choose\_shortest\_row}(A); \\
\quad MIS = MIS \cup \{i\}; \\
\quad A = \text{delete\_intersecting\_rows}(A, i); \\
\text{while } (A \neq \emptyset); \\
\text{return } (|MIS| + \text{cost}(x));
\]
Asynchronous Circuit Design

Bounding Example

\[ A = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \end{bmatrix} \]

\[ \text{MIS} = \{1, 3\} \]

BCP Algorithm

\[ \text{bcp}(A, x, b) \]
\[ (A, x) = \text{reduce}(A, x); \]
\[ L = \text{lower_bound}(A, x); \]
\[ \text{if } (L \geq \text{cost}(b)) \text{ then return } (b); \]
\[ \text{if } (\text{terminalCase}(A)) \text{ then } \]
\[ \text{if } (A \text{ has no rows}) \text{ return } (x); \text{ else return } (b); \]
\[ c = \text{choose_column}(A); \]
\[ x_c = 1; A^1 = \text{select_column}(A, c); x^1 = \text{bcp}(A^1, x, b); \]
\[ \text{if } (\text{cost}(x^1) < \text{cost}(b)) \text{ then } \]
\[ b = x^1; \]
\[ \text{if } (\text{cost}(b) = L) \text{ return } (b); \]
\[ x_c = 0; A^0 = \text{remove_column}(A, c); x^0 = \text{bcp}(A^0, x, b); \]
\[ \text{if } (\text{cost}(x^0) < \text{cost}(b)) \text{ then } b = x^0; \]
\[ \text{return } (b); \]

Termination

- If \( A \) has no more rows, then all the constraints have been satisfied by \( x \), and it is a terminal case.
- If no solution exists, it is also a terminal case.

Bounding Example

\[ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9 \]

\[ \text{MIS} = \{1, 3, 6\} \]

Infeasible Problems

\[ f = (x_1 + x_2)(x_3 + x_4)(x_5 + x_6)(x_7 + x_8) \]

\[ A = \begin{bmatrix} x_1 & x_2 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \]

BCP Algorithm

\[ \text{bcp}(A, x, b) \]
\[ (A, x) = \text{reduce}(A, x); \]
\[ L = \text{lower_bound}(A, x); \]
\[ \text{if } (L \geq \text{cost}(b)) \text{ then return } (b); \]
\[ \text{if } (\text{terminalCase}(A)) \text{ then } \]
\[ \text{if } (A \text{ has no rows}) \text{ return } (x); \text{ else return } (b); \]
\[ c = \text{choose_column}(A); \]
\[ x_c = 1; A^1 = \text{select_column}(A, c); x^1 = \text{bcp}(A^1, x, b); \]
\[ \text{if } (\text{cost}(x^1) < \text{cost}(b)) \text{ then } \]
\[ b = x^1; \]
\[ \text{if } (\text{cost}(b) = L) \text{ return } (b); \]
\[ x_c = 0; A^0 = \text{remove_column}(A, c); x^0 = \text{bcp}(A^0, x, b); \]
\[ \text{if } (\text{cost}(x^0) < \text{cost}(b)) \text{ then } b = x^0; \]
\[ \text{return } (b); \]
Branching

- If $A$ is not a terminal case, matrix is cyclic.
- To find minimal solution, must determine column to branch on.
- A column intersecting short rows is preferred for branching.
- Assign a weight to each row that is inverse of row length.
- If a column intersects short rows is preferred for branching.
- To find minimal solution, must determine column to branch on.
- Sum the weights of all the rows covered by a column.
- Column $x_c$ with highest value is chosen for case splitting.

Branching Example

$\begin{bmatrix}
    x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
$

BCP Algorithm

```python
bcp(A, x, b)
{A, x} = reduce(A, x); 
L = lower_bound(A, x); 
if (L ≥ cost(b)) then return (b); 
if (terminalCase(A)) then 
  c = choose_column(A); 
x_c = 1; A^T = select_column(A, c); x^T = bcp(A^T, x, b) 
if (cost(x^T) < cost(b)) then 
  b = x^T; 
if (cost(b) = L) return (b); 
x_c = 0; A^T = remove_column(A, c); x^T = bcp(A^T, x, b) 
if (cost(x^T) < cost(b)) then b = x^T; 
return (b); 
```
### Branching Example

- \( x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 0, x_6 = 0, x_8 = 0, x_9 = 0 \)
- \( \text{cost}(x_1) = 3 \)
- \( \text{Recall that } L = 3 \)
- Therefore, we are done.

### Branching Example

- \( x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 0, x_6 = 0, x_7 = 0, x_9 = 0 \)
- \( \text{cost}(x_1') = 3 \)
- Recall that \( L = 3 \)
- Therefore, we are done.

\[
A = \begin{bmatrix}
1 & 1 & - & - & - & - & - & - & - \\
1 & - & - & 1 & - & - & - & - & - \\
- & - & 1 & 1 & - & - & - & - & - \\
- & - & 1 & - & 1 & - & - & - & - \\
- & - & 1 & - & 1 & - & - & - & - \\
- & - & 1 & - & - & 1 & - & - & - \\
- & - & 1 & - & - & 1 & - & - & - \\
1 & - & - & - & - & - & 1 & - & - \\
1 & - & - & - & - & - & - & 1 & - \\
1 & - & - & - & - & - & - & - & 1
\end{bmatrix}
\]
Branching Example

\[
A = \begin{bmatrix}
1 & 1 & - & - & 1 \\
1 & - & 1 & - & 2 \\
- & 1 & 1 & 1 & 6 \\
- & - & - & 1 & 7
\end{bmatrix}
\]

\[x_4 = 0, x_5 = 1, x_6 = 1, x_7 = 1, x_8 = 1\]

State Minimization Overview

- Original flow table may contain redundant rows, or states.
- Reducing number of states, reduces number of state variables.
- State minimization procedure:
  - Identify all compatible pairs of states.
  - Finds all maximal compatibles.
  - Find set of prime compatibles.
  - Setup a covering problem where prime compatibles are the solutions, and states are what needs to be covered.
- For SIC fundamental mode, same as for synchronous FSMs.

Example Huffman Flow Table

<table>
<thead>
<tr>
<th>(X_1)</th>
<th>(X_2)</th>
<th>(X_3)</th>
<th>(X_4)</th>
<th>(X_5)</th>
<th>(X_6)</th>
<th>(X_7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a,0</td>
<td>–</td>
<td>d,0</td>
<td>e,1</td>
<td>b,0</td>
<td>a,–</td>
</tr>
<tr>
<td>b</td>
<td>b,0</td>
<td>d,1</td>
<td>a,–</td>
<td>a,–</td>
<td>a,1</td>
<td>–</td>
</tr>
<tr>
<td>c</td>
<td>b,0</td>
<td>d,1</td>
<td>a,1</td>
<td>–</td>
<td>–</td>
<td>g,0</td>
</tr>
<tr>
<td>d</td>
<td>e,–</td>
<td>e,–</td>
<td>b,–</td>
<td>b,0</td>
<td>–</td>
<td>a,–</td>
</tr>
<tr>
<td>e</td>
<td>b,–</td>
<td>e,–</td>
<td>a,–</td>
<td>b,–</td>
<td>e,–</td>
<td>a,1</td>
</tr>
<tr>
<td>f</td>
<td>b,0</td>
<td>c,–</td>
<td>–,1</td>
<td>h,1</td>
<td>1,1</td>
<td>g,0</td>
</tr>
<tr>
<td>g</td>
<td>–</td>
<td>c,1</td>
<td>–</td>
<td>e,1</td>
<td>–</td>
<td>g,0</td>
</tr>
<tr>
<td>h</td>
<td>a,1</td>
<td>e,0</td>
<td>d,1</td>
<td>b,0</td>
<td>b,–</td>
<td>e,–</td>
</tr>
</tbody>
</table>

Pair Chart

Unconditionally Compatible

- Two states \(u\) and \(v\) are output compatible when for each input in which both are specified, they produce the same output.
- Two states \(u\) and \(v\) are unconditionally compatible when output compatible and go to the same next states.
- When two states \(u\) and \(v\) are unconditionally compatible, the \((u,v)\) entry is marked with the symbol \(\sim\).
When two states $u$ and $v$ are not output compatible, the states are incompatible.

When two states $u$ and $v$ are incompatible, the $(u, v)$ entry is marked with the symbol $\times$.

Two states are conditionally compatible when there exists differences in their next state entries.

If differing next states are merged, they become compatible.

When two states $u$ and $v$ are compatible only when states $s$ and $t$ are merged then the $(u, v)$ entry is marked with $s, t$. 
The final step is to check each pair of conditional compatibles.

If any pair of next states are known to be incompatible, then the states are also incompatible.

In this case, the \((u,v)\) entry is marked with the symbol \(\times\).

Next need to find larger sets of compatible states.

If \(S\) is compatible, then any subset of \(S\) is also compatible.

A maximal compatible is a compatible that is not a subset of any larger compatible.

From maximal compatibles, can determine all other compatibles.
Initial Boolean formula for incompatibles:
\[
\Gamma_1 = \{ (a + b), (a + c), (a + d), (a + e), (a + f), (a + g), (a + h), (b + c), (b + d), (b + e), (b + f), (b + g), (b + h), (c + d), (c + e), (c + f), (c + g), (c + h), (d + e), (d + f), (d + g), (d + h), (e + f), (e + g), (e + h), (f + g), (f + h), (g + h) \}
\]

Each term defines a maximal compatible set where states that do not occur make up the maximal compatible.

\[
cfg, deh, bcd, abde, ag
\]
### Example for Prime Compatibles

Maximal compatibles = \{ abde, bcd, cfg, deh, ag \}

### Setting up the Covering Problem

- A collection of prime compatibles forms a valid solution when it is a closed cover.
- A collection of compatibles is a cover when all states are contained in some compatible in the set.
- A collection is closed when all implied states are contained in some other compatible.
- \( c_i = 1 \) when the \( i \)-th prime compatible is in the solution.
- Using \( c_i \) variables, can write a Boolean formula that represents the conditions for a solution to be a closed cover.
- The formula is a product-of-sums where each product is a covering constraint.

### Covering Constraints

- There is one covering constraint for each state.
- The product is simply a disjunction of the prime compatibles that include the state.
- In other words, for the covering constraint to yield 1, one of the primes that includes the state must be in the solution. For example, the covering constraint for state \( a \) is:

\[
(c_1 + c_{11})
\]

### Closure Constraints

- There is a closure constraint for each implied compatible for each prime compatible.

For example, the prime bcd requires the following states to be merged: \((a,b), (a,g), (d,e)\).

Therefore, if we include bcd in the cover (i.e., \( c_2 \)), then we must also select compatibles which will merge these other state pairs.

- abde is the only prime compatible that merges \( a \) and \( b \).
- Therefore, we have a closure constraint of the form:

\[
c_2 \Rightarrow c_1
\]
Prime Compatibles

<table>
<thead>
<tr>
<th>Prime compatibles</th>
<th>Class set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  abde</td>
<td>{}</td>
</tr>
<tr>
<td>2  bcd</td>
<td>{(a,b),(a,g),(d,e)}</td>
</tr>
<tr>
<td>3  cfg</td>
<td>{(c,d),(e,h)}</td>
</tr>
<tr>
<td>4  deh</td>
<td>{(a,b),(a,d)}</td>
</tr>
<tr>
<td>5  bc</td>
<td>{}</td>
</tr>
<tr>
<td>6  cd</td>
<td>{(a,g),(d,e)}</td>
</tr>
<tr>
<td>7  cf</td>
<td>{(c,d)}</td>
</tr>
<tr>
<td>8  cg</td>
<td>{(c,d),(f,g)}</td>
</tr>
<tr>
<td>9  fg</td>
<td>{(e,h)}</td>
</tr>
<tr>
<td>10 dh</td>
<td>{}</td>
</tr>
<tr>
<td>11 ag</td>
<td>{}</td>
</tr>
<tr>
<td>12 f</td>
<td>{}</td>
</tr>
</tbody>
</table>

Solving the Covering Problem

Rows 4, 11, and 17 dominate row 5, Row 14 dominates row 15.

\[
\begin{align*}
\text{Cyclic MIS} & = \{1, 6, 8\}, \\
& \Rightarrow L = 3, c_1 \text{ has a branching weight of 1.33 which is best.}
\end{align*}
\]

Product of Sums Formulation

\[
\begin{align*}
(c_1 + c_2)(c_1 + c_2 + c_3)(c_1 + c_2 + c_3 + c_4 + c_6 + c_7) \\
& (c_1 + c_2 + c_3 + c_4)(c_1 + c_2)(c_1 + c_2 + c_3 + c_5) \\
& (c_1 + c_2 + c_3 + c_4 + c_5)(c_1 + c_2 + c_3 + c_4) \\
& (c_1 + c_2 + c_3 + c_4 + c_5 + c_6)(c_1 + c_2 + c_3 + c_4 + c_6) \\
& (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7)(c_1 + c_2 + c_3 + c_4 + c_6 + c_7) \\
& (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8)(c_1 + c_2 + c_3 + c_4 + c_6 + c_7 + c_8) = 1
\end{align*}
\]
Solving the Covering Problem

Cyclic MIS = \{ 12 \}, \ L = 3 \ c_2 \ has \ best \ branching \ value \ of \ 0.75.

\[
A = \begin{bmatrix}
\end{bmatrix}
\]

\[c_1 = 1, c_4 = 1, c_{10} = 0, c_{12} = 0\]
Solving the Covering Problem

Let's try \( c_2 = 0 \).

\[
A = \begin{bmatrix}
0 & -1 \\
1 & 1 & - \\
- & 0 & 1 & - \\
1 & 1 & 1 & 1 & - \\
-1 & -1 & -1 & -1 & - \\
-1 & -1 & -1 & 1 & 1 \\
0 & - & - & - & - & - \\
1 & 0 & -1 & - & - & - \\
- & - & - & 0 & - & - \\
- & - & - & 0 & - & - \\
- & - & - & 0 & 1 & - \\
\end{bmatrix}
\]

\( c_1 = 1, c_3 = 1, c_4 = 1, c_5 = 0, c_7 = 0, c_8 = 0, c_9 = 0, c_{10} = 0, c_{12} = 0 \)

Now must select both \( c_9 \) and \( c_{11} \), so another solution of cost 5.

\[
A = \begin{bmatrix}
1 & - \\
0 & 1 \\
\end{bmatrix}
\]

\( c_1 = 1, c_2 = 0, c_3 = 1, c_4 = 1, c_5 = 0, c_7 = 0, c_8 = 0, c_9 = 0, c_{10} = 0, c_{12} = 0 \)

Cyclic Branch on \( c_9 \).

\[
A = \begin{bmatrix}
1 & 1 & 1 & 1 & - & - & - & - \\
- & - & - & - & 1 & - & - & - \\
- & - & - & - & - & 1 & 1 & 1 \\
0 & - & - & - & - & - & - & 1 \\
1 & 0 & -1 & - & - & - & - & - \\
- & - & 0 & - & - & - & - & - \\
1 & - & - & 0 & - & - & - & - \\
- & - & - & - & - & 0 & - & - \\
- & - & - & - & - & - & 1 & - \\
\end{bmatrix}
\]

\( c_1 = 1, c_3 = 0, c_4 = 1, c_{10} = 0, c_{12} = 0 \)

Column \( c_9 \) dominates \( c_7 \) and \( c_8 \).

\[
A = \begin{bmatrix}
1 & 1 & 1 & 1 & - & - & - & - \\
- & - & - & - & 1 & - & - & - \\
- & - & - & - & - & 1 & 1 & 1 \\
0 & - & - & - & - & - & - & 1 \\
1 & 0 & -1 & - & - & - & - & - \\
- & - & 0 & - & - & - & - & - \\
1 & - & - & 0 & - & - & - & - \\
- & - & - & - & - & 0 & - & - \\
- & - & - & - & - & - & 1 & - \\
\end{bmatrix}
\]

\( c_1 = 1, c_3 = 0, c_4 = 1, c_{10} = 0, c_{12} = 0 \)

Column \( c_9 \) dominates \( c_2 \) and \( c_{11} \).

\[
A = \begin{bmatrix}
1 & 1 & - \\
0 & - & 1 \\
1 & 0 & - \\
1 & -1 & 0 & - \\
\end{bmatrix}
\]

\( c_1 = 1, c_3 = 0, c_4 = 1, c_7 = 0, c_8 = 0, c_9 = 1, c_{10} = 0, c_{12} = 0 \)
### Solving the Covering Problem

$c_2$ is essential. Found solution \{c_1, c_2, c_3, c_4\} with cost 4.

Not as good as lower bound of 3.

Continue with $c_3 = 0$ to obtain solution \{c_1, c_2, c_3, c_7, c_{11}\}.

\[
A = \begin{bmatrix}
    c_5 & c_{11} \\
    1 & - \\
\end{bmatrix}
\]

$c_1 = 1, c_3 = 0, c_4 = 1, c_7 = 0, c_9 = 1, c_{10} = 0, c_{12} = 0$

---

### Let’s try $c_5 = 0$.

\[
A = \begin{bmatrix}
    c_2 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_0 & c_{11} & c_{12} \\
    1 & - & - & - & - & 1 & - & - & - & 1 \\
    1 & 1 & - & - & 1 & 1 & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & - & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & - & 1 & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & - & 1 & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & - & 1 \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & - \\
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\]

$c_1 = 1, c_2 = 0, c_3 = 0, c_4 = 1, c_5 = 0, c_6 = 1, c_7 = 0, c_8 = 0, c_9 = 1, c_{10} = 0, c_{11} = 0, c_{12} = 0$

---

### Solving the Covering Problem

$c_{11}$ is essential and $c_2$ and $c_4$ are unacceptable.

\[
A = \begin{bmatrix}
    c_2 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
    1 & 1 & 1 & 1 & 1 & - & - & - & - & 1 \\
\end{bmatrix}
\]

---

### Solving the Covering Problem

All rows dominate row 5.

\[
A = \begin{bmatrix}
    c_2 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} \\
\end{bmatrix}
\]

$c_2 = 0, c_4 = 0, c_{11} = 1$

---

### Solving the Covering Problem

All columns mutually dominate.

\[
A = \begin{bmatrix}
    c_2 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} \\
\end{bmatrix}
\]

$c_1 = c_2 = c_4 = 0, c_{11} = 1$

---

### Solving the Covering Problem

No solution, so bcp returns best solution of \{c_1, c_4, c_5, c_9\}.

\[
A = \begin{bmatrix}
    c_9 \\
    - & - \\
\end{bmatrix}
\]

$c_1 = c_2 = c_4 = c_5 = c_9 = 0, c_7 = c_8 = c_{10} = 0, c_{11} = 1, c_{12} = 0$
Final Solution

<table>
<thead>
<tr>
<th>Prime compatibles</th>
<th>Class set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 abde</td>
<td>0</td>
</tr>
<tr>
<td>4 deh</td>
<td>(a, b), (a, d)</td>
</tr>
<tr>
<td>5 bc</td>
<td>0</td>
</tr>
<tr>
<td>9 fg</td>
<td>(e, h)</td>
</tr>
</tbody>
</table>

Example Huffman Flow Table

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
<th>( x_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a, 0</td>
<td>d, 0</td>
<td>e, 1</td>
<td>b, 0</td>
<td>a, -</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>b, -</td>
<td>d, 1</td>
<td>a, -</td>
<td>a, -</td>
<td>a, 1</td>
<td>-</td>
</tr>
<tr>
<td>c</td>
<td>b, 0</td>
<td>d, 1</td>
<td>a, -</td>
<td>a, -</td>
<td>-</td>
<td>g, 0</td>
</tr>
<tr>
<td>d</td>
<td>-</td>
<td>e, -</td>
<td>b, -</td>
<td>b, -</td>
<td>a, -</td>
<td>-</td>
</tr>
<tr>
<td>e</td>
<td>b, -</td>
<td>e, -</td>
<td>b, -</td>
<td>e, -</td>
<td>a, 1</td>
<td>-</td>
</tr>
<tr>
<td>f</td>
<td>b, 0</td>
<td>c, -</td>
<td>h, 1</td>
<td>f, 1</td>
<td>g, 0</td>
<td>-</td>
</tr>
<tr>
<td>g</td>
<td>-</td>
<td>c, 1</td>
<td>e, 1</td>
<td>-</td>
<td>g, 0</td>
<td>f, 0</td>
</tr>
<tr>
<td>h</td>
<td>a, 1</td>
<td>e, 0</td>
<td>d, 1</td>
<td>b, 0</td>
<td>b, -</td>
<td>e, -</td>
</tr>
</tbody>
</table>

Reduced Flow Table

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
<th>( x_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 0</td>
<td>[1, 4], 1</td>
<td>1, 1</td>
<td>1, 0</td>
<td>1, 1</td>
<td>1, 1</td>
</tr>
<tr>
<td>4</td>
<td>1, 1</td>
<td>[1, 4], 0</td>
<td>1, 1</td>
<td>[1, 5], 0</td>
<td>1, 1</td>
<td>1, 1</td>
</tr>
<tr>
<td>5</td>
<td>[1, 5], 0</td>
<td>[1, 4], 1</td>
<td>1, 1</td>
<td>-</td>
<td>1, 1</td>
<td>1, 0</td>
</tr>
<tr>
<td>9</td>
<td>[1, 5], 0</td>
<td>5, 1</td>
<td>-</td>
<td>4, 1</td>
<td>1, 1</td>
<td>9, 0</td>
</tr>
</tbody>
</table>

Final Reduced Flow Table

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
<th>( x_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 0</td>
<td>1, 1</td>
<td>1, 1</td>
<td>1, 1</td>
<td>1, 1</td>
<td>1, 1</td>
</tr>
<tr>
<td>4</td>
<td>1, 1</td>
<td>1, 0</td>
<td>1, 1</td>
<td>1, 0</td>
<td>1, 1</td>
<td>1, 1</td>
</tr>
<tr>
<td>5</td>
<td>1, 1</td>
<td>1, 1</td>
<td>-</td>
<td>1, 1</td>
<td>1, 1</td>
<td>5, 0</td>
</tr>
<tr>
<td>9</td>
<td>1, 0</td>
<td>5, 1</td>
<td>-</td>
<td>4, 1</td>
<td>1, 1</td>
<td>9, 0</td>
</tr>
</tbody>
</table>

State Assignment

- Each row must be encoded using a unique binary code.
- In synchronous design, a correct encoding can be assigned arbitrarily using \( n \) bits for a flow table with \( 2^n \) rows or less.
- In asynchronous design, more care must be taken to ensure that a circuit can be built that is independent of signal delays.

Critical Races

- When present state equals next state, circuit is stable.
- When codes differ in one bit, the circuit is in transition.
- When the codes differ in multiple bits, the circuit is racing.
- A race is critical when differences in delay can cause it to reach different stable states.
- A state assignment is correct when it is free of critical races.
### Minimum Transition Time State Assignment

A transition from state $s_i$ to state $s_j$ is **direct** (denoted $[s_i; s_j]$) when all state variables are excited to change at the same time.

$[s_i; s_j]$ **races critically** with $[s_k; s_l]$ when unequal delays can cause these transitions to pass through a common state.

When all state transitions are direct, the state assignment is called a **minimum transition time state assignment**.

A **flow table** in which each unstable state leads directly to a stable state is called a **normal flow table**.

---

### A Simple Huffman Flow Table

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>a</td>
<td>00</td>
<td>000</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>01</td>
<td>011</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>10</td>
<td>110</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>d</td>
<td>d</td>
<td>b</td>
<td>11</td>
<td>101</td>
</tr>
</tbody>
</table>

---

### Partition Theory

A partition $\pi$ on a set $S$ is a set of subsets of $S$ such that their pairwise intersection is empty.

The disjoint subsets of $\pi$ are called **blocks**.

A partition is **completely specified** if union of subsets is $S$.

Otherwise, the partition is **incompletely specified**.

Elements of $S$ which do not appear in $\pi$ are **unspecified**.

---

### Partition Theory and State Assignment

$n$ state variables $y_1, \ldots, y_n$ induce $\tau$-partitions $\tau_1, \ldots, \tau_n$.

States with $y_1 = 0$ are in one block of $\tau_1$ while those with $y_1 = 1$ are in the other block.

Each partition is composed of only one or two blocks.

Order blocks appear or which is assigned a 0 or 1 is arbitrary.

Once we find one valid assignment, others can be found by complementing or reordering variables.

---

### Partition Example

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_1y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00</td>
<td>000</td>
</tr>
<tr>
<td>b</td>
<td>01</td>
<td>011</td>
</tr>
<tr>
<td>c</td>
<td>10</td>
<td>110</td>
</tr>
<tr>
<td>d</td>
<td>11</td>
<td>101</td>
</tr>
</tbody>
</table>

---

### Partition Example

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_1y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00</td>
<td>000</td>
</tr>
<tr>
<td>b</td>
<td>01</td>
<td>011</td>
</tr>
<tr>
<td>c</td>
<td>10</td>
<td>110</td>
</tr>
<tr>
<td>d</td>
<td>11</td>
<td>101</td>
</tr>
</tbody>
</table>

$\tau_1 = \{ab; cd\}$

$\tau_2 = \{ac; bd\}$

$\tau_1 = \{ab; cd\}$

$\tau_2 = \{ad; bc\}$

$\tau_3 = \{ac; bd\}$
Partition List

- $\pi_2 \leq \pi_1$ if all elements specified in $\pi_2$ are specified in $\pi_1$, and each block of $\pi_2$ appears in a unique block of $\pi_1$.
- A partition list is a collection of partitions of the form:
  - $\{s_i, s_j, s_k\}$ where $[s_i, s_j]$ and $[s_i, s_k]$ are transitions in the same column.
  - $\{s_p, s_q, s_r\}$ where $[s_p, s_q]$ and is a transition in the same column as the stable state $s_i$.
- A state assignment for a normal flow table is a minimum transition time assignment free of critical races if each partition in the partition list is $\leq$ some $\tau_i$.

Tracey's Theorem

**Theorem 5.2 (Tracey, 1966)** A row assignment allotting one $y$-state per row can be used for direct transition realization of normal flow tables without critical races if, and only if, for every transition $[s_i, s_j]$:
- If $[s_m, s_n]$ is another transition in the same column, then at least one $y$-variable partitions the pair $\{s_i, s_j\}$ and the pair $\{s_m, s_n\}$ into separate blocks.
- If $s_k$ is a stable state in the same column then at least one $y$-variable partitions the pair $\{s_i, s_j\}$ and the state $s_k$ into separate blocks.
- For $i \neq j$, $s_i$ and $s_j$ are in separate blocks of at least one $y$-variable partition.

Partition List Example

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>d</td>
<td>b</td>
</tr>
</tbody>
</table>

Larger Example

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a,0</td>
<td>c,1</td>
<td>d,0</td>
</tr>
<tr>
<td>b</td>
<td>a,0</td>
<td>f,1</td>
<td>c,1</td>
</tr>
<tr>
<td>c</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>d</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>e</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>f</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a,0</td>
<td>c,1</td>
<td>d,0</td>
</tr>
<tr>
<td>b</td>
<td>a,0</td>
<td>f,1</td>
<td>c,1</td>
</tr>
<tr>
<td>c</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>d</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>e</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>f</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a,0</td>
<td>c,1</td>
<td>d,0</td>
</tr>
<tr>
<td>b</td>
<td>a,0</td>
<td>f,1</td>
<td>c,1</td>
</tr>
<tr>
<td>c</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>d</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>e</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>f</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a,0</td>
<td>c,1</td>
<td>d,0</td>
</tr>
<tr>
<td>b</td>
<td>a,0</td>
<td>f,1</td>
<td>c,1</td>
</tr>
<tr>
<td>c</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>d</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>e</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>f</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a,0</td>
<td>c,1</td>
<td>d,0</td>
</tr>
<tr>
<td>b</td>
<td>a,0</td>
<td>f,1</td>
<td>c,1</td>
</tr>
<tr>
<td>c</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>d</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>e</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>f</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a,0</td>
<td>c,1</td>
<td>d,0</td>
</tr>
<tr>
<td>b</td>
<td>a,0</td>
<td>f,1</td>
<td>c,1</td>
</tr>
<tr>
<td>c</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>d</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>e</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
<tr>
<td>f</td>
<td>f,1</td>
<td>f,1</td>
<td>e,1</td>
</tr>
</tbody>
</table>
## Boolean Matrix Example

\[
\begin{align*}
\pi_1 &= \{ ab; cf \} & a & b & c & d & e & f \\
\pi_2 &= \{ ac; de \} & 0 & 0 & 1 & - & - & 1 \\
\pi_3 &= \{ ac; de \} & 0 & - & 1 & - & 0 & 1 \\
\pi_4 &= \{ ac; bf \} & 0 & 0 & - & 1 & 1 & - \\
\pi_5 &= \{ bd; de \} & 0 & 1 & 0 & - & - & 1 \\
\pi_6 &= \{ ad; bc \} & 0 & - & 0 & - & 1 & 0 \\
\pi_7 &= \{ ad; ce \} & 0 & 1 & 0 & 1 & 0 & - \\
\pi_8 &= \{ ac; bd \} & 0 & 0 & 1 & 0 & 1 & - \\
\pi_9 &= \{ ac; ef \} & 0 & 0 & - & 0 & - & 1 \\
\pi_{10} &= \{ bd; ef \} & 0 & 1 & 0 & - & 0 & 1 \\
\end{align*}
\]

## Boolean Matrix and State Assignment

- State assignment problem is to find a Boolean matrix \( C \) with a minimum number of rows such that each row in the original partition list matrix is covered by some row of \( C \).
- The rows of this reduced matrix represent the \( c \)-partitions.
- The columns of this matrix represent a state assignment.
- Number of rows is the same as the number of state variables.

## Intersection

- Two rows of a Boolean matrix, \( R_i \) and \( R_j \), have an intersection if \( R_i \) and \( R_j \) agree wherever both \( R_i \) and \( R_j \) are specified.
- The intersection is formed by creating a row which has specified values taken from either \( R_i \) or \( R_j \).
- Entries where neither \( R_i \) or \( R_j \) are specified are left unspecified.
- A row, \( R_i \), includes another row, \( R_j \), when \( R_i \) agrees with \( R_j \) wherever \( R_j \) is specified.
- A row, \( R_i \), covers another row, \( R_j \), if \( R_j \) includes \( R_i \) or \( R_j \) includes the complement of \( R_i \).
- The complement of \( R_i \) is denoted \( \overline{R_i} \).

## Boolean Matrix Reduction

\[
\begin{align*}
\begin{array}{cccccc}
\pi_1 & \pi_2 & \pi_3 & \pi_4 & \pi_5 & \pi_6 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\end{align*}
\]

## Minimal Boolean Matrix

\[
\begin{align*}
\begin{array}{cccccc}
\gamma_i \gamma_j & a & b & c & d & e \\
(\pi_1, \pi_2, \pi_3) & 0 & 0 & 1 & 0 & 1 \\
(\pi_2, \pi_3, \pi_6) & 0 & 1 & 1 & 0 & 0 \\
(\pi_2, \pi_3, \pi_6) & 0 & 1 & 0 & 1 & 1 \\
\end{array}
\end{align*}
\]

## Intersectables

- If a set of rows, \( \pi_i, \pi_j, \ldots, \pi_k \), have an intersection, they are called an intersectable.
- An intersectable may be enlarged by adding a row \( \pi_l \) if \( \pi_l \) has an intersection with every element in the set.
- An intersectable which cannot be enlarged further is called a maximal intersectable.
Theorem 5.3 (Unger, 1969) Let $D$ be a set of ordered partitions derived from some set of unordered partitions. For some state $s$, label as $\rho_1, \rho_2, \ldots$ etc. the members of $D$ having $s$ in their left sets, and label as $q_1, q_2, \ldots$ etc. the members of $D$ that do not contain $s$ in either set. Then a minimal set of maximal intersectables covering each member of $D$ or its complement can be found by considering only the ordered partitions labeled as $\rho$'s or $q$'s. (The complements of the $p$'s can be ignored.)
### Setting up the Covering Problem

Cyclic with $L = 3$

Branch on $x_4$

\[
A = \begin{bmatrix}
- & - & - & - & - & - & - & - & 1 & 1 & 1 \\
\end{bmatrix}
\]

$x_4 = 1, x_5 = 0, x_6 = 0, x_7 = 0, x_9 = 0$

### Reduced Covering Problem

$x_2$ and $x_3$ are now essential

\[
A = \begin{bmatrix}
1 & - & 1 & - & - \\
1 & - & - & 1 & - \\
- & - & 1 & - & 1 \\
- & - & - & 1 & 1 \\
1 & - & - & - & 1 \\
- & - & - & - & - \\
1 & - & - & - & - \\
- & - & - & - & - \\
\end{bmatrix}
\]

### Minimal Boolean Matrix

\[
x_2 : (\pi_1, \pi_7, \pi_{10}) \quad 0 \quad 0 \quad 1 \\
x_3 : (\pi_2, \pi_5, \pi_6) \quad 0 \quad 1 \quad 1 \\
x_4 : (\pi_3, \pi_4, \pi_9) \quad 0 \quad 1 \quad 1 \\
x_5 = 1, x_6 = 0, x_7 = 0, x_9 = 0
\]

### Original Flow Table

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a,0</td>
<td>c,1</td>
<td>d,0</td>
</tr>
<tr>
<td>b</td>
<td>a,0</td>
<td>f,1</td>
<td>c,1</td>
</tr>
<tr>
<td>c</td>
<td>f,1</td>
<td>c,1</td>
<td>c,1</td>
</tr>
<tr>
<td>d</td>
<td>--</td>
<td>a,0</td>
<td>d,0</td>
</tr>
<tr>
<td>e</td>
<td>a,0</td>
<td>d,0</td>
<td>c,1</td>
</tr>
<tr>
<td>f</td>
<td>f,1</td>
<td>f,1</td>
<td>--</td>
</tr>
</tbody>
</table>

### Encoded Flow Table

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000.0</td>
<td>011.1</td>
<td>100.0</td>
</tr>
<tr>
<td>110</td>
<td>000.0</td>
<td>111.1</td>
<td>011.1</td>
</tr>
<tr>
<td>011</td>
<td>111.1</td>
<td>011.1</td>
<td>011.1</td>
</tr>
<tr>
<td>100</td>
<td>--</td>
<td>100.0</td>
<td>011.0</td>
</tr>
<tr>
<td>101</td>
<td>000.0</td>
<td>100.0</td>
<td>011.1</td>
</tr>
<tr>
<td>111</td>
<td>111.1</td>
<td>111.1</td>
<td>--</td>
</tr>
</tbody>
</table>
Fed-Back Outputs as State Variables

- Previously ignored outputs during state assignment.
- May be possible to feed back outputs as state variables.
- Determine in each state under each input the value of each output upon entry.
- This information can satisfy some partitions.
- Satisfying partitions, can reduce number of state variables.

Modified Partition List

<table>
<thead>
<tr>
<th>π₂</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>π₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>π₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>π₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>π₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>π₈</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>π₉</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example Flow Table

<table>
<thead>
<tr>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>x₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>c₁</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>f₁</td>
<td>c₁</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>c₁</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>d₀</td>
<td>b₀</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>c₁</td>
<td>e₁</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>f₁</td>
<td>e₁</td>
</tr>
</tbody>
</table>

Pairwise Intersectibles

- Pairwise Intersectibles
- Maximal Intersectibles
- Maximal Intersectibles (cont)
### Constraint Matrix

\[
A = \begin{bmatrix}
1 & - & - & - & \pi_2 \\
-1 & 1 & - & - & \pi_3 \\
1 & - & 1 & 1 & \pi_4 \\
1 & - & - & - & \pi_5 \\
-1 & 1 & - & - & \pi_6 \\
-1 & - & - & - & \pi_7 \\
-1 & 1 & 1 & 1 & \pi_8 \\
-1 & - & - & - & \pi_9 \\
\end{bmatrix}
\]

### Minimal Boolean Matrix

\[
x_1 x_2 x_3 x_4
\begin{array}{c}
\pi_2 \\
\pi_3 \\
\pi_4 \\
\pi_5 \\
\pi_6 \\
\pi_7 \\
\pi_8 \\
\pi_9 \\
\end{array}
\]

### Original Flow Table

\[
x_1 x_2 x_3 x_4
\begin{array}{c|c|c|c|c|c}
a & b & c & d & e & f \\
0 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

### New Encoded Flow Table

\[
x_1 x_2 x_3 x_4
\begin{array}{c|c|c|c|c|c}
00 & 0 & 1 & 0 & 1 & 0 \\
11 & 0 & 1 & 1 & 1 & 1 \\
01 & 1 & 1 & 0 & 1 & 1 \\
10 & 0 & 0 & 1 & 0 & 1 \\
11 & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]

### Hazard-free Logic Synthesis

- For each next state and output signal:
  - Derive sum-of-products (SOP) implementation.
  - Transform SOP using laws of Boolean algebra into a multi-level logic implementation.
  - Map to gates found in the given gate library.
- For asynchronous FSMs, must avoid hazards in SOP.
- Some laws of Boolean algebra introduce hazards.
- First describe for SIC fundamental-mode.

### Boolean Functions and Minterms

- A Boolean function \( f \) of \( n \) variables \( x_1, x_2, \ldots, x_n \) is a mapping:
  \[ f : \{0, 1\}^n \to \{0, 1, -\}. \]
- Each element \( m \) of \( \{0, 1\}^n \) is called a minterm.
- The value of a variable \( x_i \) in a minterm \( m \) is given by \( m(i) \).
- The ON-set of \( f \) is the set of minterms which return 1.
- The OFF-set of \( f \) is the set of minterms which return 0.
- The DC-set of \( f \) is the set of minterms which return -.
Asynchronous Circuit Design

The consensus of any two terms of the formula either does not exist or is in some term of the formula.

The consensus theorem states: \( xy + xz = xy + xz + yz \).

The product \( yz \) is called the consensus for \( xy \) and \( xz \).

A complete sum is defined to be a SOP formula composed of all the prime implicants.

Theorem 5.5 (Blake, 1937) A SOP is a complete sum iff:

1. No term includes any other term.
2. The consensus of any two terms of the formula either does not exist or is contained in some term of the formula.

Theorem 5.6 (Blake, 1937) If we have two complete sums \( f_1 \) and \( f_2 \), we can obtain the complete sum for \( f_1 \) - \( f_2 \) using the following two steps:

1. Multiply out \( f_1 \) and \( f_2 \) using the following properties
   - \( x \cdot x = x \) (idempotent)
   - \( x \cdot (y + z) = xy + xz \) (distributive)
   - \( x \cdot 0 = 0 \) (complement)

2. Eliminate all terms contained in some other term.

A recursive procedure for finding the complete sum for \( f \):

\[
\text{cs}(f) = \text{abs}(x_1 + \text{cs}(f(0, x_2, \ldots, x_n))) \\
-\text{abs}(x_1 + \text{cs}(f(1, x_2, \ldots, x_n)))
\]

where \( \text{abs}(f) \) removes absorbed terms from \( f \) (abs\((a + ab) = a\)).

For functions of less than 4 variables, can use a Karnaugh map.

For more variables, Karnaugh maps too tedious.

Quine’s tabular method is better but requires all minterms be explicitly listed.

Briefly describe a recursive procedure based on consensus and complete sums.
Combinational Hazards

- For asynchronous design, two-level logic minimization problem is complicated by hazards.
- Let us consider the design of a function $f$ to implement either an output or next variable.
- When input changes under SIC, circuit moves from minterm $m_1$ to another $m_2$ which differ in value in exactly one $x_i$.
- During this transition, there are four possible transitions of $f$:
  - Static 0 $\rightarrow$ 0 transition: $f(m_1) = f(m_2) = 0$.
  - Static 1 $\rightarrow$ 1 transition: $f(m_1) = f(m_2) = 1$.
  - Dynamic 0 $\rightarrow$ 1 transition: $f(m_1) = 0$ and $f(m_2) = 1$.
  - Dynamic 1 $\rightarrow$ 0 transition: $f(m_1) = 1$ and $f(m_2) = 0$.

Static 0-Hazard

- If during a static 0 $\rightarrow$ 0 transition, the cover of $f$ can due to differences in delays momentarily evaluate to 1, then we say that there exists a static 0-hazard.
- In a SOP cover of a function, no product term is allowed to include either $m_1$ or $m_2$ since they are in the OFF-set.
- Static 0-hazard exists only if some product includes both $x_i$ and $\overline{x_i}$.
- Such a product is not useful since it contains no minterms.
- If we exclude such product terms from the cover, then the SOP cover can never produce a static 0-hazard.

Static 1-Hazard

- If during a static 1 $\rightarrow$ 1 transition, the cover of $f$ can evaluate to 0, then we say that there exists a static 1-hazard.
- Consider case where one product $p_1$ contains $m_1$ but not $m_2$ and another product $p_2$ contains $m_2$ but not $m_1$.
- If $p_1$ is implemented with a faster gate than $p_2$, then the gate for $p_1$ can turn off faster than the gate for $p_2$ turns on which can lead to the cover momentarily evaluating to 0.
- To eliminate static 1-hazards, for each $m_1 \rightarrow m_2$, there must exist a product in the cover that includes both $m_1$ and $m_2$. 

Prime Implicant Selection

<table>
<thead>
<tr>
<th>$xz$</th>
<th>$yz$</th>
<th>$x\overline{z}$</th>
<th>$w\overline{z}$</th>
<th>$wx$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Recursive Prime Generation: Example

$$f(w,x,y,z) = y'z' + xz + wx'y'z + wxyz'$$
$$f(w,x,y,0) = y' + w'xy$$
$$f(w,x,0,0) = 1$$
$$f(w,x,1,0) = w'x$$
$$CS(f(w,x,y,0)) = \text{abs}(y+1)(y'+w'x) = y'+w'x$$
$$f(w,x,y,1) = x + wx'y'$$
$$f(w,0,y,1) = wy'$$
$$f(w,1,y,1) = 1$$
$$CS(f(w,x,y,1)) = \text{abs}(x + wy')(x' + 1) = x + wy'$$
$$CS(f(w,x,y,z)) = \text{abs}(z + y' + w'x)(z' + x + wy')$$
$$= \text{abs}(xz + wy'z + y'z' + xy' + wy' + w'xz' + w'x)$$
$$= xz + y'z' + xy' + wy' + w'x$$
### Static 1-Hazard Example

#### Dynamic Hazards

- If during a $0 \rightarrow 1$ transition, the cover can change from 0 to 1 back to 0 and finally stabilize at 1, we say the cover has a dynamic $0 \rightarrow 1$ hazard.
- Assuming no useless product terms (ones that include both $x_i$ and $\overline{x_i}$), this is impossible under the SIC assumption.
- No product is allowed to include $m_1$ since it is in the OFF-set.
- Any product that includes $m_2$ turns on monotonically.
- Similarly, there are no dynamic $1 \rightarrow 0$ hazards.

### Removing Hazards

- A simple, inefficient approach to produce a hazard-free SOP cover is to include all prime implicants in the cover.
- Since two minterms $m_1$ and $m_2$ in a transition are distance 1 apart, they must be included together in some prime.
- An implicant exists which is made up of all literals that are equal in both $m_1$ and $m_2$.
- This implicant must be part of some prime implicant.
- For our example, the following cover is guaranteed to be hazard-free under SIC:

$$ f = xz + yz' + x'y + wy + w'x $$

### Better Approach to Remove Hazards

- Form an implicant out of each pair of states $m_1$ and $m_2$ involved in a static $1 \rightarrow 1$ transition which includes each literal that is the same value in both $m_1$ and $m_2$.
- The covering problem is now to find the minimum number of prime implicants that cover each of these transition cubes.

### 2-Level Hazard-Free Synthesis: Example

$$
\begin{array}{cccc}
xz & yz & x'y & wy \\
\overline{x}y & 1 & 1 & 1 \\
x'y & 1 & 1 & 1 \\
x'z & 1 & 1 & 1 \\
xz & 1 & 1 & 1 \\
wz & 1 & 1 & 1 \\
wz & 1 & 1 & 1 \\
wy & 1 & 1 & 1 \\
\end{array}
$$

### Extensions for MIC Operation

- Preceding restricted the class of circuits to SIC.
- Each input burst can have only a single transition.
- Now extend the synthesis method to MIC.
- Synthesize any XBM machine satisfying the maximal set property.
Function Hazards

- If a transition has a function hazard, there is no implementation of the function which avoids the hazard during the transition.
- Fortunately, the synthesis method never produces a design with a transition that has a function hazard.

Combinational Hazards for State Variables

- A minimum transition time state assignment has MIC hazards.
- Multiple changing next state variables may be fed back to the input of the FSM.
- The circuit moves from one minterm \( m_1 \) to another minterm \( m_2 \), but multiple state variables may be changing concurrently.
- For normal flow tables with outputs that change only in unstable states then only static transitions possible.
Asynchronous Circuit Design

MIC Static Hazards

- There can be no static 0-hazards.
- Since multiple variables may be changing concurrently, the cover may pass through other minterms between \( m_1 \) and \( m_2 \).
- To be free of static 1-hazards, it is necessary that a single product in the cover include all these minterms.
- Each \([m_1, m_2]\) where \( f(m_1) = f(m_2) = 1 \), must be contained in some product in the cover to eliminate static 1-hazards.

MIC Dynamic Hazards

- For each \( 1 \rightarrow 0 \) transition, \([m_1, m_2]\), if a product in the cover intersects \([m_1, m_2]\), then it must include the start point, \( m_1 \).
- For each \( 0 \rightarrow 1 \) transition, \([m_1, m_2]\), if a product in the cover intersects \([m_1, m_2]\), then it must include the end point, \( m_2 \).

Burst-Mode Transitions

- In legal BM machines, types of transitions are restricted.
- A function may only change value after every transition in the input burst has occurred.
- \([m_1, m_2]\) for a function \( f \) is a burst-mode transition if for every minterm \( m \in [m_1, m_2] \), \( f(m_1) = f(m) \).
- The result is that if a function \( f \) only has burst-mode transitions, then it is free of function hazards.
- Also, any dynamic \( 0 \rightarrow 1 \) transition is free of dynamic hazards.
- For any legal BM machine, there exists a hazard-free cover for each output and next state variable before state minimization.
### Burst-Mode Machine to Flow Table

<table>
<thead>
<tr>
<th>s0</th>
<th>s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,0</td>
<td>01,0</td>
</tr>
<tr>
<td>11,1</td>
<td></td>
</tr>
<tr>
<td>10,0</td>
<td></td>
</tr>
<tr>
<td>xy</td>
<td></td>
</tr>
</tbody>
</table>

### Generalized Transition Cube

- Generalized transition cube allows start and end points to be cubes rather than simply minterms.
- In the generalized transition cube \([c_1, c_2]\), the cube \(c_1\) is called the start cube and \(c_2\) is called the end cube.
- The open generalized transition cube \([c_1, c_2)\) is all minterms in \([c_1, c_2]\) excluding those in \(c_2\) (i.e., \([c_1, c_2) = [c_1, c_2] - c_2\)).

### Extended Burst-Mode Transitions

- In XBM machine, some signals are rising, some are falling, and others are levels which can change nonmonotonically.
- Rising and falling signals change monotonically.
- Level signals must hold the same value in \(c_1\) and \(c_2\), where the value is either a constant (0 or 1) or a don’t care (−).
- Level signals may change nonmonotonically.
- Transitions are restricted such that each function may change value only after the completion of an input burst.

\([c_1, c_2]\) for a function \(f\) is an extended burst-mode transition if for every minterm \(m_i \in [c_1, c_2]\), \(f(m_i) = f(c_1)\) and for every minterm \(m_i \in c_2\), \(f(m_i) = f(c_2)\).

If a function has only extended burst-mode transitions, then it is function hazard-free.

### Start and End Subcubes

- **Start subcube**, \(c_1'\), is maximal subcube of \(c_1\) where signals having directed don’t-care transitions are set to initial value.
- **End subcube**, \(c_2'\), is maximal subcube of \(c_2\) where signals having directed don’t-care transitions are set to final value.

### Start and End Subcube: Example

- Assume that \(z\) is a rising directed don’t care transition.
### Hazard Issues

- Considering \([c_1, c_2]\), hazard considerations are same.
- If a static 1 → 1 transition the entire transition cube must be included in some product term in the cover.
- If a dynamic 1 → 0 transition, any product that intersects this transition cube must contain the start subcube, \(c_1\).
- Must also consider dynamic 0 → 1 transitions.
- Any product that intersects transition cube for a dynamic 0 → 1 transition must contain the end subcube, \(c_2\).

### XBM Hazard Issues: Example

<table>
<thead>
<tr>
<th>(xy)</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(yz)</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Extended Burst-Mode to Flow Table

<table>
<thead>
<tr>
<th>(ab)</th>
<th>s0</th>
<th>s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0,0</td>
<td>0,1</td>
</tr>
<tr>
<td>01</td>
<td>0,0</td>
<td>1,1</td>
</tr>
<tr>
<td>11</td>
<td>1,1</td>
<td>1,1</td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td>0,1</td>
</tr>
</tbody>
</table>

### State Minimization: Burst-Mode

- After state minimization, it is possible that no hazard-free cover exists for some variable in the design.

<table>
<thead>
<tr>
<th>Inputs a b c</th>
<th>000</th>
<th>001</th>
<th>011</th>
<th>010</th>
<th>110</th>
<th>111</th>
<th>110</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>A,0</td>
<td>A,0</td>
<td>B,1</td>
<td>D,0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
<td>B,1</td>
<td>B,1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>C,0</td>
<td>C,0</td>
<td>B,1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### State Minimization: Extended Burst-Mode

### DHF-Compatibles

- Two states \(s_1\) and \(s_2\) are dhf-compatible when they are compatible and for each output \(z\) and transition \([c_1, c_2]\) of \(s_1\) and for each transition \([c_3, c_4]\) of \(s_2\):
  - If \(z\) has a 1 → 0 transition in \([c_1, c_2]\) and a 1 → 1 transition in \([c_3, c_4]\), then \([c_1, c_2] \cap [c_3, c_4] = \emptyset\).
  - If \(z\) has a 1 → 0 transition in \([c_1, c_2]\) and a 1 → 0 transition in \([c_3, c_4]\), then \([c_1, c_2] \cap [c_3, c_4] = \emptyset\).
  - If \(z\) has a 0 → 1 transition in \([c_1, c_2]\) and a 1 → 1 transition in \([c_3, c_4]\), then \([c_1, c_2] \cap [c_3, c_4] = \emptyset\).

- Two states \(s_1\) and \(s_2\) are dhf-compatible when they are compatible and for each output \(z\) and transition \([c_1, c_2]\) of \(s_1\) and for each transition \([c_3, c_4]\) of \(s_2\):
  - If \(z\) has a 1 → 0 transition in \([c_1, c_2]\) and a 1 → 1 transition in \([c_3, c_4]\), then \([c_1, c_2] \cap [c_3, c_4] = \emptyset\).
  - If \(z\) has a 1 → 0 transition in \([c_1, c_2]\) and a 1 → 0 transition in \([c_3, c_4]\), then \([c_1, c_2] \cap [c_3, c_4] = \emptyset\).
  - If \(z\) has a 0 → 1 transition in \([c_1, c_2]\) and a 1 → 1 transition in \([c_3, c_4]\), then \([c_1, c_2] \cap [c_3, c_4] = \emptyset\).

### State Minimization

- State minimization: Extended Burst-Mode

<table>
<thead>
<tr>
<th>(&lt;b+&gt;a+/c+)</th>
<th>B</th>
<th>(a+/c+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs a b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- DHF-Compatibles
Further Restrictions

- $s_1$ and $s_2$ must also satisfy the following further restriction for each $s_3$ which can transition to $s_1$ in $[c_3, c_4]$ and another transition $[c_1, c_2]$ of $s_2$:
  - If $z$ has a $1 \rightarrow 0$ transition in $[c_1, c_2]$ and a $1 \rightarrow 1$ transition in $[c_3, c_4]$, then $[c_3, c_4] \cap [c_1, c_2] = \emptyset$ or $c_2 \in [c_3, c_4]$.
  - If $z$ has a $0 \rightarrow 0$ transition in $[c_1, c_2]$ and a $1 \rightarrow 1$ transition in $[c_3, c_4]$, then $[c_3, c_4] \cap [c_1, c_2] = \emptyset$ or $c_2 \in [c_3, c_4]$.
  - For each $s_3$ which can transition to $s_2$ in $[c_3, c_4]$ and another transition $[c_1, c_2]$ of $s_2$:
    - If $z$ has a $1 \rightarrow 0$ transition in $[c_1, c_2]$ and a $1 \rightarrow 1$ transition in $[c_3, c_4]$, then $[c_3, c_4] \cap [c_1, c_2] = \emptyset$ or $c_2 \in [c_3, c_4]$.
    - If $z$ has a $0 \rightarrow 0$ transition in $[c_1, c_2]$ and a $1 \rightarrow 1$ transition in $[c_3, c_4]$, then $[c_3, c_4] \cap [c_1, c_2] = \emptyset$ or $c_2 \in [c_3, c_4]$.

Extended Burst-Mode Dynamic Hazard Problem

State Assignment

Hazard-free dff Circuit

Required Cubes

- Transition cubes for each $1 \rightarrow 1$ transition are required cubes.
- The end point of the transition cube for a $0 \rightarrow 1$ transition is a required cube.
- Transition subcubes for each $1 \rightarrow 0$ transition are required cubes.
- The transition subcubes for $1 \rightarrow 0$ transition $[m_1, m_2]$ are all cubes of the form $[m_1, m_2]$ such that $f(m_2) = 1$.
- Can eliminate any subcube contained in another.
- The union of the required cubes forms the ON-set.
- Each of the required cubes must be contained in some product of the cover to ensure hazard-freedom.

Required Cubes: BM Example

- Transition cubes for each $1 \rightarrow 1$ transition are required cubes.
- The end point of the transition cube for a $0 \rightarrow 1$ transition is a required cube.
- Transition subcubes for each $1 \rightarrow 0$ transition are required cubes.
- The transition subcubes for $1 \rightarrow 0$ transition $[m_1, m_2]$ are all cubes of the form $[m_1, m_2]$ such that $f(m_2) = 1$.
- Can eliminate any subcube contained in another.
- The union of the required cubes forms the ON-set.
- Each of the required cubes must be contained in some product of the cover to ensure hazard-freedom.
Priveleged Cubes

- The transition cubes for each dynamic \(1 \rightarrow 0\) or \(0 \rightarrow 1\) transition are called priveleged cubes.
- They cannot be intersected unless the intersecting product also includes its start subcube (\(1 \rightarrow 0\)) or end subcube (\(0 \rightarrow 1\)).
- If a cover includes a product that intersects a priveleged cube without including its start subcube (or end subcube), then the cover is not hazard-free.

DHF-Prime Implicants

- We may not be able to produce a SOP cover that is free of dynamic hazards using only prime implicants.
- A dhf-implicant is an implicant which does not illegally intersect any priveleged cube.
- A dhf-prime implicant is a dhf-implicant that is contained in no other dhf-implicant.
- A dhf-prime implicant may not be a prime implicant.
- A minimal hazard-free cover includes only dhf-prime implicants.

Prime Implicants: Example

\[
\begin{align*}
f(a, b, c, d) &= a \bar{c} + \bar{a} b c + b c d + \bar{a} c \\
f(a, b, 0, d) &= a + a b + a b d \\
f(0, b, 0, d) &= \bar{a} + b \\
f(1, b, 0, d) &= 1 \\
cs(f(a, b, c, 0)) &= \text{abs}(a + a + b)(\bar{a} + 1)) = a + \bar{b} \\
f(a, b, 1, d) &= b d + \bar{a} \\
f(0, b, 1, d) &= 1 \\
f(1, b, 1, d) &= b d \\
cs(f(a, b, 1, d)) &= \text{abs}(a + 1)(\bar{a} + b d)) = \bar{a} + b d \\
cs(f(a, b, c, d)) &= \text{abs}(c + a + b)(c + \bar{a} + b d)) \\
&= \text{abs}(b d + a b + a b d + a b d + a b d + a b d + a b d) \\
&= \bar{a} c + a c + \bar{a} c + b c + a b + b d
\end{align*}
\]

DHF-Prime Implicants: Example

Required Cubes: XBM Example

<table>
<thead>
<tr>
<th>( \bar{a} \bar{c} \bar{b} \bar{d} )</th>
<th>( a \bar{c} \bar{b} )</th>
<th>( a b \bar{c} )</th>
<th>( \bar{b} d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{a} \bar{c} \bar{b} \bar{d} )</td>
<td>( a \bar{c} \bar{b} )</td>
<td>( a b \bar{c} )</td>
<td>( \bar{b} d )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
t_0 &= \{a b c d, a b d\} \\
b &= \{a b \bar{c} d, a b c d\} \\
b &= \{a \bar{b} d, a b d\}
\end{align*}
\]
Multi-Level Logic Synthesis

- Two-level SOP implementations cannot be realized directly for most technologies.
- AND or OR stages of arbitrarily large fan-in not practical.
- In CMOS, gates with more than 3 or 4 inputs are too slow.
- Two-level SOP implementations must be decomposed using Boolean algebra laws into multi-level implementations.
- Care must be taken not to introduce hazards.
- We present a number of hazard-preserving transformations.
- If we begin with a hazard-free SOP implementation and only apply hazard-preserving transformations than the resulting multi-level implementation is also hazard-free.

Hazard-Preserving Transformations

- **Theorem 5.19** (Unger, 1969) Given any expression \( f_1 \), if we transform it into another expression, \( f_2 \), using the following laws:
  - \( A + (B + C) \Leftrightarrow A + B + C \) (associative law)
  - \( A(BC) \Leftrightarrow ABC \) (associative law)
  - \( AB \Leftrightarrow A + B \) (DeMorgan’s theorem)
  - \( AB \Leftrightarrow A(B + C) \) (distributive law)
  - \( A + AB \Leftrightarrow A \) (absorptive law)
  - \( A + \overline{A}B \Leftrightarrow A \) (DeMorgan’s theorem)

then a circuit corresponding to \( f_2 \) will have no combinational hazards not present in circuits corresponding to \( f_1 \).

More Hazard-Preserving Transformations

- Hazard exchanges:
  - Insertion or deletion of inverters at the output of a circuit only interchanges 0 and 1-hazards.
  - Insertion or deletion of inverters at the inputs only relocates hazards to duals of original transition.
  - The dual of a circuit (exchange AND and OR gates) produces dual function with dual hazards.

Multilevel Logic Synthesis: Example

\[
 f = a c + a c + c d + b c d + b c \\
 f = c (a + b + d) + c (a + b d)
\]

Technology Mapping

- Technology mapping step takes as input a set of technology-independent logic equations and a library of cells, and it produces a netlist of cells.
- Broken up into three major steps:
  - Decomposition,
  - Partitioning, and
  - Matching/covering.

Decomposition

- Decomposition transforms logic equations into equivalent network using only two-input/one-output base functions.
- A typical choice of base function is two-input NAND gates.
- Decomposition performed using recursive applications of DeMorgan’s theorem and the associative law.
- These operations are hazard-preserving.
- Simplification during this step may remove redundant logic added to eliminate hazards, so must be avoided.
**Decomposition Example**

\[ f = c \left( a + b + d \right) \]
\[ f = c \left( (a + b) + d \right) \]  (associative law)
\[ f = c \left( (a + b) d \right) + c \left( a (b d) \right) \]  (DeMorgan's theorem)
\[ f = (c (a b d)) \]  (DeMorgan's theorem)

**Partitioning**

- Partitioning breaks up decomposed network at points of multiple fanout into single output cones of logic.
- Since partitioning step does not change the topology of the network, it does not affect the hazard behavior of the network.

**Gate Library**

- \( \text{Inv} \) (Cost = 1)
- \( 2\text{NAND} \) (Cost = 3)
- \( 3\text{NAND} \) (Cost = 5)
- \( 2\text{NOR} \) (Cost = 2)
- \( \text{AOI1} \) (Cost = 3)
- \( \text{AOI2} \) (Cost = 4)

**Matching and Covering**

- Matching and covering examines each cone of logic and finds cells in the library to implement subnetworks within the cone.
- Can be implemented either using structural pattern-matching or Boolean matching techniques.
- In the structural techniques, each library element is also decomposed into base functions.
- Library elements are then compared against portions of the network to be mapped using pattern matching.
- Assuming that the decomposed logic and library gates are hazard-free, the resulting mapped logic is also hazard-free.
Boolean Matching

Generalized C-Elements

Generalized C-Element Hazard Issues

Hazard Requirements

Generalized C-Element Example

Generalized C-Element Example
Sequential Hazards

- Huffman circuits require that outputs and state variables stabilize before either new inputs or feedback state variables arrive at the input to the logic.
- A violation of this assumption can result in a sequential hazard.
- Presence of a sequential hazard is dependent on timing of the environment, circuit, and feedback delays.

Essential Hazard

- Given table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>3.1</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>5.1</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Feedback Delay Requirement

- To eliminate essential hazards, there is a feedback delay requirement:
  \[ D_f \geq d_{\text{max}} - d_{\text{min}} \]
  where \( D_f \) is the feedback delay, \( d_{\text{max}} \) is the maximum delay in the combinational logic, and \( d_{\text{min}} \) is the minimum delay through the combinational logic.

Fundamental-Mode Constraint

- Sequential hazards can also result if the environment reacts too quickly.
- Fundamental-mode environmental constraint says inputs are not allowed to change until the circuit stabilizes.
- To satisfy this constraint, a conservative separation time needed between inputs can be expressed as follows:
  \[ d_i \geq 2d_{\text{max}} + D_f \]
  where \( d_i \) is the separation time needed between input bursts.
- Separation needs a \( 2d_{\text{max}} \) term since the circuit must respond to the input change followed by the subsequent state change.

Setup and Hold Time Constraint

- XBM machines require a setup time and hold time for conditional signals.
- Conditional signals must stabilize a setup time before the compulsory signal transition which samples them.
- It must remain stable a hold time after the output and state changes complete.
- Outside this window of time, the conditional signals are free to change arbitrarily.

Summary

- Binate covering problems
- State minimization
- State assignment
- Hazard-free logic synthesis
- Extensions for MIC operation
- Multilevel logic synthesis
- Technology mapping
- Generalized C-element implementation
- Sequential hazards