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.

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.

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.
Asynchronous Circuit Design

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.

Simple Example Complete Circuit

- Consider a complete circuit composed of four states, \(S = \{a, b, c, d\}\), which has the following two allowed sequences:
  - \(a, b, a, b, \ldots\)
  - \(a, c, d\)
- The sequences above imply the following allowed sequences:
  - \(b, a, b, a, \ldots\)
  - \(c, d\)
  - \(d\)
  - \(a, b, a, c, d\)
  - \(a, b, a, b, a, c, d\)
  - \(b, a, c, d\)
  - etc.

State Diagram For Simple Example

\[
\begin{array}{ccc}
  a & \rightarrow & b \\
  \downarrow & & \downarrow \\
  c & \rightarrow & d \\
\end{array}
\]

\(\mathcal{R}\)-related and \(\mathcal{R}\)-sequences

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

The Followed and Equivalence Relations

- A state \(s_i\) is followed by a state \(s_j\) (denoted \(s_i \mathcal{F} s_j\)) if there exists an \(\mathcal{R}\)-sequence \((s_i, \ldots, s_j)\).
- The \(\mathcal{F}\)-relation is reflexive and transitive, but not necessarily symmetric.
- If two states \(s_i\) and \(s_j\) are symmetric under the \(\mathcal{F}\)-relation (i.e., \(s_i \mathcal{F} s_j\) and \(s_j \mathcal{F} s_i\)), they are said to be equivalent (denoted \(s_i \mathcal{E} s_j\)).

Equivalence Classes

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

---

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

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.

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.
A Non-Semi-Modular Example

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 transitions can disable other input signal transitions.

∀t₁ ∈ T₀ ∃s ∈ S : (s₁, t₁, s₂) ∈ δ \(→ \) \(∃s ∈ S : (s₁, t₂, s₃) ∈ δ \wedge (s₂, t₁, s₃) \in δ \)

\[ T₀ = \{ u^+ | u ∈ O \} \]

Output Semi-Modularity Example

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:

\[ ES(u^+) = \{ s ∈ S | s(u) = 0 \wedge u ∈ X(s) \} \]
\[ ES(u^-) = \{ s ∈ S | s(u) = 1 \wedge u ∈ X(s) \} \]

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:

\[ QS(u^+) = \{ s ∈ S | s(u) = 1 \wedge u \notin X(s) \} \]
\[ QS(u^-) = \{ s ∈ S | s(u) = 0 \wedge u \notin X(s) \} \]
**Asynchronous Circuit Design**

**Excitation and Quiescent States Example**

\[ \text{ES}(y^+) = \{ \langle RR0 \rangle, \langle 1R0 \rangle \} \]

\[ \text{ES}(y^-) = \{ \langle FF1 \rangle, \langle 0F1 \rangle \} \]

**Excitation Regions**

- An excitation region for signal \( u \) is a maximally connected subset of either \( \text{ES}(u^+) \) or \( \text{ES}(u^-) \).
- If it is a subset of \( \text{ES}(u^+) \), it is a set region (denoted \( \text{ER}(u^+, k) \)).
- Similarly, a reset region is denoted \( \text{ER}(u^-, k) \).

**Switching Regions**

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

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

**Distributive State Graphs**

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

\[ \forall \text{ER}(u, k). \exists! s_i \in \text{ER}(u, k). \neg \exists s_t \in \text{ER}(u, k). (s_i, t, s_t) \in \delta \]

**A Distributive State Graph**

- \( \text{ES}(y^+) = \{ \langle RR0 \rangle, \langle 1R0 \rangle \} \)
- \( \text{ES}(y^-) = \{ \langle FF1 \rangle, \langle 0F1 \rangle \} \)
- \( \text{ER}(y^+, 1) = \{ \langle RR0 \rangle, \langle 1R0 \rangle \} \)
- \( \text{ER}(y^-, 1) = \{ \langle FF1 \rangle, \langle 0F1 \rangle \} \)
- \( \text{SR}(y^+, 1) = \{ \langle R10 \rangle, \langle 11R \rangle \} \)
- \( \text{SR}(y^-, 1) = \{ \langle F01 \rangle, \langle 00F \rangle \} \)
Asynchronous Circuit Design

Chris J. Myers (Lecture 6: Muller Circuits)

### A Non-Distributive State Graph

```
\text{ER}(u^*,k) = \{ v \in N | \exists s_i, s_j \in S.((s_i, t, s_j) \in \delta) \\
\land (t = v + \lor t = v-) \\
\land (s_i \notin \text{ER}(u^*,k)) \land (s_j \in \text{ER}(u^*,k))\}
```

### 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 \(\text{ER}(u^*,k)\) is:

\[
\text{CS}(u^*,k) = \{ v \in N | v \notin \text{TS}(u^*,k) \\
\land \forall s_i, s_j \in \text{ER}(u^*,k).s_j(i) = s_i(i)\}
\]

### 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 \(\text{ER}(u^*,k)\) is:

\[
\text{TS}(u^*,k) = \{ v \in N | \exists s_i, s_j \in S.((s_i, t, s_j) \in \delta) \\
\land (t = v + \lor t = v-) \\
\land (s_i \notin \text{ER}(u^*,k)) \land (s_j \in \text{ER}(u^*,k))\}
\]

### Trigger and Context Signals Example

\[
\text{ER}(y^+,1) = \{ z \} \\
\text{ER}(y^-,1) = \{ z \} \\
\text{SR}(y^+,1) = \{ y \} \\
\text{SR}(y^-,1) = \{ y \}
\]

### The Passive/Active Wine Shop: Petri-net

- \text{req\_patron}\-
- \text{ack\_patron}\+
- \text{req\_wine}\+
- \text{ack\_wine}\-
- \text{req\_patron}\+
- \text{ack\_patron}\-
- \text{req\_wine}\-
- \text{ack\_wine}\+

### The Passive/Active Wine Shop: State Graph

\[
\begin{align*}
\text{ES}(\text{req\_patron}+) &= \{(R00R), (100R)\} \\
\text{ES}(\text{req\_patron}-) &= \{(R01F), (110F)\} \\
\text{QS}(\text{req\_patron}+) &= \{(R01R), (110R)\} \\
\text{QS}(\text{req\_patron}-) &= \{(RF00), (1F00)\} \\
\text{ER}(\text{req\_patron}+; 1) &= \{(R00R), (100R)\} \\
\text{ER}(\text{req\_patron}--; 1) &= \{(R01F), (110F)\} \\
\text{SR}(\text{req\_patron}+; 1) &= \{(RF00), (1F00)\} \\
\text{TS}(\text{req\_patron}+; 1) &= \{\text{ack\_wine}\} \\
\text{TS}(\text{req\_patron}--; 1) &= \{\text{ack\_patron}\} \\
\text{CS}(\text{req\_patron}+; 1) &= \{\text{ack\_patron}\} \\
\text{CS}(\text{req\_patron}--; 1) &= \{\text{ack\_wine}, \text{req\_patron}\}
\end{align*}
\]
Unique State Codes (USC)

- Two states have unique state codes (USC) if they are labeled with different binary vectors.
  \[ USC(s_i, s_j) \iff \lambda_2(s_i) \neq \lambda_2(s_j) \]
- A SG has USC if all states pairs have USC.
  \[ USC(S) \iff \forall (s_i, s_j) \in S \times S . \ USC(s_i, s_j) \]

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.
  \[ CSC(s_i, s_j) \iff USC(s_i, s_j) \vee X(s_i) \cap O = X(s_j) \cap O \]
  \[ CSC(S) \iff \forall (s_i, s_j) \in S \times S . \ CSC(s_i, s_j) \]
- A set of state pairs which violate CSC is defined as:
  \[ CSCV(S) = \{(s_i, s_j) \in S \times S \mid \neg CSC(s_i, s_j)\} \]

The Passive/Active Wine Shop: State Graph

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 reschedule the protocol as described earlier.
- Now introduce method for inserting state variables.

Insertion Points

- Need to insert a rising and falling transition for new signal.
- A transition point is \( TP = (t_p, t_e) \), where \( t_p \) 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 \( t_p \) 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 Passive/Active Wine Shop: Petri-net
Asynchronous Circuit Design

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 \( \cap \text{SR}(t) \).
- 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:

\[
S(TP) = \{ s_i \in S | s_i \in \cap S(t) \lor (\exists(s, t, t_i) \in \delta . s_i \in S(TP) \land t \notin t_e)\}
\]

Insertion Point Explosion

- The set of all possible insertion points includes all combinations of transitions in \( t_r \) 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.

Transition Point Restrictions

- A transition point must satisfy the following three restrictions:
  - Start and end sets are disjoint [i.e., \( t_r \cap t_e = \emptyset \)].
  - End set does not include input transitions [i.e., \( \forall t \in t_e . t \notin T \)].
  - Start and end sets include only concurrent transitions [i.e., \( \forall t_1, t_2 \in t_r . t_1 \parallel t_2 \) and \( \forall t_1, t_2 \in t_e . t_1 \parallel t_2 \)].

Transition Point Examples

\[
\begin{align*}
\{\text{ack\_wine}+, \text{ack\_wine}^{-}\} \\
\{\text{req\_patron}+, \text{ack\_patron}^{-}\} \\
\{\text{ack\_wine}+, \text{ack\_wine}^{-}\} \\
\{\text{req\_wine}+, \text{req\_patron}^{-}\} \\
\{\text{ack\_patron}+, \text{req\_patron}^{-}\} \\
\{\text{ack\_wine}+, \text{req\_patron}^{-}\} \\
\{\text{req\_wine}+, \text{req\_patron}^{-}\} \\
\{\text{req\_wine}+, \text{ack\_patron}^{-}\} \\
\{\text{ack\_wine}+, \text{ack\_wine}^{-}\} \\
\{\text{req\_wine}+, \text{ack\_wine}^{-}\} \\
\{\text{req\_patron}+, \text{req\_patron}^{-}\} \\
\{\text{req\_patron}+, \text{ack\_wine}^{-}\} \\
\{\text{req\_wine}+, \text{ack\_wine}^{-}\} \\
\{\text{req\_patron}+, \text{req\_patron}^{-}\} \\
\{\text{req\_wine}+, \text{req\_patron}^{-}\} \\
\{\text{req\_wine}+, \text{ack\_patron}^{-}\} \\
\end{align*}
\]

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:

\[
TP_R(t_r) \cap TP_F(t_e) \neq \emptyset \\
TP_R(t_e) \cap TP_F(t_r) \neq \emptyset
\]

- An incompatible insertion point always creates an inconsistent state assignment.

Example:

\[
IP = (\{\text{ack\_wine}+, \text{ack\_wine}^{-}\}, \{\text{req\_wine}+, \text{req\_patron}^{-}\}, \{\text{ack\_wine}^{-}\})
\]
**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.

**State Graph Coloring Procedure**

- States in S(TPr) are colored as rising.
- States in S(TPf) 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.

---

**The Colored Passive/Active Wine Shop SG**

- The primary component of the cost function is the number of remaining CSC violations after a state signal is inserted.
- Eliminate from CSCV any pair of violations in which one state is colored high while the other is colored low.
- States with a USC violation may now have a CSC violation.
- For each pair of states with a USC violation (but not a CSC violation), if one is colored rising while the other is colored low, there is now a CSC violation.
- Similarly, if one is colored falling and the other is colored high, there is also a new CSC violation.
- Each new CSC violation must be added to the total remaining.
The IP with the smallest sum $|TP_R(t_o)| + |TP_F(t_o)|$.
The IP with the smallest sum $|TP_R(t_o)| + |TP_F(t_o)|$.

State signal can be inserted into a Petri-net by adding arcs from each transition in $t_o$ 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.

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

```
csc_solver(SG) {
    CSCV = find_csc_violations(SG);
    if (|CSCV| = 0) return SG;
    best = CSCV;
    best = (0, 0);
    TP = find_all_transition_points(SG);
    foreach TP(e in TP
        if IP = (TP_R, TP_F) is legal then {
            CSG = color_state_graph(SG, TP_R, TP_F);
            CSCV = find_csc_violations(CSG);
            if (CSG is consistent) and ((CSCV < best) or (|CSCV| = best) and (cost(IP) < cost(best))) then {
                best = (CSCV);
                best = (TP_R, TP_F);
            }
        } SG = insert_state_signal(SG, best); SG = csc_solver(SG);
    return SG;
}
```
Hazard-free Logic Synthesis

- Requires modified minimization to obtain hazard-free logic.
- Modifications needed are dependent upon technology.

We consider the following technologies:

- Complex gates
- Generalized C-elements
- Basic gates

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 the 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.

\[
\text{ON-set} = \{s \in (ES(u^+) \cup QS(u^+) ) | s \in \lambda_s \}
\]

\[
\text{OFF-set} = \{s \in (ES(u^-) \cup QS(u^-) ) | s \in \lambda_s \}
\]

\[
\text{DC-set} = \{(0,1)^N - (\text{ON-set} \cup \text{OFF-set})\}
\]

- Find primes using recursive procedure described earlier.
- Setup and solve a covering problem.

Atomic Gate: Example

\[
\begin{align*}
\text{ON-set} &= \{10000, 10100, 00100, 10101\} \\
\text{OFF-set} &= \{00101, 00001, 10011, 10011, 00100, 10010, 01010, 11010, 01000, 11000, 11011, 00000\} \\
\text{DC-set} &= \{01110, 01111, 01010, 01110, 01111, 10110, 10111, 11001, 11100, 11110, 11111\}
\end{align*}
\]

\[
P = \{1-1, -1, -1, -1, -1, -1, 0, -1, 0, 0, 0, 1, 0\}
\]

\[
\begin{array}{cccccccc}
1 & 1 & -1 & -1 & -1 & -1 & -1 & 0 & 0, 0, 1 \\
10000 & - & - & - & - & - & - & 1 \\
10100 & 1 & - & - & - & 1 & - & 1 \\
00100 & - & - & - & 1 & - & - & - \\
10101 & 1 & - & - & - & - & - & - \\
\end{array}
\]

Passive/Active Wine Shop: Atomic Gate

Passive/Active Shop: Atomic Gate Circuit
Asynchronous Circuit Design

**Generalized C-Elements**

![Diagram of Generalized C-Elements]

**gC Logic Synthesis**

- Two minimization problems must be solved for each signal \( u \): set of the function (i.e., \( \text{set}(u) \)) and reset (i.e., \( \text{reset}(u) \)).
- 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.

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

- Can now apply standard methods to find a minimum number of primes to implement the set and reset functions.

**gC Circuit: Example**

\[
\begin{align*}
\text{ON-set} & = \{ 10000 \} \\
\text{OFF-set} & = \{ 00010, 00001, 10001, 00111, 10011, 01010, 00010, 10010, 01010, 11010, 01000, 11000, 11011, 00000 \} \\
\text{DC-set} & = \{ 00110, 11101, 01110, 01101, 01111, 10110, 10111, 11000, 11011, 11100, 11111, 10100, 01010 \}
\end{align*}
\]

\[
P = \{ -1, -1, -1, -1, -1, 0, -1, 0, 1, 0 \}
\]

**Passive/Active Wine Shop: gC**

![Diagram of Passive/Active Wine Shop]

**Passive/Active Shop: gC Circuit**

- \( \text{ack}_\text{wine} \)
- \( \text{req}_\text{patron} \)
- \( \text{CSC0} \)
- \( \text{ack}_\text{wine} \)
- \( \text{CSC0} \)
- \( \text{C} \)
- \( \text{req}_\text{patron} \)
- \( \text{CSC0} \)
- \( \text{CSC0} \)
A Simple Example

- 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.

Combinational Optimization

- If $\text{set}(u)$ 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 $\text{set}(u)$.
- If $\text{reset}(u)$ is on in all states in which $u$ should be falling or low, then the signal $u$ can be implemented with $\text{reset}(u)$.

Gate-Level Hazard

- 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.

Region Function Operation

- Requires a modified logic minimization procedure.
Asynchronous Circuit Design

**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 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.

$$ER(u^+, k) \subseteq \left[ C(u^+, k) \cap S \right] \subseteq [ER(u^+, k) \cup QS(u^+)]$$

**Entrance Constraint**

A cover must only be entered through excitation region states.

$$[(s_i, t, s_j) \in \delta \land s_i \not\in C(u^+, k) \land s_j \in C(u^+, k)] \implies s_j \in ER(u^+, k)$$

**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.

$$[c \cap S] \subseteq [ER(u^+, k) \cup QS(u^+)]$$

**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.

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

$$\begin{align*}
\text{DC-set} &= \{ \lambda_d(s) \mid s \in (ES(u^-) \cup QS(u^-)) \} \\
\text{DC-set} &= \{ \lambda_d(s) \mid s \in (ES(u^-) \cup ER(u^+, k)) \} \\
\end{align*}$$

$$\begin{align*}
\text{DC-set} &= \{ \lambda_d(s) \mid s \in (ES(u^-) \cup QS(u^-)) \} \\
\text{DC-set} &= \{ \lambda_d(s) \mid s \in (ES(u^-) \cup ER(u^+, k)) \} \\
\end{align*}$$

$$\begin{align*}
\text{DC-set} &= \{ \lambda_d(s) \mid s \in (ES(u^-) \cup QS(u^-)) \} \\
\text{DC-set} &= \{ \lambda_d(s) \mid s \in (ES(u^-) \cup ER(u^+, k)) \} \\
\end{align*}$$
Gate Level Logic Synthesis: Reset Regions

- For a reset region $ER(u^-, k)$:
  
  \[
  \begin{align*}
  \text{ON-set} & = \{ \lambda_S(s) | s \in (ER(u^-, k)) \} \\
  \text{OFF-set} & = \{ \lambda_S(s) | s \in (ES(u^+) \cup QS(u^+)) \cup (ES(u^-) \cup \overline{ER(u^-, k)}) \} \\
  \text{DC-set} & = \{ 0,1 \}^{|N|} - (\text{ON-set} \cup \text{OFF-set})
  \end{align*}
  \]

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 $ER(c^+, 1)$.
  
  \[
  \begin{align*}
  \text{ON-set} & = \{ 0100 \} \\
  \text{OFF-set} & = \{ 0000, 1000, 0010, 1100, 1101 \} \\
  \text{DC-set} & = \{ 0011, 0111, 1011, 1110, 1111 \}
  \end{align*}
  \]

- The primes found are as follows:
  
  \[
  P = \{ 01-, 1-, -1-, -1-, -1-, 0-, 0-, -1- \}
  \]

Implied States

- The entrance constraint creates a set of implied states for each implicant $c$ (denoted $IS(c)$).
- A state 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 $ER(u^+, k)$.
  
  \[
  IS(c) = \{ s_i | s_i \notin c \land \exists s_j (s_i,t,s_j) \in \delta \land (s_j \in c) \land (s_j \notin ER(u^+, k)) \}
  \]

- This means that the product $c$ becomes excited in a quiescent state instead of an excitation region state.
- If there is 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.

Existence of a Prime Cover: Example

- The primes found are as follows:
  
  \[
  P = \{ 01-, 1-, -1-, -1-, -1-, 0-, 0-, -1- \}
  \]
## 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:
  
  \[ c_j \supset c_i \]
  
  \[ IS(c_j) \subseteq IS(c_i). \]

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

## Candidate Implicant Algorithm

```plaintext
candidate_implementants(SG, P){
  done = 0
  for (k = |largest(P)|; k ≥ 1; k −−) {
    foreach (q ∈ P | q = k) { enqueue(c, q) }
    foreach (c ∈ C | |q| ≥ k) {
      if (IS(SG, c) = 0) continue
      foreach (s ∈ IS(extend(c))) {
        if (s ∈ done) continue
        IS = IS(SG, s)
        prime = true
        foreach (q ∈ C | q = k) {
          if (s ⊂ IS(q)) {
            IS = IS(SG, q)
            if (prime = false; break) {
              break
            }
          }
        }
        if (prime = 1) enqueue(C, s)
        done = done ∪ \{s\}]
    }
  }
}
```

## Covering Clauses

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

\[ \lor_{j \in C} x_j. \]

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

\[ (x_1 + x_2) \]

## 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.

## Closure Clauses

- For each candidate implicant $c_i$, a closure clause is constructed for each of its implied states $s \subseteq IS(c_i)$.
- Each closure clause represents an implication if a candidate implicant used, its implied states must be covered.

\[ \bar{x} \lor \lor_{j \in C} x_j. \]

- The candidate implicant $c_1$ (01$\rightarrow$) has implied state 0110.
- 0110 included in implicants $c_2$ (1$\rightarrow$) and $c_3$ (1$\rightarrow$).

\[ (\bar{x} + x_1 + x_3) (\bar{x} + x_3 + x_5) (\bar{x} + x_5) \]

- Complete formula: $(x_1 + x_2)(\bar{x} + x_1 + x_3)(\bar{x} + x_3 + x_5) (\bar{x} + x_5)$
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_j, with a 0 in the column corresponding to c_j and a 1 in each column with a candidate implicant c_i that covers s.

Combinational Optimization

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

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.

Constraint Matrix for ER(c+,1)

<table>
<thead>
<tr>
<th></th>
<th>01-</th>
<th>010-</th>
<th>1-1-</th>
<th>101-</th>
<th>11-1</th>
<th>0-1-</th>
<th>0-1</th>
<th>1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:
  \[ P(c_+,1) = \{10-,-11\} \]
  \[ P(c_+,2) = \{11-,-11\} \]

Gate Sharing Example: Original Circuit

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.

Single Cube Algorithm

```c
single_cube(SG, technology) {
  foreach u ∈ O {
    EC = find_excitation_cubes(SG);
    foreach u ∈ EC {
      TC(u, k) = trigger_cube(SG, EC(u, k));
      CS(u, k) = context_signals(SG, EC(u, k), TC(u, k));
      V(u, k) = violations(SG, EC(u, k), TC(u, k), tech);
      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; }
}
```

Example: Sharing

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

Gate Sharing Example: Optimized Circuit

Chris J. Myers (Lecture 6: Muller Circuits) Asynchronous Circuit Design
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:

$$EC(u^+,k)(v) = \begin{cases} 
0 & \text{if } \forall s \in ER(u^+,k), s(v) = 0 \\
1 & \text{if } \exists s \in ER(u^+,k), s(v) = 1 \\
& \text{otherwise}
\end{cases}$$

- 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.

Example: Excitation and Trigger Cubes

Excitation and Trigger Cubes

- 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.

\[
\begin{align*}
V(u^+,k) &= \{ s \in S | s \in TC(u^+,k) \land massEC(u^+,k) \cup ES(u^+) \cup QS(u^+) \} \\
V(u^-,k) &= \{ s \in S | s \in TC(u^-,k) \land massEC(u^-,k) \cup ES(u^+) \cup QS(u^+) \}
\end{align*}
\]
Example: Violating States

\[
\begin{array}{cccc}
01R0 & c+ & b- & a+ \\
0F10 & b+ & 00F0 & 1R00 \\
a- & d+ & c+ & 1R1 \\
F110 & d- & c- & 111F \\
\end{array}
\]

State = \text{abcd}

Example: Context Signals

\[
\begin{array}{cccc}
01R0 & c+ & b- & a+ \\
0F10 & b+ & 00F0 & 1R00 \\
a- & d+ & c+ & 1R1 \\
F110 & d- & c- & 111F \\
\end{array}
\]

State = \text{abcd}

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 \(EC(u_*, k)(v) = s(v)\).

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:

\[
\begin{array}{cccc}
\text{a} & \text{c} & \text{d} \\
111F & 1 & 1 \\
F110 & 1 & 1 \\
0F10 & 1 & 1 \\
01R0 & 1 & 1 \\
\end{array}
\]

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

\[
\begin{array}{cccc}
01R0 & c+ & b- & a+ \\
0F10 & b+ & 00F0 & 1R00 \\
a- & d+ & c+ & 1R1 \\
F110 & d- & c- & 111F \\
\end{array}
\]

State = \text{abcd}
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 context signal choice.
- An 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 context signal excludes the violating state.
- The constraint matrix for \( ER(c+, 1) \) is shown below:

\[
\begin{array}{cccc}
\text{a} & \text{c} & \text{d} \\
11R & 1 & - & - \\
11F & 1 & 1 & - \\
0F10 & 0 & 1 & - \\
F110 & 1 & 1 & 0 \\
\end{array}
\]

Example: Constraint Matrix

Example: Non-Persistent Trigger Signals

Example: Non-Persistent Trigger Signals

Example: Entrance Violations

Example: Entrance Violations

Gate Level Circuits: Entrance Violations

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

\[
EV(u+, k) = \{ s_j \in S \mid (s, u, s_j) \in \delta \land s_j \in QS(u+) \land s_j \in EC(u+, k)(v) = \overline{a(v)} \}
\]

\[
EV(u-, k) = \{ s_j \in S \mid (s, u, s_j) \in \delta \land s_j \in QS(u-) \land s_j \in EC(u-, k)(v) = \overline{a(v)} \}
\]

- For each potential entrance violation, a context signal must be added which excludes \( s_j \) from the cover when \( \lambda_T(t) \) is included.
- If \( \lambda_T(t) \) is a trigger signal, then the state \( s_j \) is a violating state.
- If \( \lambda_T(t) \) is a possible context signal choice, then \( s_j \) becomes a violating state when \( \lambda_T(t) \) is included in the cover.
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

Example: Insertion Points

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.
Transition Point Filters for Decomposition

- Consider decomposition of \( C(u^*, k) \) 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.
- Consider concurrent subsets of the context signals.
- Only consider transitions that occur after those in the start set and before any transitions in the first start set.

Algorithm for Decomposition

```plaintext
def decomposition(SG, design, maxsize):
    HF = find_high_fanin_gates(design, maxsize);
    best = [HF]; best_weight = design;
    TP = find_all_transition_points(SG, design, HF);
    foreach TP in TP:
        if IP = {TPH, TP} is legal then {
            CSG = color_state_graph(SG, TPH, TP);
            if (CSG is consistent) then {
                SG = insert_state_signal(SG, IP);
                design = synthesis(SG);
                HF = find_high_fanin_gates(design, maxsize);
                if ([HF] is best) or ([HF] > best) and
                    (cost(design) < cost(best_weight)) then {
                best = [HF]; best_weight = design;
            }
            design = decomposition(SG, design);
        return design;
    ```

Passive/Active Shop: State Graph
Passive/Active Shop: gC Circuit

Rising Transition Point Choices

\[
(\{\text{CSC0}^-, \text{ack}_\text{wine}^+\})
\]

\[
(\{\text{req}_\text{patron}^-, \text{ack}_\text{wine}^+\})
\]

\[
(\{\text{req}_\text{wine}^+, \text{ack}_\text{wine}^+\})
\]

\[
(\{\text{ack}_\text{patron}^-, \text{ack}_\text{wine}^+\})
\]

Falling Transition Point Choices

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.