Asynchronous Circuit Design

Chris J. Myers
Lecture 6: Muller Circuits
Chapter 6

Muller Circuits
- Uses the unbounded gate delay model.
- Circuits are guaranteed to work regardless of gate delays assuming that wire delays are negligible.
- Requires knowledge of the allowed behaviors of the environment.
- There are no restrictions on the speed of the environment.

Muller Circuit Design
- Translate higher level specification into a state graph.
- If not complete state coded, change the protocol or add new internal state signal(s).
- Derive logic using modified logic minimization procedure.
- Map design to gates in a given gate library.

Overview
- Formal definition of speed independence.
- State assignment of Muller circuits.
- Logic minimization of Muller circuits.
- Technology mapping of Muller circuits.
Complete Circuits

- To design a speed independent circuit, must have complete information about both the circuit and its environment.
- We restrict our attention to complete circuits.
- A complete circuit $C$ is defined by a finite set of states, $S$.
- At any time, $C$ is said to be in one of these states.

Properties of Allowed Sequences

- For a sequence $(s_1, s_2, \ldots)$, consecutive states must be different (i.e., $s_i \neq s_{i+1}$).
- Each state $s \in S$ is the initial state of some allowed sequence.
- If $(s_1, s_2, s_3, \ldots)$ is allowed sequence then so is $(s_2, s_3, \ldots)$.
- If $(s_1, s_2, \ldots)$ and $(t_1, t_2, \ldots)$ are allowed sequences and $s_2 = t_1$, then $(s_1, t_1, t_2, \ldots)$ is also an allowed sequence.

Allowed Sequences

- Behavior of a complete circuit is defined by set of allowed sequences of states.
- Each allowed sequence can be either finite or infinite, and the set of allowed sequences can also be finite or infinite.
- The sequence $(s_1, s_2, s_3, \ldots)$ says that state $s_1$ is followed by state $s_2$, but it does not state at what time.

Simple Example Complete Circuit

- Consider a complete circuit composed of four states, $S = \{a, b, c, d\}$, which has the following two allowed sequences:
  1. $a, b, a, b, \ldots$
  2. $a, c, d$
- The sequences above imply the following allowed sequences:
  1. $b, a, b, a, \ldots$
  2. $c, d$
  3. $d$
  4. $a, b, a, c, d$
  5. $a, b, a, b, a, c, d$
  6. $b, a, c, d$
  7. etc.
The Followed and Equivalence Relations
- A state $s_i$ is followed by a state $s_j$ (denoted $s_iF s_j$) if there exists an $R$-sequence $(s_i, \ldots, s_j)$.
- The $F$-relation is reflexive and transitive, but not necessarily symmetric.
- If two states $s_i$ and $s_j$ are symmetric under the $F$-relation (i.e., $s_iF s_j$ and $s_jF s_i$), they are said to be equivalent (denoted $s_iE s_j$).

Equivalence Classes
- The equivalence relation $E$ partitions the finite set of states $S$ of any circuit into equivalence classes of states.
- The $F$-relation can be extended to these equivalence classes.
- If $A$ and $B$ are two equivalence classes, then $A FB$ if there exists states $a \in A$ and $b \in B$ such that $aF b$.
- If $a \in A$ and $b \in B$ and $A FB$, then $aF b$.

\( \mathcal{R} \)-related and \( \mathcal{R} \)-sequences
- Two states $s_i, s_j \in S$ are $\mathcal{R}$-related, (denoted $s_iR s_j$) when:
  1. $s_i = s_j$ or
  2. $s_i, s_j$ appear consecutively in some allowed sequence.
- A sequence $(s_1, s_2, \ldots, s_m)$ is an $\mathcal{R}$-sequence if $s_iR s_{i+1}$ for each $1 \leq i \leq m-1$. 
**Speed Independence**

- For any allowed sequence, there is a definite last class which is called the *terminal class*.
- A circuit $C$ is *speed independent with respect to a state $s$* if all allowed sequences starting with $s$ have the same terminal class.

---

**Allowed Sequences on State Graphs**

- An allowed sequence of states $(s_1, s_2, \ldots)$ is any sequence of states satisfying the following three conditions:
  1. No two consecutive states $s_i$ and $s_{i+1}$ are equal.
  2. For any state $s_{j+1}$ and signal $u_i$ one of the following is true:
     \[
     s_{j+1}(i) = s_j(i) \\
     s_{j+1}(i) = s'_j(i)
     \]
  3. If there exists a signal $u_i$ and a state $s_j$ such that
     \[
     s_j(i) = s_r(i) \text{ and } s'_j(i) = s'_r(i)
     \]
     for all $s_r$ in the sequence following $s_j$, then
     \[
     s_j(i) = s'_j(i)
     \]

---

**Equivalence Classes for Simple Example**

- A
- B
- C

**Simple Speed-Independent Circuit**

- A state graph with labeled states and transitions is shown.
Totally Sequential

- A circuit is **totally sequential** with respect to a state $s$ if there is only one allowed sequence starting with $s$.

Semi-Modularity

- A circuit is **semi-modular** if in each state in which multiple signals are excited, that in the states reached after one signal has transitioned, that the remaining signals are still excited.

$$\forall t_1, t_2 \in T \cdot (s_i, t_1, s_j) \in \delta \wedge (s_k, t_2, s_l) \in \delta \Rightarrow \exists s_t \in S \cdot (s_j, t_2, s_t) \in \delta \wedge (s_k, t_1, s_t) \in \delta$$

- A totally sequential circuit is semi-modular but the converse is not necessarily true.

- A semi-modular circuit is also speed independent, but again the converse is not necessarily true.

Output Semi-Modularity

- Input transitions are typically allowed to be disabled by other input transitions, so another useful class of circuits are those which are **output semi-modular**.

- A SG is output semi-modular if only input signal transition can disable other input signal transitions.

$$\forall t_1 \in T_O \cdot \forall t_2 \in T \cdot (s_i, t_1, s_j) \in \delta \wedge (s_k, t_2, s_l) \in \delta \Rightarrow \exists s_t \in S \cdot (s_j, t_2, s_t) \in \delta \wedge (s_k, t_1, s_t) \in \delta$$

where $T_O$ is the set of output transitions (i.e., $T_O = \{u+, u- \mid u \in O\}$).
Excitation States

- It is often useful to be able to determine in which states a signal is excited to rise or fall.
- The sets of excitation states, $ES(u^+)$ and $ES(u^-)$, are defined as follows:
  \[
  ES(u^+) = \{ s \in S \mid s(u) = 0 \land u \in X(s) \}
  \]
  \[
  ES(u^-) = \{ s \in S \mid s(u) = 1 \land u \in X(s) \}
  \]
- Recall that $X(s)$ is the set of signals that are excited in state $s$.

Quiescent States

- For each signal $u$, there are two sets of quiescent states.
- The sets $QS(u^+)$ and $QS(u^-)$ are defined as follows:
  \[
  QS(u^+) = \{ s \in S \mid s(u) = 1 \land u \notin X(s) \}
  \]
  \[
  QS(u^-) = \{ s \in S \mid s(u) = 0 \land u \notin X(s) \}
  \]

Example

\[
ES(y^+) = \{ (RR0), (R00) \}
\]
\[
ES(y^-) = \{ (FF1), (0F1) \}
\]
\[
QS(y^+) = \{ (R10), (11R) \}
\]
\[
QS(y^-) = \{ (F01), (00F) \}
\]

Excitation Regions

- An excitation region for signal $u$ is a maximally connected subset of either $ES(u \uparrow)$ or $ES(u \downarrow)$.
- If it is a subset of $ES(u \uparrow)$, it is a set region (denoted $ER(u \uparrow, k)$).
- Similarly, a reset region is denoted $ER(u \downarrow, k)$. 
Switching Regions

- The switching region for a transition \( u^* \), \( SR(u^*, k) \), is the set of states directly reachable through transition \( u^* \):

\[
SR(u^*, k) = \{ s_j \in S \mid \exists s_i \in ER(u^*, k). (s_i, u^*, s_j) \in \delta \}
\]

Example

- \( ER(y^+, 1) = \{(RR0), (1R0)\} \)
- \( ER(y^-, 1) = \{(FF1), (0F1)\} \)
- \( SR(y^+, 1) = \{(R10), (11R)\} \)
- \( SR(y^-, 1) = \{(F01), (00F)\} \)

Distributive State Graphs

- A state graph is distributive if each excitation region has a unique minimal state.
- A minimal state for \( ER(u^*, k) \) is a state in \( ER(u^*, k) \) which cannot be directly reached by any other state in \( ER(u^*, k) \).
- More formally, a SG is distributive if:

\[
\forall ER(u^*, k) . \exists exactly one s_j \in ER(u^*, k) .
\]

\[
\neg \exists s_i \in ER(u^*, k) . (s_i, t, s_j) \in \delta
\]

Trigger Signals

- Each cube in the implementation is composed of trigger signals and context signals.
- For an excitation region, a trigger signal is a signal whose firing can cause the circuit to enter the excitation region.
- The set of trigger signals for an excitation region \( ER(u^*, k) \) is:

\[
TS(u^*, k) = \{ v \in N \mid \exists s_i, s_j \in S, (v, (s_i, t, s_j) \in \delta)
\]

\[
\land (t = v + \lor t = v-)
\]

\[
\land (s_i \notin ER(u^*, k)) \land (s_j \in ER(u^*, k))\}
\]

- Any non-trigger signal which is stable in the excitation region can potentially be a context signal.
Context Signals

- Any non-trigger signal which is stable in the excitation region can potentially be a context signal.
- The set of context signals for an excitation region $ER(u_*, k)$ is:

$$CS(u_*, k) = \{ v_i \in N | v_i \notin TS(u_*, k) \land \forall s_j, s_l \in ER(u_*, k). s_j(i) = s_l(i) \}$$

Example

- $TS(y^+, 1) = \{ z \}$
- $TS(y^-, 1) = \{ z \}$
- $CS(y^+, 1) = \{ y \}$
- $CS(y^-, 1) = \{ y \}$
The Passive/Active Wine Shop: State Sets

\[
\begin{align*}
\text{ES}(\text{req\_patron}+) & = \{ (R00R), (100R) \} \\
\text{ES}(\text{req\_patron}--) & = \{ (R10F), (110F) \} \\
\text{QS}(\text{req\_patron}+) & = \{ (RR01), (1R01) \} \\
\text{QS}(\text{req\_patron}--) & = \{ (RF00), (1F00), (R000), (10R0), \\
& \quad (F010), (00F0) \} \\
\text{ER}(\text{req\_patron}+, 1) & = \{ (R00R), (100R) \} \\
\text{ER}(\text{req\_patron}--, 1) & = \{ (R10F), (110F) \} \\
\text{SR}(\text{req\_patron}+, 1) & = \{ (RR01), (1R01) \} \\
\text{SR}(\text{req\_patron}--, 1) & = \{ (RF00), (1F00) \}
\end{align*}
\]

Unique State Codes (USC)

- Two states have unique state codes (USC) if they are labeled with different binary vectors.
  \[ \text{USC}(s_i, s_j) \Leftrightarrow \lambda_S(s_i) \neq \lambda_S(s_j) \]
- A SG has USC if all states pairs have USC.
  \[ \text{USC}(S) \Leftrightarrow \forall(s_i, s_j) \in S \times S. \text{USC}(s_i, s_j) \]

The Passive/Active Wine Shop: Signal Sets

\[
\begin{align*}
\text{TS}(\text{req\_patron}+, 1) & = \{ \text{ack\_wine} \} \\
\text{TS}(\text{req\_patron}--, 1) & = \{ \text{ack\_patron} \} \\
\text{CS}(\text{req\_patron}+, 1) & = \{ \text{ack\_patron}, \text{req\_patron} \} \\
\text{CS}(\text{req\_patron}--, 1) & = \{ \text{ack\_wine}, \text{req\_patron} \}
\end{align*}
\]

Complete State Codes (CSC)

- Two states have complete state codes (CSC) if they either have USC or if they do not have USC but do have the same output signals excited in each state.
  \[ \text{CSC}(s_i, s_j) \Leftrightarrow \text{USC}(s_i, s_j) \lor X(s_i) \cap O = X(s_j) \cap O \]
  \[ \text{CSC}(S) \Leftrightarrow \forall(s_i, s_j) \in S \times S. \text{CSC}(s_i, s_j) \]
- A set of state pairs which violate CSC is defined as:
  \[ \text{CSCV}(S) = \{ (s_i, s_j) \in S \times S \mid \neg \text{CSC}(s_i, s_j) \} \]
The Passive/Active Wine Shop: State Graph

Insertion Points
- Need to insert a rising and falling transition for new signal.
- A transition point is $TP = (t_s, t_e)$, where $t_s$ is a set of start transitions and $t_e$ is a set of end transitions.
- The transition point represents the location in the protocol in which a transition on a new state signal is to be inserted.
- In a Petri net, a TP represents a transition with incoming arcs from $t_s$ and with outgoing arcs to $t_e$.
- An insertion point is $IP = (TP_R, TP_F)$, where $TP_R$ is for the rising transition and $TP_F$ is for the falling transition.

The CSC Problem
- If a circuit does not have USC but has CSC, then the present state/next state relationship is not unique for input signals.
- Circuit only synthesized for outputs, so not a problem.
- When a circuit does not have CSC, the present state/next state relationship for output signals is ambiguous.
- Could resuffle the protocol as described earlier.
- Now introduce method for inserting state variables.

Transitioning States
- It is necessary to determine in which states a transition can occur when inserted into a $TP$.
- The transition on the new state signal becomes excited when the circuit enters $\cap_{t \in t_s} SR(t)$.
- Once this transition becomes excited it may remain excited in any subsequent states until there is a transition in $t_e$.
- The set of states in which a new transition is excited is defined recursively as follows:

$$S(TP) = \{s_j \in S \mid s_j \in \cap_{t \in t_s} SR(t) \vee (\exists(s_i, t, s_j) \in \delta . s_i \in S(TP) \wedge t \not\in t_e)\}$$
\[ TP = (\{req\_patron^{+}\}, \{req\_patron^{-}\}) \]

**Transition Point Restrictions**
- A transition point must satisfy the following three restrictions:
  1. Start and end sets are disjoint (i.e., \( t_s \cap t_e = \emptyset \)).
  2. End set does not include input transitions (i.e., \( \forall t \in t_e . t \notin T_I \)).
  3. Start and end sets include only concurrent transitions (i.e., \( \forall t_1, t_2 \in t_s . t_1 \parallel t_2 \) and \( \forall t_1, t_2 \in t_e . t_1 \parallel t_2 \)).

**Transition Point Examples**
- \( (\{req\_wine^{+}\}, \{ack\_wine^{-}\}) \)
- \( (\{req\_patron^{+}\}, \{ack\_patron^{+}\}) \)
- \( (\{req\_wine^{-}\}, \{ack\_wine^{+}\}) \)
- \( (\{req\_wine^{-}\}, \{ack\_wine^{-}\}) \)
- \( (\{req\_patron^{+}\}, \{req\_patron^{-}\}) \)
- \( (\{ack\_patron^{+}\}, \{req\_patron^{+}\}) \)
- \( (\{req\_wine^{+}, req\_patron^{-}\}, \{ack\_wine^{+}\}) \)
- \( (\{req\_wine^{+}, req\_patron^{-}\}, \{ack\_wine^{-}\}) \)
- \( (\{req\_wine^{+}, req\_patron^{-}\}, \{ack\_wine^{+}, req\_patron^{-}\}) \)
- \( (\{req\_wine^{+}, req\_patron^{-}\}, \{ack\_wine^{+}\}) \)
- \( (\{req\_wine^{+}, req\_patron^{-}\}, \{ack\_wine^{+}\}) \)
- \( (\{req\_wine^{+}, req\_patron^{-}\}, \{ack\_wine^{-}\}) \)

**Insertion Point Explosion**
- The set of all possible insertion points includes all combinations of transitions in \( t_s \) and \( t_e \) for \( TP_R \) and \( TP_F \).
- Upper bound on number of possible insertion points is \( 2^{|T|} \).
- Fortunately, many of these insertion points can be quickly eliminated because they either:
  - Never lead to a satisfactory solution of the CSC problem or
  - The same solution is found using a different insertion point.
Insertion Point Restrictions

- Each $IP = (TP_R, TP_F)$ must be checked for compatibility.
- Two TP’s are incompatible when either of the following is true:
  \[ TP_R(t_s) \cap TP_F(t_s) = \emptyset \]
  \[ TP_R(t_e) \cap TP_F(t_e) = \emptyset \]
- An incompatible insertion point always creates an inconsistent state assignment.
- Example:
  \[ IP = (\{\text{ack\_wine+}, \text{ack\_wine-}\}, \{\text{ack\_wine-}\}) \]

State Graph Coloring Procedure

- States in $S(TP_R)$ are colored as rising.
- States in $S(TP_F)$ are colored as falling.
- If a state is colored both rising and falling, this IP leads to an inconsistent state assignment and must be discarded.
- All states following those colored rising before reaching any colored falling are colored as high.
- Similarly, all states between those colored as falling and those colored as rising are colored as low.
- While coloring high or low, if a state to be colored is found to already have a color, IP leads to inconsistent state assignment.

State Graph Coloring

- Need to determine effect of inserting a state variable in an IP.
- Can be done by inserting the state signal and finding new SG.
- This approach is unnecessarily time consuming and may produce a SG with an inconsistent state assignment.
- Instead, SG is partitioned into four parts corresponding to the rising, falling, high, and low sets for the new state signal.
The IP with the smallest sum $\sum_{i \in I} (f_i(r) \cdot B_i) + \sum_{i \in I} (f_i(t) \cdot B_i)$.

Insertion Point Secondary Cost Functions

- Each new CSC violation must be added to the total remaining.
- If there is also a new CSC violation in the same state, then the total remaining increases.
- Similarly, if one is colored rising and the other is colored falling.
- For each pair of states with a USC violation but not a CSC violation, eliminate from CSC any part of violations in which one state is colored high while the other is colored low.
- Eliminating CSC violations after a state transition is inserted.
- The primary component of the cost function is the number of CSC violations.

The Colored Passive/Active Wine Shop SG
State Signal Insertion: Petri-net

- State signal can be inserted into a Petri-net by adding arcs from each transition in $t_e$ to the new state signal transition.
- Similarly, arcs are added from the new transition to each of the transitions $t_e$.
- The same steps are followed for the reverse transition.
- The state signal is assigned an initial value based on the coloring of the initial state.
- At this point, a new SG can be found.

The Passive/Active Wine Shop with CSC: SG

The Passive/Active Wine Shop with CSC: STG

State Signal Insertion: State Graph

- Alternatively, the new SG could be found directly.
- Each state in the original SG is extended to include new signal.
- If a state is colored low, then the new signal is ‘0’.
- If a state is colored high, then the new signal is ‘1’.
- If a state is colored rising then it must be split into two new states, one with new signal ‘R’ and the other has it as ‘1’.
- If a state is colored falling then it must be split into two new states, one has the new signal ‘F’ and the other has it as ‘0’.
CSC Solver Algorithm

```c
CSCSolverAlgorithm(SG) {
    CV = findCSCViolations(SG);
    best = CV;
    bestj = 0;
    TP = findTransitionPoints(SG);
    foreach TPj ∈ TP
        foreach TPk ∈ TP
            if IP = (TPj, TPk) is legal then {
                CSG = colorStateGraph(SG, TPj, TPk);
                CVj = findCSCViolations(CSG);
                if (CVj is consistent) and ((CVj < best) or
                    ((CVj = best) and (cost(IP) < cost(bestj)))) then {
                    best = CVj;
                    bestj = (TPj, TPk); }
            }
    SG = insertStateSignal(SG, bestj);
    SG = CSCSolverAlgorithm(SG); /* Add more signals, if needed. */
    return SG;
}
```

Atomic Gate Implementation

- Assume that each output to be synthesized is implemented using a single complex atomic gate.
- A gate is atomic when its delay is modeled by a single delay element connected to its output.

Atomic Gate Logic Synthesis

- On-set for a signal \( u \) is the set of states in which \( u \) is excited to rise or stable high.
- Off-set is set of states in which \( u \) is excited to fall or stable low.
- DC-set is the set of all unreachable states, or equivalently those states not included in either the on-set or off-set.

\[
ON-set = \{ \lambda_S(s) \mid s ∈ (ES(u+) \cup QS(u+)) \} \\
OFF-set = \{ \lambda_S(s) \mid s ∈ (ES(u-) \cup QS(u-)) \} \\
DC-set = \{ 0, 1 \}^{|N|} - (ON-set ∪ OFF-set)
\]

- Find primes using recursive procedure described earlier.
- Setup and solve a covering problem.
Atomic Gate: Example

\[
\begin{align*}
ON-set &= \{10000, 10100, 00100, 10101\} \\
OFF-set &= \{00101, 00001, 10001, 00011, 10011, 01011, 00010, \\
& \quad 10010, 01010, 11010, 01000, 11000, 11011, 00000\} \\
DC-set &= \{00110, 00111, 01001, 01100, 01101, 01110, 01111, \\
& \quad 10110, 10111, 11001, 11100, 11101, 11110, 11111\}
\end{align*}
\]

\[P = \{1-1--,-11--, --11-, --1-0, -1-01, 10-00\}\]

<table>
<thead>
<tr>
<th>(1 \leftrightarrow)</th>
<th>(-11\leftrightarrow)</th>
<th>(-11\leftrightarrow)</th>
<th>(-1\leftrightarrow0)</th>
<th>(-1-01)</th>
<th>(10-00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>10100</td>
<td>1</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>00100</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>10101</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
Two minimization problems must be solved for each signal $u$:
- For $\text{set}(u)$:
  - On-set is states in which $u$ is excited to rise.
  - Off-set is states in which $u$ is excited to fall or is stable low.
  - DC-set is stable high and unreachable states.
  - Stable high states are don’t cares, since once a gC is set its feedback holds its state.

$$\begin{align*}
\text{ON-set} &= \{ \lambda_S(s) \mid s \in (ES(u^+)) \} \\
\text{OFF-set} &= \{ \lambda_S(s) \mid s \in (ES(u^-) \cup QS(u^-)) \} \\
\text{DC-set} &= \{0, 1\}^N - (\text{ON-set} \cup \text{OFF-set})
\end{align*}$$

For $\text{reset}(u)$:
- On-set is states in which $u$ is excited to fall.
- Off-set is states in which $u$ is either rising or high.
- DC-set is stable low and unreachable states.

$$\begin{align*}
\text{ON-set} &= \{ \lambda_S(s) \mid s \in (ES(u^-)) \} \\
\text{OFF-set} &= \{ \lambda_S(s) \mid s \in (ES(u^+) \cup QS(u^+)) \} \\
\text{DC-set} &= \{0, 1\}^N - (\text{ON-set} \cup \text{OFF-set})
\end{align*}$$

Can now apply standard methods to find a minimum number of primes to implement the set and reset functions.
Passive/Active Shop: gC Circuit

Combinational Optimization

- If $set(u)$ is on in all states in which $u$ should be rising or high, then the state holding element can be removed.
- Implementation for $u$ is equal to the logic for $set(u)$.
- If $reset(u)$ is on in all states in which $u$ should be falling or low, then the signal $u$ can be implemented with $reset(u)$.
Gate-Level Hazard

State = \{abcde\}

Region Function Operation
- Each region function must:
  - Turn on only when it enters a state in its excitation region.
  - Turn off monotonically sometime after the signal \( u \) changes.
  - Must stay off until the excitation region is entered again.
- To guarantee this behavior, each region function must satisfy certain correctness constraints.
- Requires a modified logic minimization procedure.

Standard C-implementation
- Structure similar to gC-implementation, but built differently.
- Each AND gate, called a region function, implements a single (or possibly a set of) excitation region(s) for the signal \( u \).
- In gC-implementation, an excitation region can be implemented by multiple product terms.
- A region function may need to be implemented using SOP.

Region Function Covers
- Each region function is implemented using a single atomic gate, corresponding to a cover of an excitation region.
- A cover \( C(u*, k) \) is a set of states for which the corresponding region function evaluates to one.
- First present a method in which each region function only implements a single excitation region.
- Later extend the method to allow a single region function to implement multiple excitation regions to promote gate sharing.
Correctness Constraints: Intuition
- Each region function can only change when it is needed to actively drive the output signal to change.
- Consider a region function for a set region:
  - Gate turns on when circuit enters a state in the set region.
  - When region function changes to 1, it excites the OR gate.
  - When the OR gate changes to 1 in excites the C-element (assuming the reset network is low) to set $u$ to 1.
  - Only after $u$ rises can the region function be excited to fall.
  - The region function then must fall monotonically.
  - The signal $u$ will not be able to fall until the region function has fallen and the OR gate for the set network has fallen.
  - Once region function falls, it cannot be excited again until the circuit again enters a state in this set region.

Covering Constraint
- The reachable states in a correct cover must include the entire excitation region.
- It must not include any states outside the union of the excitation region and associated quiescent states.

Entrance Constraint
- A cover must only be entered through excitation region states.

Excitation Region Implicants
- Goal of logic minimization is to find an optimal SOP for each region function that satisfies the definition of a correct cover.
- An implicant of an excitation region is a product that may be part of a correct cover.
- $c$ is an implicant of an excitation region $ER(u^*, k)$ if the reachable states covered by $c$ are a subset of the states in the union of the excitation region and associated quiescent states.
Gate Level Logic Synthesis: Set Regions

- For each set region $ER(u+, k)$:
  - On-set is those states in $ER(u+, k)$.
  - Off-set includes states in which $u$ is falling or low, and also
    the states outside this excitation region where $u$ is rising.
  - This additional restriction is necessary to make sure that a
    region function can only turn on in its excitation region.

\[
\begin{align*}
ON-set & = \{ \lambda_S(s) \mid s \in (ER(u+, k)) \} \\
OFF-set & = \{ \lambda_S(s) \mid s \in (ES(u-) \cup QS(u-)) \cup \\
& \quad (ES(u+) - ER(u+, k)) \} \\
DC-set & = \{0, 1\}^{|N|} - (ON-set \cup OFF-set)
\end{align*}
\]

Gate Level Logic Synthesis: Reset Regions

- For a reset region $ER(u-, k)$:

\[
\begin{align*}
ON-set & = \{ \lambda_S(s) \mid s \in (ER(u-, k)) \} \\
OFF-set & = \{ \lambda_S(s) \mid s \in (ES(u-) \cup QS(u+)) \cup \\
& \quad (ES(u+) - ER(u-, k)) \} \\
DC-set & = \{0, 1\}^{|N|} - (ON-set \cup OFF-set)
\end{align*}
\]

Gate Level Circuit: Example

- There are two set regions for $c$: $ER(c+, 1) = 01R0$ and $ER(c+, 2) = 11R1$.
- Let’s examine the implementation of $ER(c+, 1)$.

\[
\begin{align*}
ON-set & = \{0100\} \\
OFF-set & = \{0000, 1000, 0010, 1100, 1101\} \\
DC-set & = \{0001, 0011, 0101, 0110, 0111, \\
& \quad 1001, 1010, 1011, 1110, 1111\}
\end{align*}
\]

- The primes found are as follows:

\[
P = \{01--,-1-1-, -11-, --01, -0-1, --11\}
\]
**Implied States**

- The entrance constraint creates a set of *implied states* for each implicant \(c\) (denoted \(IS(c)\)).
- A state \(s\) is in \(IS(c)\) if it is not covered by \(c\) but due to the entrance constraint must be covered if \(c\) is part of the cover.
- A state \(s_i\) is in \(IS(c)\) for \(ER(u*, k)\) if it is not covered by \(c\), and \(s_i\) leads to \(s_j\) which is both covered by \(c\) and not in \(ER(u*, k)\).

\[ IS(c) = \{ s_i \mid s_i \not\in c \land \exists s_j. ((s_i, t, s_j) \in \delta \land (s_j \in c \land s_j \not\in ER(u*, k))) \} \]

- This means that the product \(c\) becomes excited in a quiescent state instead of an excitation region state.
- If there no other product in the cover contains this implied state, the cover violates the entrance constraint.

---

**Existence of a Prime Cover**

- An implicant may have implied states that are outside the excitation region and the corresponding quiescent states.
- Implied states may not be covered by any other implicant.
- If this implicant is the only prime which covers some excitation region state, then no cover can be found using only primes.

---

**Candidate Implicants**

- Implicant is a *candidate implicant* if there does not exist one which properly contains it with a subset of the implied states.
- \(c_i\) is a candidate implicant if there *does not exist* an implicant \(c_j\) that satisfies the following two conditions:

\[ c_j \supset c_i \quad IS(c_j) \subseteq IS(c_i) \]

- Prime implicants are always candidate implicants, but not all candidate implicant are prime.
- An optimal cover exists using only candidate implicants.
- NOTE: similar to prime compatibles.
Candidate Implicant Algorithm

\[
\text{candidateImplicants}(SG, P) \{ \text{ done } = \emptyset \\
\text{ for}(k = |\text{largest}(P)|; k \geq 1; k--) \{ \\
\quad \text{ foreach}(q \in P; |q| = k) \text{ enqueue}(C, q) \\
\quad \text{ foreach}(c \in C; |c| = k) \{ \\
\quad \quad \text{ if}(IS(SG, c) = \emptyset) \text{ continue} \\
\quad \quad \text{ foreach}(s \in \text{litExtend}(c)) \{ \\
\quad \quad \quad \text{ if}(s \in \text{done}) \text{ continue} \\
\quad \quad \quad \Gamma_s = IS(SG, s) \\
\quad \quad \quad \text{ prime } = \text{true} \\
\quad \quad \quad \text{ foreach}(q \in C; |q| \geq k) \{ \\
\quad \quad \quad \quad \text{ if}(s \subseteq q) \{ \\
\quad \quad \quad \quad \quad \Gamma_s = IS(SG, q) \\
\quad \quad \quad \quad \quad \text{ if}(\Gamma_s \geq \Gamma_q) \{ \\
\quad \quad \quad \quad \quad \quad \text{ prime } = \text{false}; \\
\quad \quad \quad \quad \quad \quad \text{ break } \} \} \\
\quad \quad \quad \text{ if}(\text{prime } = 1) \text{ enqueue}(C, s) \\
\quad \quad \text{ done } = \text{done } \cup \{s\} \} \} \} \\
\} \\
\}
\]

Formulating the Covering Problem

- Introduce a Boolean variable \(x_i\) for each candidate implicant \(c_i\).
- The variable \(x_1 = 1\) when the candidate implicant is included in the cover and 0 otherwise.
- Using these variables, we can construct a product of sums representation of the covering and entrance constraints.

Covering Clauses

- A covering clause is constructed for each state \(s\) in \(ER(u*, k)\).
- Each clause consists of disjunction of candidates that cover \(s\).

\[
\bigvee_{1 \leq i \leq k} x_i.
\]

- \(ER(u*, k) = 0100\) which is included in only candidate implicants \(c_1 (01--)\) and \(c_2 (010-)\):

\[
(x_1 + x_2)
\]
Closure Clauses
- For each candidate implicant $c_i$, a closure clause is constructed for each of its implied states $s \in IS(c_i)$.
- Each closure clause represents an implication if a candidate implicant used, its implied states must be covered.
  $$\overline{\pi} \lor \bigvee_{j: s \in c_i} x_j,$$
- The candidate implicant $c_1$ (01-) has implied state 0110.
- 0110 included in implicants $c_3$ (1-1-) and $c_5$ (-11-).
  $$(\overline{x_1} + x_3 + x_5)$$
- Complete formula: $(x_1 + x_2)(\overline{x_1} + x_3 + x_5)\overline{x_3} x_3 x_8$

Setting Up the Constraint Matrix
- Find $x_i$'s that satisfy function with minimum cost.
- Since negated variables, the covering problem is binate.
- The constraint matrix has one row for each clause and one column for each candidate implicant.
- Rows divided into a covering section and a closure section.
- Covering section: row for each excitation region state $s$, with a 1 in every column with a candidate implicant that includes $s$.
- Closure section: row for each implied state $s$ of each candidate implicant $c_i$, with a 0 in the column corresponding to $c_i$ and a 1 in each column with a candidate implicant $c_j$ that covers $s$.

Constraint Matrix for $ER(c+,1)$

\[
\begin{array}{cccccccccccc}
01-- & 010- & 1-1- & 101- & -11- & 0-1- & -01- & --11 \\
1 & 1 & 1 & - & - & - & - & - \\
2 & 0 & - & 1 & - & 1 & - & - \\
3 & - & - & 0 & - & - & - & - \\
4 & - & - & - & 0 & - & - & - \\
5 & - & - & - & - & - & - & 0 \\
\end{array}
\]

A Simple Example

\[
\text{State } = \begin{array}{cccc}
a & b & c & d \\
010 & 0 & 1 & 0 \\
011 & 0 & 1 & 1 \\
101 & 1 & 0 & 0 \\
110 & 1 & 0 & 1 \\
\end{array}
\]
Combinational Optimization

- Can remove the C-element when the covers for the set function for a signal $u$ include all states where $u$ is rising or high.
  \[
  \bigcup_i C(u+, l) \supseteq ER(u+, l) \cup QS(u+)
  \]
- Or the covers for the reset function include all states where $u$ is falling or low.
  \[
  \bigcup_l C(u-, l) \supseteq ER(u-, l) \cup QS(u-)
  \]

Gate Sharing

- Single gate can implement multiple excitation regions.
- Need to modify the covering constraint to allow the cover to include states from other excitation regions.
  \[
  ER(u*, k) \subseteq [C(u*, k) \cap S] \subseteq \bigcup_l ER(u*, l) \cup QS(u*)
  \]
- Entrance constraint must be modified to allow the cover to be entered from any corresponding excitation region state.
  \[
  [(s_i, l, s_j) \in \delta \land s_i \notin C(u*, k) \land s_j \in C(u*, k)] \Rightarrow s' \in \bigcup_l ER(u*, l)
  \]
- Additional constraint is now necessary to guarantee that a cover either includes an entire excitation region or none of it.
  \[
  ER(u*, l) \not\subseteq C(u*, k) \Rightarrow ER(u*, l) \cap C(u*, k) = \emptyset
  \]

Gate Sharing Example: SG

Example: No Sharing

- $ER(c+, 1) = 100$ and $ER(c+, 2) = 110$.
- Using the earlier constraints, the primes are found to be:
  \[
  P(c+, 1) = \{10-, 1-1, -11\}
  \]
  \[
  P(c+, 2) = \{11-, 1-1, -11\}
  \]
Gate Sharing Example: Original Circuit

Example: Sharing
- $ER(c+, 1) = 100$ and $ER(c+, 2) = 110$.
- Using the new constraints, the primes are found to be:
  - $P(c+, 1) = \{1--, -11\}$
  - $P(c+, 2) = \{1--, -11\}$

The Single Cube Algorithm
- Many region functions composed of a single product, or cube.
- Now present a more efficient algorithm which finds an optimal single-cube cover for each region function, if one exists.
The Single Cube Algorithm

```cpp
single_cube(SG, technology) {
    foreach u ∈ O { /* Consider each output signal. */
        EC = find_excitation_cubes(SG);
        foreach EC(u, k) ∈ EC { /* Find cover for each EC. */
            TC(u, k) = trigger_cube(SG, EC(u, k));
            CS(u, k) = context_signals(SG, EC(u, k), TC(u, k));
            CC = build_cover_table(CS(u, k), V(u, k));
            C(u, k) = solve_cover_table(CC, TC(u, k));
            solution(u) = optimize_logic(C); }
    return solution; }
```

Excitation Cubes

- In a single-cube cover, all literals must correspond to signals that are stable throughout the excitation region.
- ER(u*, k) is approximated using an excitation cube.
- The excitation cube is the supercube of the states in the excitation region and defined on each signal v as follows:
  
  ```
  EC(u*, k)(v) ≡ \begin{cases} 
  0 & \text{if } \forall s \in ER(u*, k) . s(v) = 0 \\
  1 & \text{if } \forall s \in ER(u*, k) . s(v) = 1 \\
  - & \text{otherwise}
  \end{cases}
  ```

- If a signal has a value of 0 or 1 in the excitation cube, the signal can be used in the cube implementing the region.
- The set of states implicitly represented by the excitation cube is always a superset of the set of excitation region states.

Trigger Cubes

- The set of trigger signals for ER(u*, k) can also be represented with a cube called a trigger cube.
- TC(u*, v) is defined as follows for each signal v:
  
  ```
  TC(u*, k)(v) ≡ \begin{cases} 
  s_j(v) & \text{If } \exists (s_i, t, s_j) ∈ \delta . (t = v + v t = v−) \land \\
  (s_i \notin ER(u*, k)) \land (s_j \in ER(u*, k)) \\
  - & \text{otherwise}
  \end{cases}
  ```

- The single cube cover of an excitation region must contain all its trigger signals (i.e., C(u*, k) ⊆ TC(u*, k)).
- Therefore, all trigger signals must be stable (i.e., EC(u*, k) ⊆ TC(u*, k)).

Example: Excitation and Trigger Cubes

![Excitation and Trigger Cubes Diagram](image)
Excitation and Trigger Cubes

<table>
<thead>
<tr>
<th>$u^*, k$</th>
<th>$EC(u^*, k)$</th>
<th>$TC(u^*, k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>c+, 1</td>
<td>0100</td>
<td>-1--</td>
</tr>
<tr>
<td>c+, 2</td>
<td>1101</td>
<td>--11</td>
</tr>
<tr>
<td>c-, 1</td>
<td>0010</td>
<td>-0--</td>
</tr>
<tr>
<td>d+, 1</td>
<td>1100</td>
<td>-1--</td>
</tr>
<tr>
<td>d-, 1</td>
<td>1111</td>
<td>--1--</td>
</tr>
</tbody>
</table>

Violating States: gC Circuits

- In gC circuits, for a set region a state is a violating state when the trigger cube intersects the *falling* or *low* sets.
- Similarly, for a reset region, a state is a violating state when the trigger cube intersects the *rising* or *high* sets.

$V(u^+, k) = \{ s \in S \mid s \in TC(u^+, k) \land s \in ES(u^+) \cup QS(u^-) \}$

$V(u^-, k) = \{ s \in S \mid s \in TC(u^-, k) \land s \in ES(u^-) \cup QS(u^+) \}$

Violating States

- Goal is to find smallest product $C(u^*, k)$ where

  
  $EC(u^*, k) \subseteq C(u^*, k) \subseteq TC(u^*, k)$

  and satisfies the required correctness constraints.

- Begin with a cube consisting only of the trigger signals.

- If this cover contains no states that violate the required correctness constraints, we are done.

- If not, context signals must be added to the cube to remove any violating states.

- For each violation, the procedure determines the choices of context signals which would exclude the violating state.

- Finding smallest set of context signals is a covering problem.

Example: Violating States

State = $\langle abcd \rangle$
Context Signal Choices

- Determine context signals which remove these violating states.
- A signal is allowed to be a context signal if it is stable in the excitation cube (i.e., $EC(u, k)(v) = 0$ or $EC(u, k)(v) = 1$).
- A context signal removes a violating state when it has a different value in the excitation cube and the violating state.
- In other words, a context signal $v$ removes a violating state $s$ when $EC(u, k)(v) = s(v)$.

Example: Context Signals

```
01R0  b+  RR00  a+
  c-    
1R00  b-  00F0
  d+    
0F10  a-  11F1
  c+    
11R1  d-  111F
```

State = \([abc]d\)

Setting Up the Covering Problem

- The constraint matrix has a row for each violating state and a column for each context signal.
- The constraint matrix for $ER(d+, 1)$ is shown below:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>111F</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F110</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>0F10</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>01R0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Gate Level Circuits: Cover Violations

- Gate level circuits have covering and entrance constraints.
- For each $ER(u*, k)$, find all states in the initial cover, $TC(u*, k)$, which violate the covering constraint:
  - A state $s$ in $TC(u*, k)$ is a violating state if:
    - The signal $u$ has the same value but is not excited,
    - Is excited in the opposite direction, or
    - Is excited in the same direction but the state is not in the current excitation region.
Example: Cover Violations

\[ EV(u+, k) = \{ s_j \in S \mid (s_i, v^*, s_j) \in \delta \land s_j \in QS(u+) \land s_j \in TC(u+, k) \land EC(u+, k)(v) = s_i(v) \} \]

Example: Entrance Violations

\[ EV(u-, k) = \{ s_j \in S \mid (s_i, v^*, s_j) \in \delta \land s_j \in QS(u-) \land s_j \in TC(u-, k) \land EC(u-, k)(v) = s_i(v) \} \]

Gate Level Circuits: Entrance Violations

- Must check state transitions for potential entrance violations.
- For each state transition \((s_i, t, s_j)\), this is possible when \(s_j\) is a quiescent state, \(s_j\) is in the initial cover, and \(\lambda_T(t)\) excludes \(s_i\).

\[ EV(u+, k) = \{ s_j \in S \mid (s_i, v^*, s_j) \in \delta \land s_j \in QS(u+) \land s_j \in TC(u+, k) \land EC(u+, k)(v) = s_i(v) \} \]

\[ EV(u-, k) = \{ s_j \in S \mid (s_i, v^*, s_j) \in \delta \land s_j \in QS(u-) \land s_j \in TC(u-, k) \land EC(u-, k)(v) = s_i(v) \} \]

Setting Up the Covering Problem

- Since inclusion of certain context signals cause some states to have entrance violations, the covering problem is binate.
- There is a row in the constraint matrix for each violation and each violation that could arise from a context signal choice.
- There is a column for each context signal.
- The entry in the matrix contains a 1 if the context signal excludes the violating state.
- An entry in the matrix contains a 0 if the inclusion of the context signal would require a new violation to be resolved.
Example: Constraint Matrix

\[
\begin{array}{ccc}
  a & c & d \\
  110R & 1 & - \\
  11R1 & 1 & - 1 \\
  0F10 & 0 & 1 & - \\
  F110 & 1 & 1 & 0 \\
\end{array}
\]

Example: Non-Persistent Trigger Signals

Example: Non-Persistent Trigger Signals

Example: Unresolvable Violations
Hazard-Free Decomposition

- Synthesis method put no restrictions on the size of the gates.
- There is always some limitation on the number of inputs.
- In CMOS, no more than 4 transistors can be in series.
- Large transistor stacks can have charge sharing problems.
- Necessary to decompose high-fanin gates.
- For Huffman circuits, decomposition of high-fanin gates can be done in an arbitrary fashion preserving hazard-freedom.
- For Muller circuits, this problem is much more difficult.

Example: Decomposition I

Example: Decomposition II

Hazard-Free Decomposition Overview

- Special care needed to guarantee a hazard-free decomposition.
- Need to find new internal signal that produces simpler circuit.
- Present here a simple technique for finding hazard-free decompositions using insertion points.
Example: Insertion Points

Slide 129

Transition Point Filters

- Requirements on transition points \((t_s, t_e)\):
  1. The start and end sets should be disjoint. 
    \((i.e., t_s \cap t_e = \emptyset)\)
  2. The end set should not include input transitions. 
    \((i.e., \forall t \in t_e . t \notin T)\)
  3. Start and end sets should only include concurrent transitions. 
    \((i.e., \forall t_1, t_2 \in t_s . t_1 \parallel t_2 \text{ and } \forall t_1, t_2 \in t_e . t_1 \parallel t_2)\)

Transition Point Filters for Decomposition

- Consider decomposition of \(C(u*\bar{k})\) composed of a single cube.
- Restrict the start set for one transition point to transitions on just those signals in the gate being decomposed.
  - Consider all possible combinations of the trigger signals.
  - Only consider concurrent subsets of the context signals.
- Only consider transitions that occur after those in the start set and before \(u*\) as potential candidates to be in the end set.
Transition Point Filters for Decomposition

- If both a set and reset regions of $u$ must be decomposed, use same restrictions for reverse transition on the new signal.
- If not:
  - Start set should include concurrent transitions which occur after $u$ and before any transitions in the first start set.
  - Including the reverse transition of $u$ in the end set is often useful, but any transition after $u$ could be used.

Algorithm for Decomposition

```
def decomposition(SG, design, maxsize):
    HF = findHighFaninGates(design, maxsize);
    if (len(HF) == 0) return design; /* No high-fanin gates, return.*/
    best = len(HF); best_JP = design; /* Initialize best found.*/
    TP = findAllTransitionPoints(SG, design, HF);
    foreach TP_R, TP_F in TP
        foreach TP_R, TP_F in TP
            if JP = (TP_R, TP_F) is legal then {
                CSG = colorStateGraph(SG, TP_R, TP_F);
                if (CSG is consistent) then {
                    SG* = insertStateSignal(SG, JP);
                    design = synthesis(SG*); /* Find new circuit */
                    HF* = findHighFaninGates(design, maxsize);
                    if (len(HF*) < best) or (len(HF) = best) and
cost(design) < cost(best_JP)) then {
                        best = len(HF*);
                        best_JP = design; }
                design = decomposition(SG, design); /* Add more signals */
            return design; }
    return design;
```
Rising Transition Point Choices

\{(CSC0-), \{ack\_wine+\}\}
\{(\text{req\_patron}-), \{ack\_wine+\}\}
\{(\text{req\_wine}+), \{ack\_wine+\}\}
\{(\text{ack\_patron}-), \{ack\_wine+\}\}
\{(\text{req\_wine}+), \{ack\_patron}-, \{ack\_wine+\}\}

Falling Transition Point Choices

\{(\text{ack\_wine}+), \{CSC0+\}\}
\{(\text{ack\_wine}+), \{ack\_wine-\}\}
\{(CSC0+), \{ack\_wine-\}\}
\{(\text{req\_wine}-), \{CSC0+\}\}
\{(\text{req\_wine}-), \{ack\_wine-\}\}
\{(CSC0+, \text{req\_wine}-), \{ack\_wine-\}\}

Checking the Insertion Points

- Form insertion points out of combinations.
- Color the graph to determine if the insertion point leads to a consistent state assignment.
- Check if any USC violations become CSC violations.
- If okay, derive a new state graph and synthesize the circuit.
- If new circuit meets the fanin constraints, then accept.
- If not, try the next insertion point.
Summary

- Formal definition of speed independence.
- Complete state coding.
- Hazard-free logic synthesis of Muller circuits.
- Hazard-free decomposition.