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.

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.

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:
  1. $a, b, a, b, \ldots$
  2. a, c, d
- The sequences above imply the following allowed sequences:

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. $b, a, b, a, \ldots$

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

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

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

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

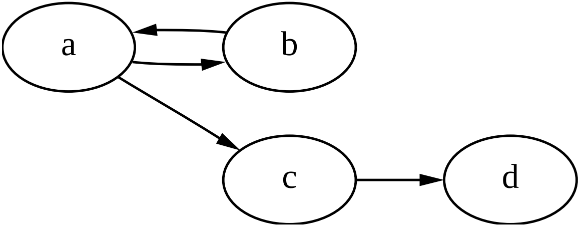
Simple Example Complete Circuit

- 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

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.

State Diagram For Simple Example



$\mathcal{R}$-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:
  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 $s_i \mathcal{R} s_{i+1}$ 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 $s_i \mathcal{E} s_j$).

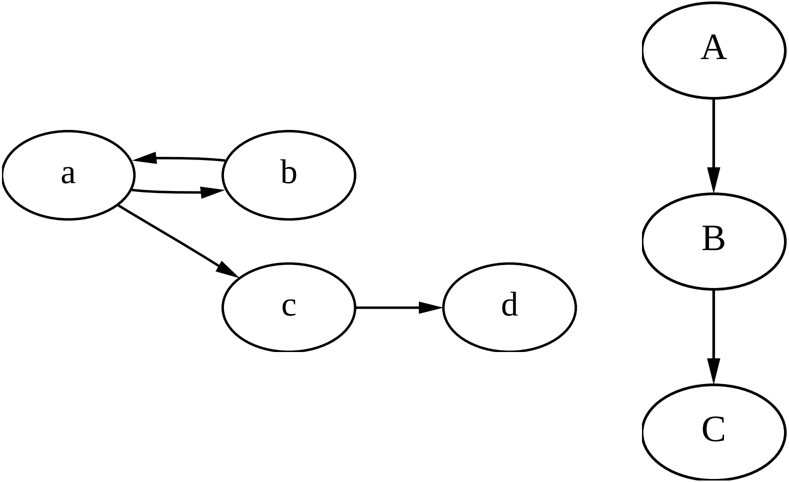
Equivalence Classes

- 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 $a \mathcal{F} b$.

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.

Equivalence Classes for Simple Example



Allowed Sequences on State Graphs

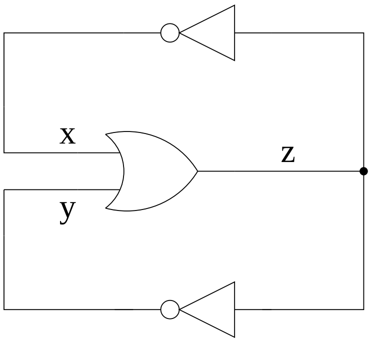
- An allowed sequence of states ($s_1, s_2, \ldots$) is any sequence of states satisfying the following three conditions:
  1. No two consecutive states s_i and s_{i+1} are equal.
  2. For any state s_{j+1} and signal u_i one of the following is true:

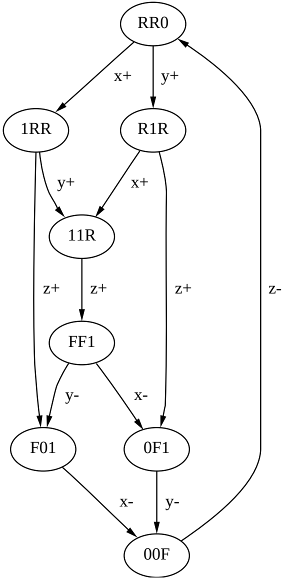
$$\begin{aligned} s_{j+1}(i) &= s_j(i) \\ s_{j+1}(i) &= s'_j(i) \end{aligned}$$

  3. If there exists a signal u_i and a state s_j such that $s_j(i) = s_r(i)$ and $s'_j(i) = s'_r(i)$ for all s_r in the sequence following s_j, then

$$s_j(i) = s'_j(i)$$

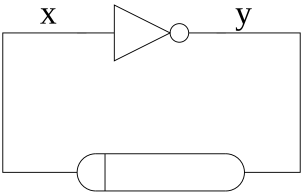
Simple Speed-Independent Circuit

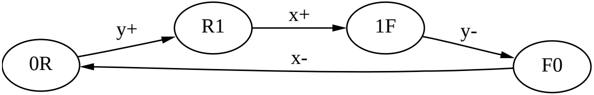




Totally Sequential

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





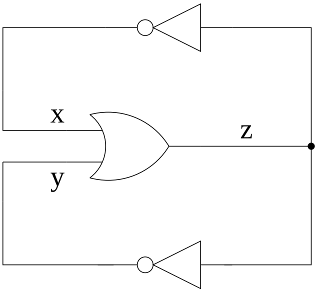
Semi-Modularity

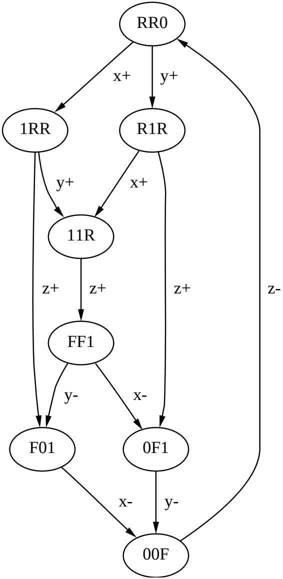
- A circuit is *semi-modular* if in each state in which multiple signals are excited, that in the states reached after one signal has transitioned, that the remaining signals are still excited.

$$\begin{aligned} & \forall t_1, t_2 \in T \,.\, (s_i, t_1, s_j) \in \delta \wedge (s_i, t_2, s_k) \in \delta \\ \Rightarrow \quad & \exists s_l \in S \,.\, (s_j, t_2, s_l) \in \delta \wedge (s_k, t_1, s_l) \in \delta \end{aligned}$$

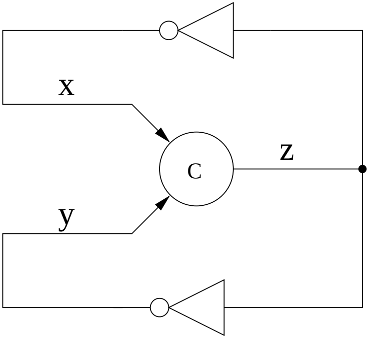
- A totally sequential circuit is semi-modular but the converse is not necessarily true.
- A semi-modular circuit is also speed independent, but again the converse is not necessarily true.

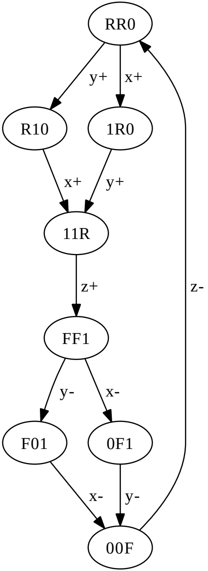
A Non-Semi-Modular Example





A Simple Semi-Modular Speed Independent Circuit





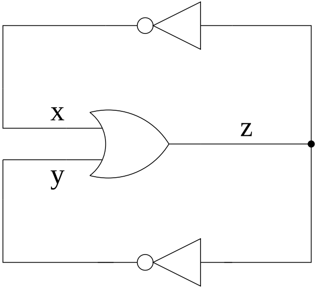
Output Semi-Modularity

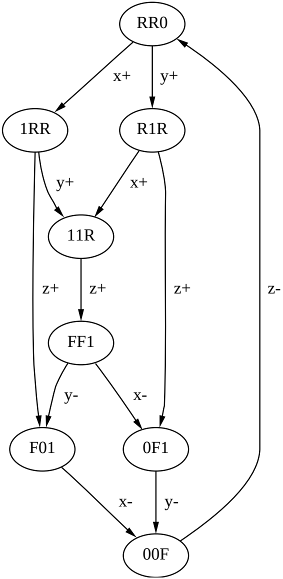
- Input transitions are typically allowed to be disabled by other input transitions, so another useful class of circuits are those which are *output semi-modular*.
- A SG is output semi-modular if only input signal transitions can disable other input signal transitions.

$$\begin{array}{ll} & \forall t_1 \in T_O \, . \, \forall t_2 \in T \, . \, (s_i, t_1, s_j) \in \delta \wedge (s_i, t_2, s_k) \in \delta \\ \Rightarrow & \exists s_l \in S \, . \, (s_j, t_2, s_l) \in \delta \wedge (s_k, t_1, s_l) \in \delta \end{array}$$

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

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:

$$\begin{aligned} ES(u+) &= \{s \in S \mid s(u) = 0 \land u \in X(s)\} \\ ES(u-) &= \{s \in S \mid s(u) = 1 \land u \in X(s)\} \end{aligned}$$

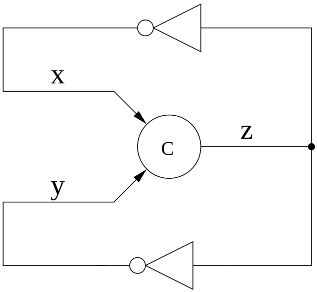
- Recall that $X(s)$ is the set of signals that are excited in state s.

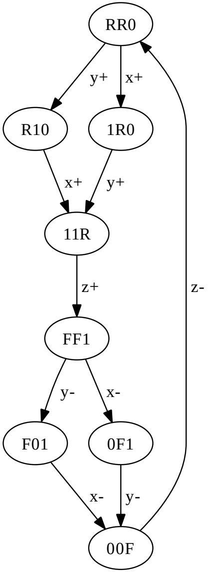
Quiescent States

- For each signal u, there are two sets of *quiescent states*.
- The sets $QS(u+)$ and $QS(u-)$ are defined as follows:

$$\begin{aligned} QS(u+) &= \{s \in S \mid s(u) = 1 \wedge u \notin X(s)\} \\ QS(u-) &= \{s \in S \mid s(u) = 0 \wedge u \notin X(s)\} \end{aligned}$$

Excitation and Quiescent States Example



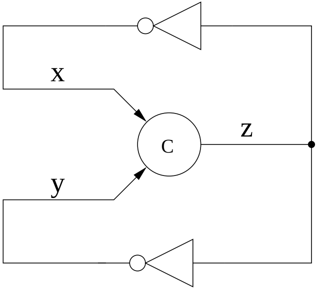


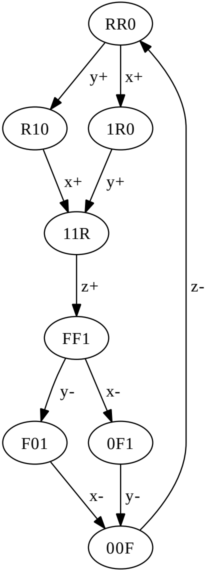
$ES(y+) =$

$ES(y-) =$

$QS(y+) =$

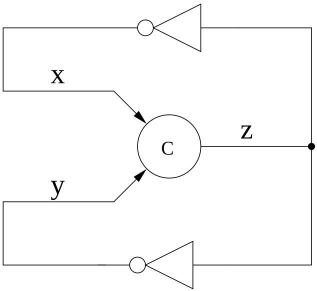
$QS(y-) =$



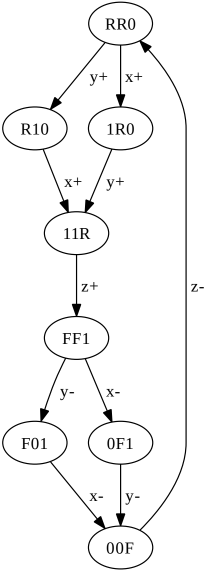


$$ES(y+) = \{\langle RR0 \rangle, \langle 1R0 \rangle\}$$
$$ES(y-) =$$
$$QS(y+) =$$
$$QS(y-) =$$

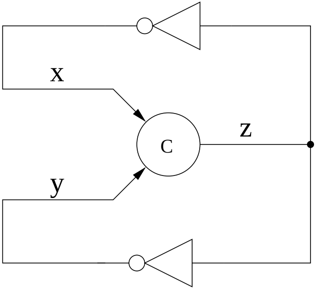
Excitation and Quiescent States Example

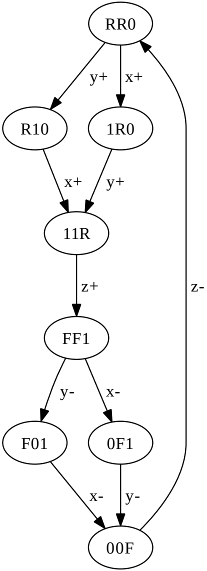


$$ES(y+) = \{\langle RR0\rangle, \langle 1R0\rangle\}$$
$$ES(y-) = \{\langle FF1\rangle, \langle 0F1\rangle\}$$
$$QS(y+) =$$
$$QS(y-) =$$



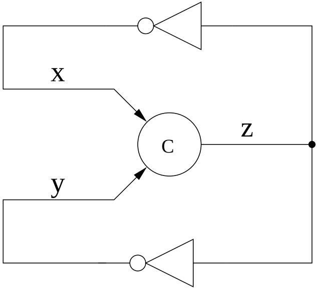
Excitation and Quiescent States Example



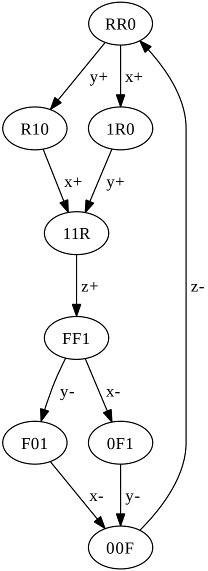


$$ES(y+) = \{\langle RR0\rangle, \langle 1R0\rangle\}$$
$$ES(y-) = \{\langle FF1\rangle, \langle 0F1\rangle\}$$
$$QS(y+) = \{\langle R10\rangle, \langle 11R\rangle\}$$
$$QS(y-) =$$

Excitation and Quiescent States Example



$$ES(y+) = \{\langle RR0\rangle, \langle 1R0\rangle\}$$
$$ES(y-) = \{\langle FF1\rangle, \langle 0F1\rangle\}$$
$$QS(y+) = \{\langle R10\rangle, \langle 11R\rangle\}$$
$$QS(y-) = \{\langle F01\rangle, \langle 00F\rangle\}$$



Excitation Regions

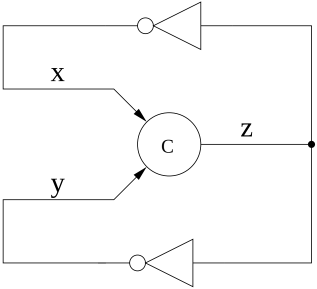
- An *excitation region* for signal u is a maximally connected subset of either $ES(u+)$ or $ES(u-)$.
- If it is a subset of $ES(u+)$, it is a *set region* (denoted $ER(u+,k)$).
- Similarly, a *reset region* is denoted $ER(u-,k)$.

Switching Regions

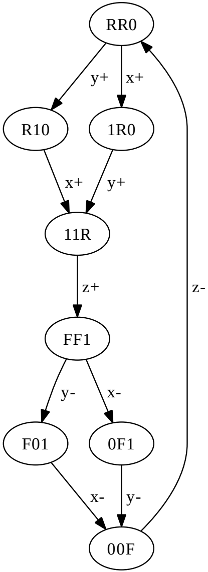
- The *switching region* for a transition $u*$, $SR(u*, k)$, is the set of states directly reachable through transition $u*$:

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

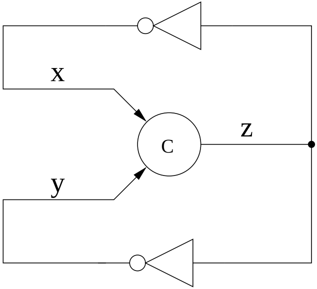
Excitation and Switching Regions Example

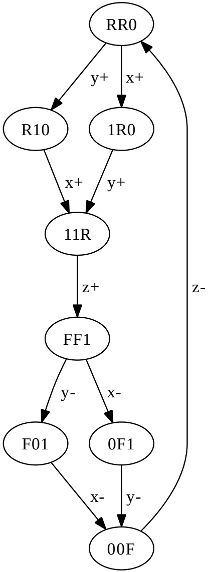


$$ER(y+,1) =$$
$$ER(y-,1) =$$
$$SR(y+,1) =$$
$$SR(y-,1) =$$



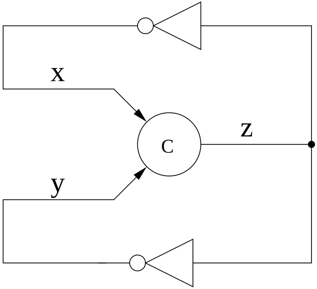
Excitation and Switching Regions Example



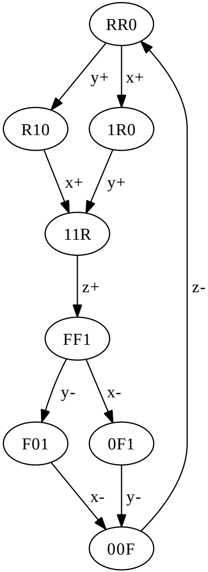


$$ER(y+,1) = \{\langle RR0\rangle, \langle 1R0\rangle\}$$
$$ER(y-,1) =$$
$$SR(y+,1) =$$
$$SR(y-,1) =$$

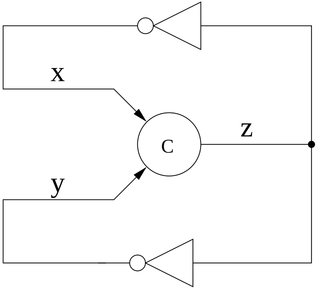
Excitation and Switching Regions Example



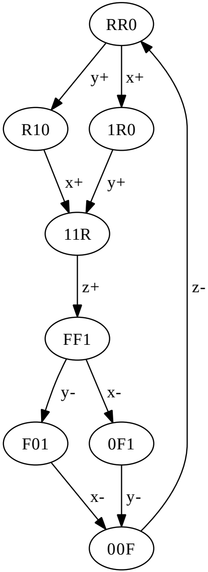
$$ER(y+,1) = \{\langle RR0\rangle, \langle 1R0\rangle\}$$
$$ER(y-,1) = \{\langle FF1\rangle, \langle 0F1\rangle\}$$
$$SR(y+,1) =$$
$$SR(y-,1) =$$

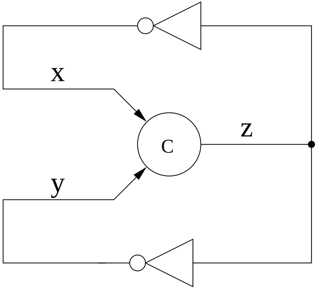


Excitation and Switching Regions Example

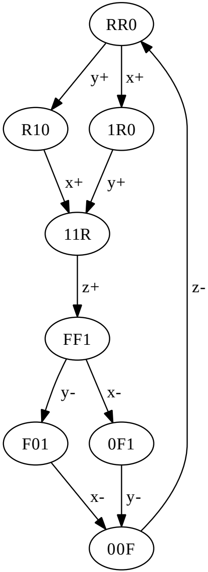


$$ER(y+,1) = \{\langle RR0\rangle, \langle 1R0\rangle\}$$
$$ER(y-,1) = \{\langle FF1\rangle, \langle 0F1\rangle\}$$
$$SR(y+,1) = \{\langle R10\rangle, \langle 11R\rangle\}$$
$$SR(y-,1) =$$





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



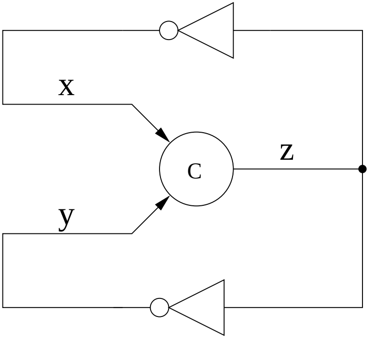
Distributive State Graphs

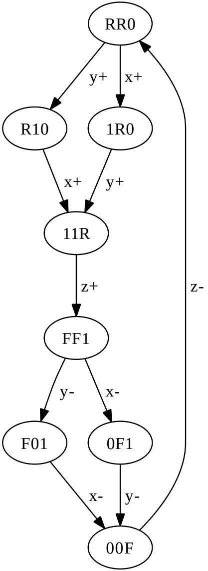
- A state graph is *distributive* if each excitation region has a unique *minimal state*.
- A minimal state for $ER(u*, k)$ is a state in $ER(u*, k)$ which cannot be directly reached by any other state in $ER(u*, k)$.
- More formally, a SG is distributive if:

$$\forall ER(u*, k) \ . \ \exists \text{exactly one } s_j \in ER(u*, k) \ .$$

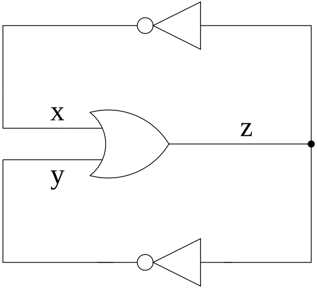
$$\neg \exists s_i \in ER(u*, k) \ . \ (s_i, t, s_j) \in \delta$$

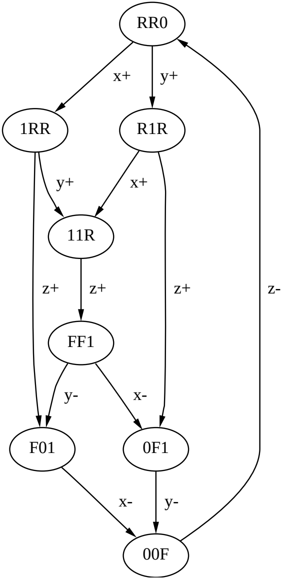
A Distributive State Graph





A Non-Distributive State Graph





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:

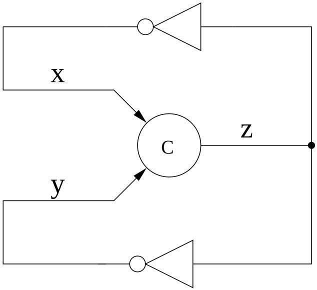
$$
\begin{aligned}
TS(u*, k) \quad = \quad & \{v \in N \mid \exists s_i, s_j \in S.((s_i, t, s_j) \in \delta) \\
& \wedge (t = v+ \vee t = v-) \\
& \wedge (s_i \notin ER(u*, k)) \wedge (s_j \in ER(u*, k))\}
\end{aligned}
$$

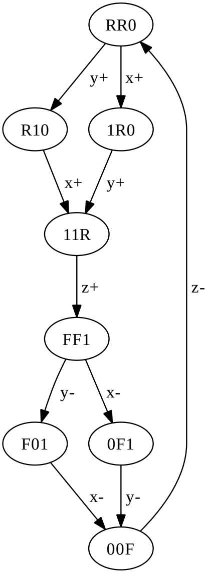
Context Signals

- 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:

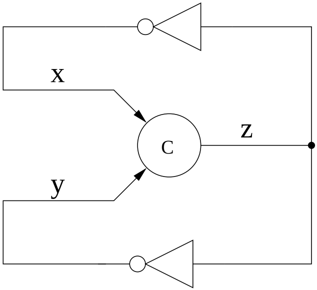
$$\begin{aligned} CS(u*, k) &= \{v_i \in N \mid v_i \notin TS(u*, k) \\ &\quad \wedge \forall s_j, s_l \in ER(u*, k) . s_j(i) = s_l(i)\} \end{aligned}$$

Trigger and Context Signals Example

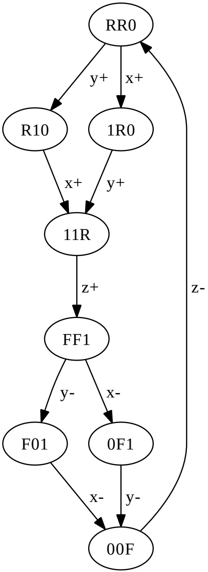




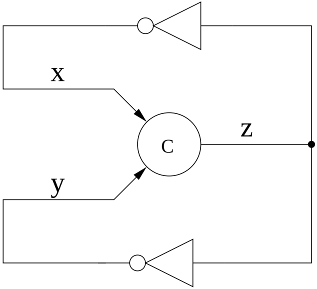
$$TS(y+,1) =$$
$$TS(y-,1) =$$
$$CS(y+,1) =$$
$$CS(y-,1) =$$



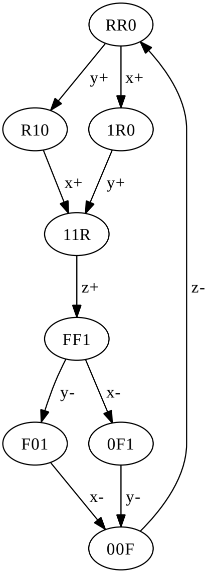
$$TS(y+,1) = \{z\}$$
$$TS(y-,1) =$$
$$CS(y+,1) =$$
$$CS(y-,1) =$$



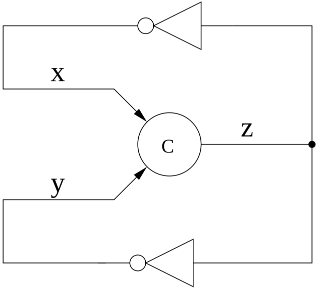
Trigger and Context Signals Example



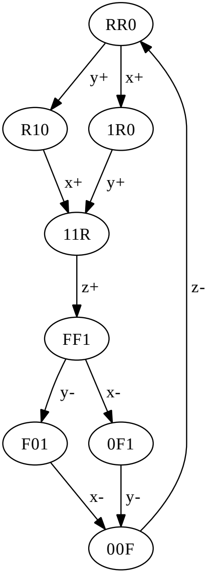
$$TS(y+,1) = \{z\}$$
$$TS(y-,1) = \{z\}$$
$$CS(y+,1) =$$
$$CS(y-,1) =$$



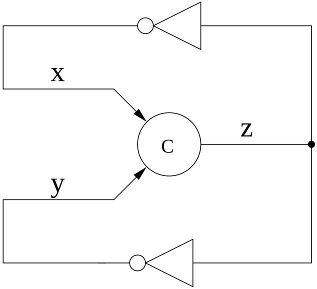
Trigger and Context Signals Example



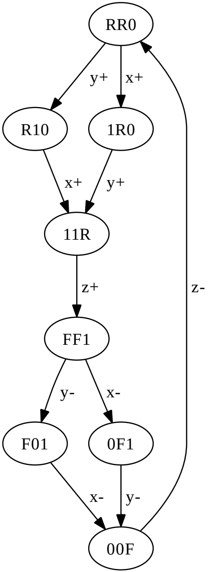
$$TS(y+,1) = \{z\}$$
$$TS(y-,1) = \{z\}$$
$$CS(y+,1) = \{y\}$$
$$CS(y-,1) =$$



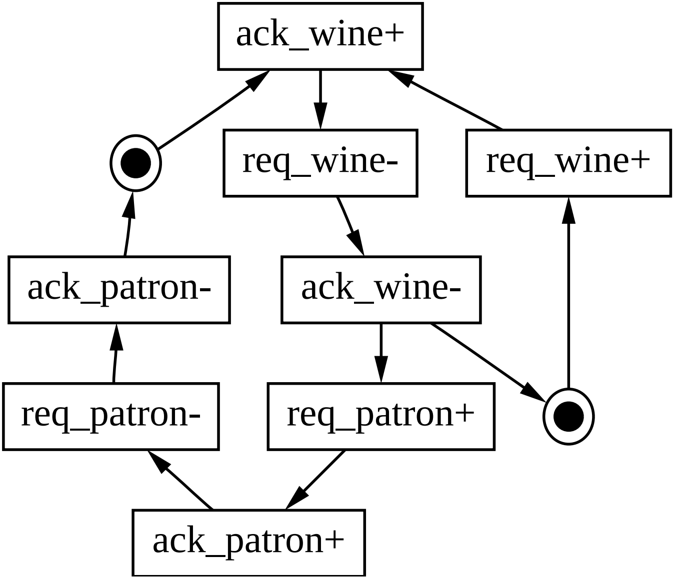
Trigger and Context Signals Example



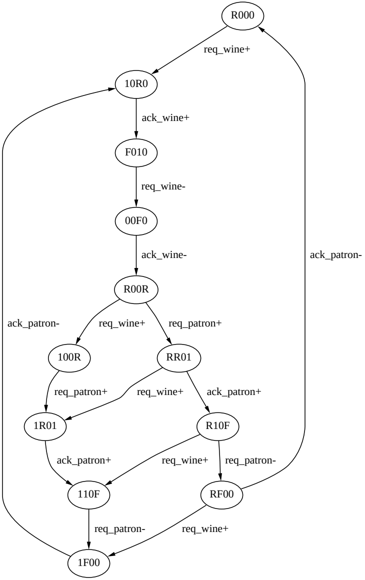
$$TS(y+,1) = \{z\}$$
$$TS(y-,1) = \{z\}$$
$$CS(y+,1) = \{y\}$$
$$CS(y-,1) = \{y\}$$



The Passive/Active Wine Shop: Petri-net



The Passive/Active Wine Shop: State Graph

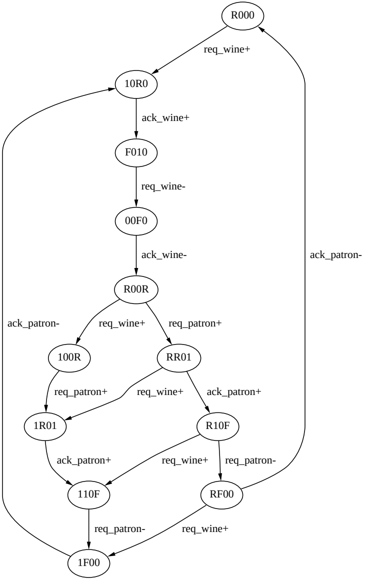


$$ES(req\_patron+) =$$
$$ES(req\_patron-) =$$
$$QS(req\_patron+) =$$
$$QS(req\_patron-) =$$

$$ER(req\_patron+, 1) =$$
$$ER(req\_patron-, 1) =$$
$$SR(req\_patron+, 1) =$$
$$SR(req\_patron-, 1) =$$
$$TS(req\_patron+, 1) =$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$

$$CS(\text{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph

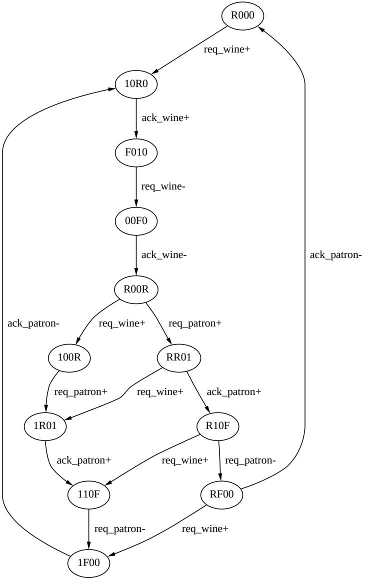


$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) =$$
$$QS(req\_patron+) =$$
$$QS(req\_patron-) =$$

$$ER(req\_patron+, 1) =$$
$$ER(req\_patron-, 1) =$$
$$SR(req\_patron+, 1) =$$
$$SR(req\_patron-, 1) =$$
$$TS(req\_patron+, 1) =$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$

$$CS(\mathrm{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph



$$ES(req\_patron+) = \{\langle R00R \rangle, \langle 100R \rangle\}$$

$$ES(req\_patron-) = \{\langle R10F \rangle, \langle 110F \rangle\}$$

$$QS(req\_patron+) =$$

$$QS(req\_patron-) =$$

$$ER(req\_patron+, 1) =$$

$$ER(req\_patron-, 1) =$$

$$SR(req\_patron+, 1) =$$

$$SR(req\_patron-, 1) =$$

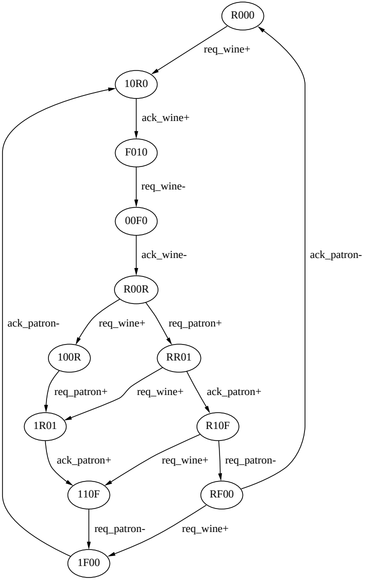
$$TS(req\_patron+, 1) =$$

$$TS(req\_patron-, 1) =$$

$$CS(req\_patron+, 1) =$$

$$CS(\mathrm{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph

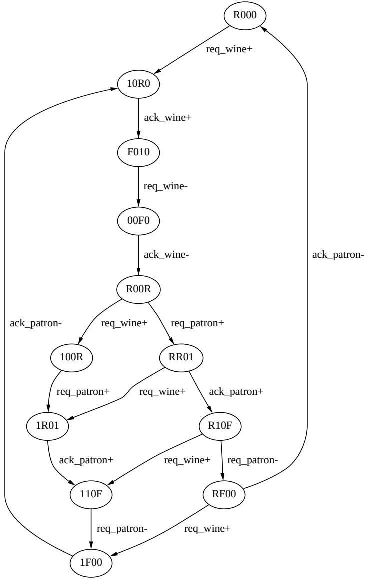


$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) =$$

$$ER(req\_patron+, 1) =$$
$$ER(req\_patron-, 1) =$$
$$SR(req\_patron+, 1) =$$
$$SR(req\_patron-, 1) =$$
$$TS(req\_patron+, 1) =$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$

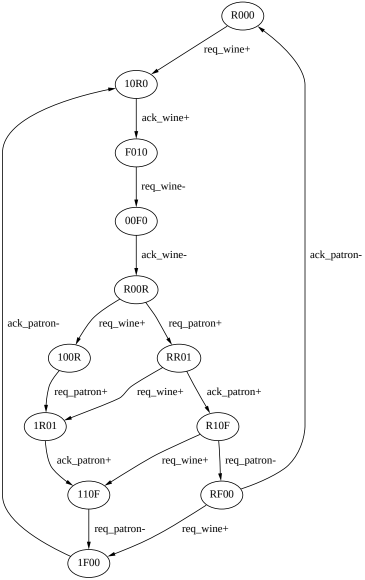
$$CS(\text{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph



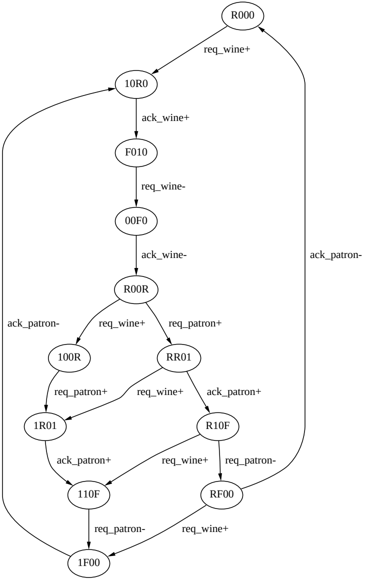
$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) =$$
$$ER(req\_patron-, 1) =$$
$$SR(req\_patron+, 1) =$$
$$SR(req\_patron-, 1) =$$
$$TS(req\_patron+, 1) =$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$
$$CS(\mathrm{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph



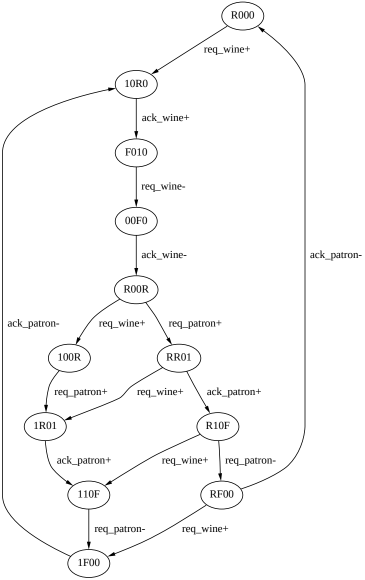
$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ER(req\_patron-, 1) =$$
$$SR(req\_patron+, 1) =$$
$$SR(req\_patron-, 1) =$$
$$TS(req\_patron+, 1) =$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$
$$CS(\mathrm{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph



$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ER(req\_patron-, 1) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$SR(req\_patron+, 1) =$$
$$SR(req\_patron-, 1) =$$
$$TS(req\_patron+, 1) =$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$
$$CS(\mathrm{req\_patron}-, 1) =$$

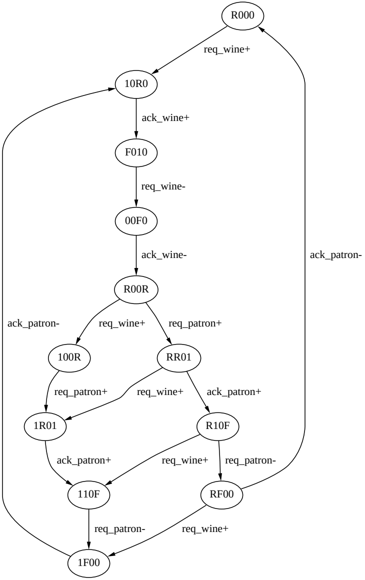
The Passive/Active Wine Shop: State Graph



$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ER(req\_patron-, 1) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$SR(req\_patron+, 1) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$SR(req\_patron-, 1) =$$
$$TS(req\_patron+, 1) =$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$

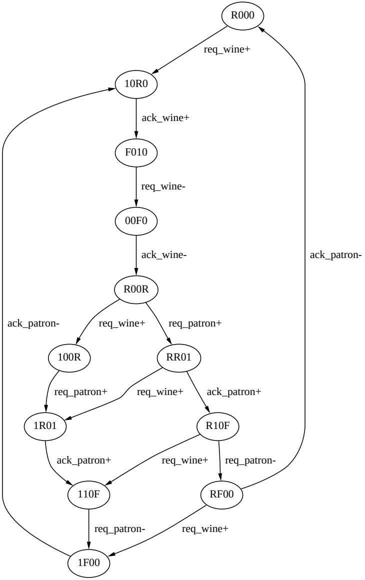
$$CS(\text{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph



$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ER(req\_patron-, 1) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$SR(req\_patron+, 1) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$SR(req\_patron-, 1) = \{\langle RF00\rangle, \langle 1F00\rangle\}$$
$$TS(req\_patron+, 1) =$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$

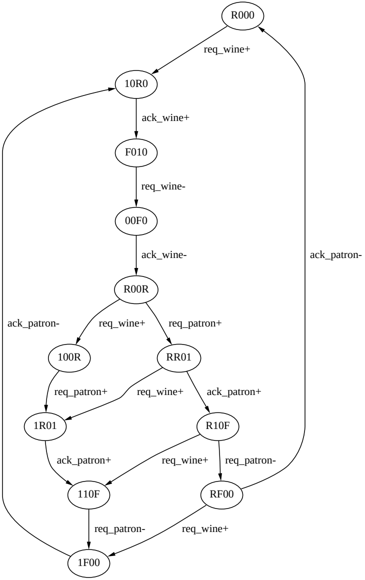
$$CS(\mathrm{req\_patron}-, 1) =$$



$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ER(req\_patron-, 1) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$SR(req\_patron+, 1) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$SR(req\_patron-, 1) = \{\langle RF00\rangle, \langle 1F00\rangle\}$$
$$TS(req\_patron+, 1) = \{ack\_wine\}$$
$$TS(req\_patron-, 1) =$$
$$CS(req\_patron+, 1) =$$

$$CS(\mathrm{req\_patron}-, 1) =$$

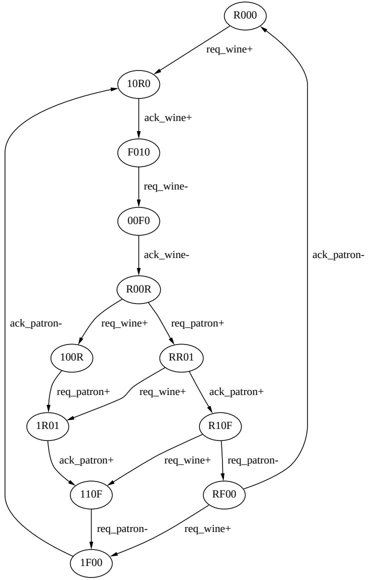
The Passive/Active Wine Shop: State Graph



$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ER(req\_patron-, 1) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$SR(req\_patron+, 1) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$SR(req\_patron-, 1) = \{\langle RF00\rangle, \langle 1F00\rangle\}$$
$$TS(req\_patron+, 1) = \{ack\_wine\}$$
$$TS(req\_patron-, 1) = \{ack\_patron\}$$
$$CS(req\_patron+, 1) =$$

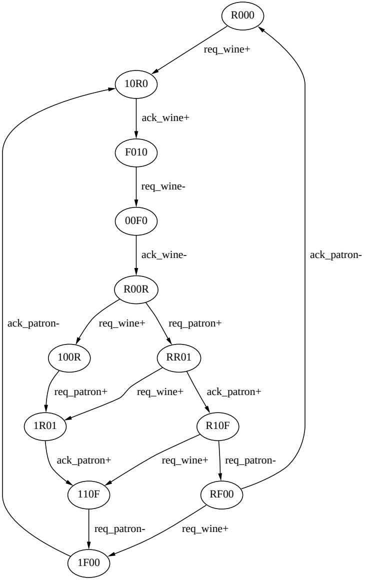
$$CS(\mathrm{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph



$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ER(req\_patron-, 1) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$SR(req\_patron+, 1) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$SR(req\_patron-, 1) = \{\langle RF00\rangle, \langle 1F00\rangle\}$$
$$TS(req\_patron+, 1) = \{ack\_wine\}$$
$$TS(req\_patron-, 1) = \{ack\_patron\}$$
$$CS(req\_patron+, 1) = \{ack\_patron, req\_patron\}$$
$$CS(\mathrm{req\_patron}-, 1) =$$

The Passive/Active Wine Shop: State Graph



$$ES(req\_patron+) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ES(req\_patron-) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$QS(req\_patron+) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$QS(req\_patron-) = \{\langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10R0\rangle, \langle F010\rangle, \langle 00F0\rangle\}$$
$$ER(req\_patron+, 1) = \{\langle R00R\rangle, \langle 100R\rangle\}$$
$$ER(req\_patron-, 1) = \{\langle R10F\rangle, \langle 110F\rangle\}$$
$$SR(req\_patron+, 1) = \{\langle RR01\rangle, \langle 1R01\rangle\}$$
$$SR(req\_patron-, 1) = \{\langle RF00\rangle, \langle 1F00\rangle\}$$
$$TS(req\_patron+, 1) = \{ack\_wine\}$$
$$TS(req\_patron-, 1) = \{ack\_patron\}$$
$$CS(req\_patron+, 1) = \{ack\_patron, req\_patron\}$$
$$CS(\mathrm{req\_patron}-, 1) = \{ack\_wine, req\_patron\}$$

Unique State Codes (USC)

- Two states have *unique state codes* (USC) if they are labeled with different binary vectors.

$$USC(s_i, s_j) \quad \Leftrightarrow \quad \lambda_S(s_i) \neq \lambda_S(s_j)$$

- A SG has USC if all states pairs have USC.

$$USC(S) \quad \Leftrightarrow \quad \forall (s_i, s_j) \in S \times S \;.\; USC(s_i, s_j)$$

Complete State Codes (CSC)

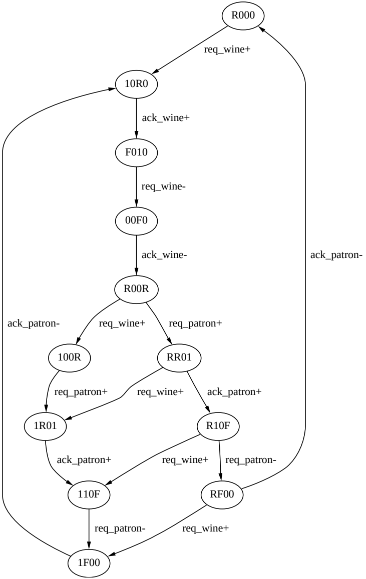
- Two states have *complete state codes* (CSC) if they either have USC or if they do not have USC but do have the same output signals excited in each state.

$$
\begin{array}{rcl}
CSC(s_i, s_j) & \Leftrightarrow & USC(s_i, s_j) \lor X(s_i) \cap O = X(s_j) \cap O \\
CSC(S) & \Leftrightarrow & \forall (s_i, s_j) \in S \times S \,.\, CSC(s_i, s_j)
\end{array}
$$

- A set of state pairs which violate CSC is defined as:

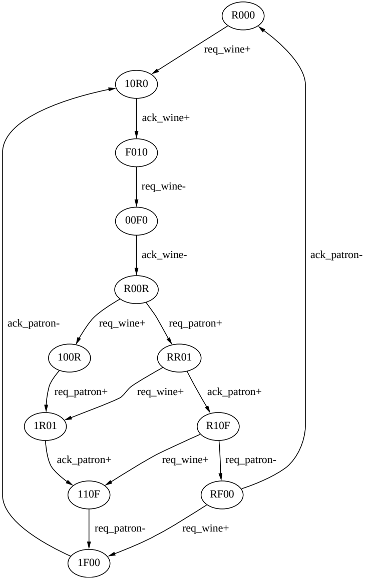
$$
CSCV(S) = \{(s_i, s_j) \in S \times S \mid \neg\, CSC(s_i, s_j)\}
$$

The Passive/Active Wine Shop: State Graph



$$CSCV \quad =$$

The Passive/Active Wine Shop: State Graph



$$CSCV = \{(\langle R000\rangle, \langle R00R\rangle), (\langle 10R0\rangle, \langle 100R\rangle)\}$$

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

Insertion Points

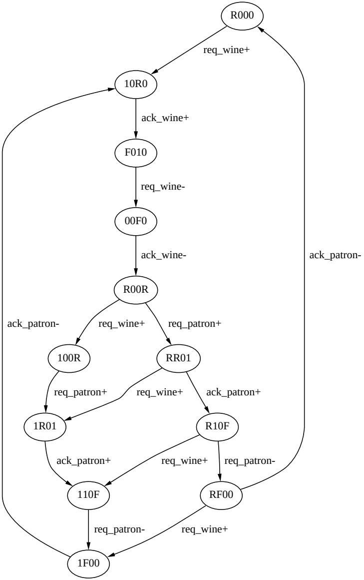
- Need to insert a rising and falling transition for new signal.
- A transition point is $TP = (t_s, t_e)$, where t_s is a set of *start transitions* and t_e is a set of *end transitions*.
- The transition point represents the location in the protocol in which a transition on a new state signal is to 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.

Transitioning States

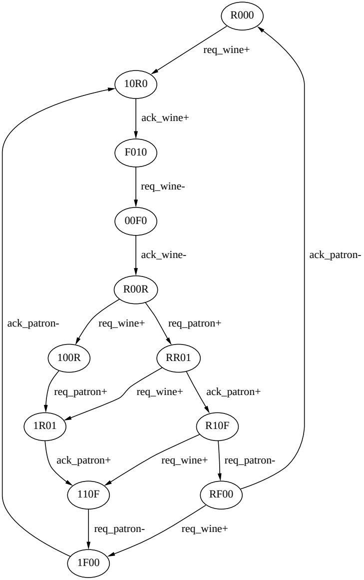
- It is necessary to determine in which states a transition can occur when inserted into a *TP*.
- The transtion on the new state signal becomes excited when the circuit enters $\cap_{t \in t_s} SR(t)$.
- Once this transition becomes excited it may remain excited in any subsequent states until there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

$$\begin{aligned} S(TP) &= \{s_j \in S \mid s_j \in \cap_{t \in t_s} SR(t) \vee \\ &\quad (\exists (s_i, t, s_j) \in \delta \,.\, s_i \in S(TP) \wedge t \notin t_e)\} \end{aligned}$$

$TP = (\{req\_patron+\}, \{req\_patron-\})$

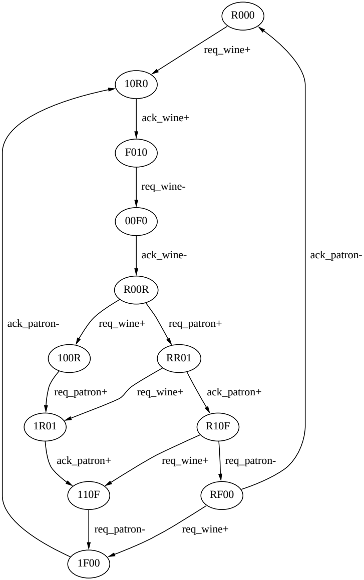


$TP = (\{req\_patron+\}, \{req\_patron-\})$



$\{\langle RR01\rangle, \langle 1R01\rangle,$

$TP = (\{req\_patron+\}, \{req\_patron-\})$



$$\{\langle RR01\rangle, \langle 1R01\rangle, \langle R10F\rangle, \langle 110F\rangle\}$$

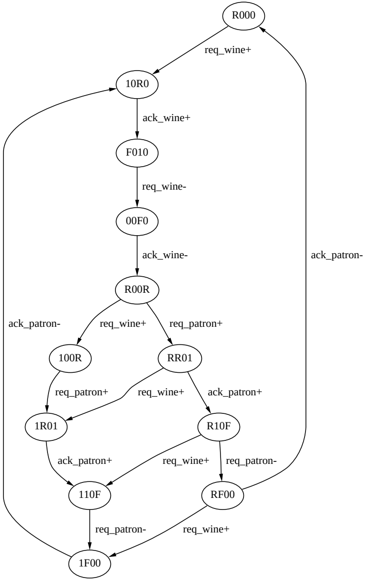
Insertion Point Explosion

- 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 $2^{|T|^4}$.
- 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:
  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 $\forall t_1, t_2 \in t_e \,.\; t_1 \parallel t_2$).

$TP = (\{ack\_wine+\}, \{ack\_wine-\})$



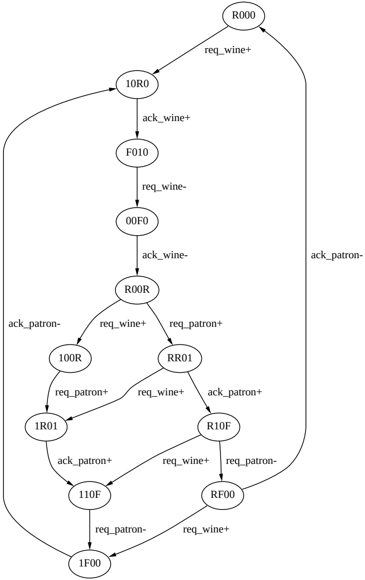
$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \,.\, t \notin T_I$
3. $\forall t_1, t_2 \in t_s \,.\, t_1 \parallel t_2$

   $\forall t_1, t_2 \in t_e \,.\, t_1 \parallel t_2$

$TP = (\{req\_patron+\}, \{ack\_patron+\})$



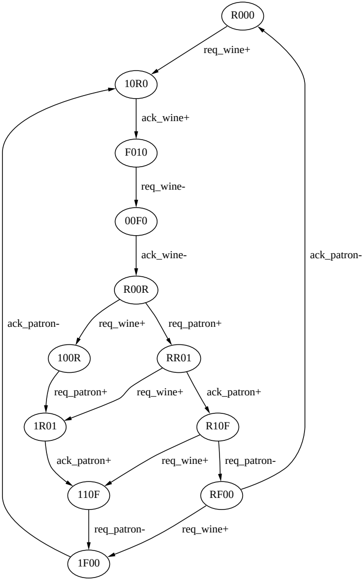
$\langle \text{req\_wine}, \text{ack\_patron}, \text{ack\_wine}, \text{req\_patron} \rangle$

$T_I = \{\text{req\_wine}, \text{ack\_patron}\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \ . \ t \notin T_I$
3. $\forall t_1, t_2 \in t_s \ . \ t_1 \parallel t_2$

   $\forall t_1, t_2 \in t_e \ . \ t_1 \parallel t_2$

$TP = (\{ack\_wine-\}, \{ack\_wine+\})$



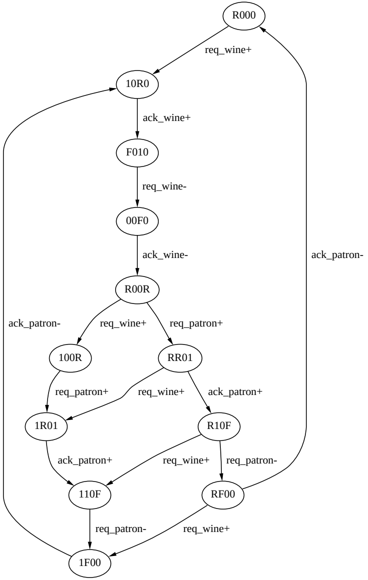
$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \;.\; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \;.\; t_1 \parallel t_2$

   $\forall t_1, t_2 \in t_e \;.\; t_1 \parallel t_2$

$TP = (\{req\_wine-\}, \{ack\_wine-\})$

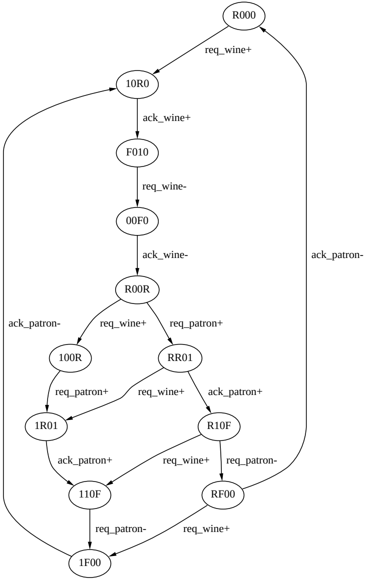


$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \; . \; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \; . \; t_1 \parallel t_2$
   $\forall t_1, t_2 \in t_e \; . \; t_1 \parallel t_2$

$TP = (\{req\_patron+\}, \{req\_patron-\})$

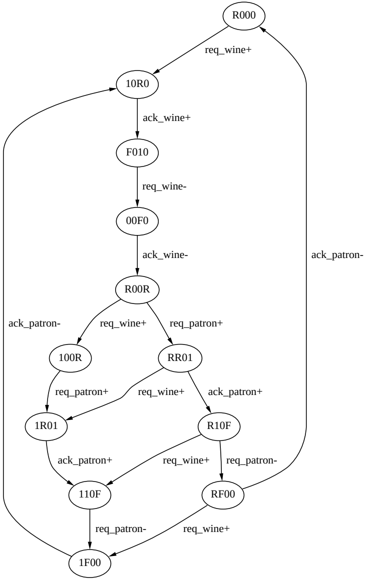


$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \;.\; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \;.\; t_1 \parallel t_2$
   $\forall t_1, t_2 \in t_e \;.\; t_1 \parallel t_2$

$TP = (\{ack\_patron+\}, \{req\_patron-\})$



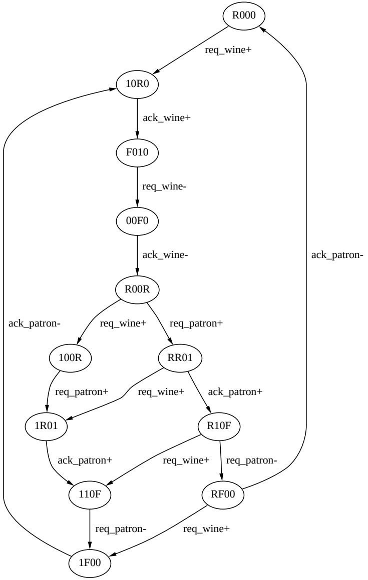
$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \; . \; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \; . \; t_1 \parallel t_2$

   $\forall t_1, t_2 \in t_e \; . \; t_1 \parallel t_2$

$TP = (\{ack\_wine-, req\_patron+\}, \{req\_patron-\})$

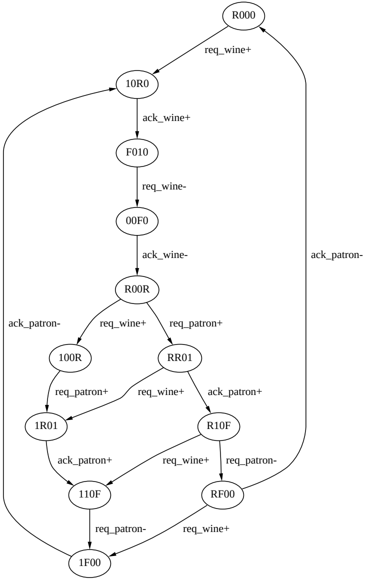


$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \;.\; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \;.\; t_1 \parallel t_2$
   $\forall t_1, t_2 \in t_e \;.\; t_1 \parallel t_2$

$TP = (\{req\_wine+, req\_patron-\}, \{ack\_wine+\})$

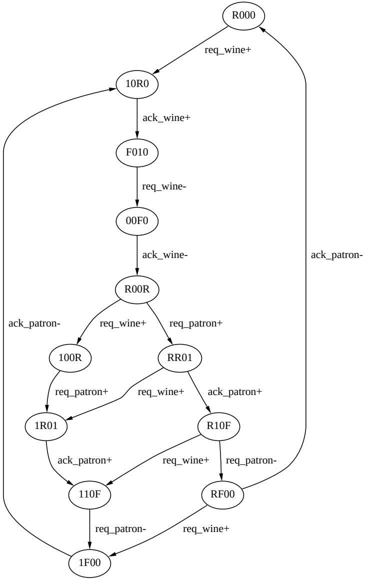


$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{\text{req\_wine}, \text{ack\_patron}\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \; . \; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \; . \; t_1 \parallel t_2$
   $\forall t_1, t_2 \in t_e \; . \; t_1 \parallel t_2$

$TP = (\{req\_wine+, req\_patron-\}, \{ack\_wine-\})$

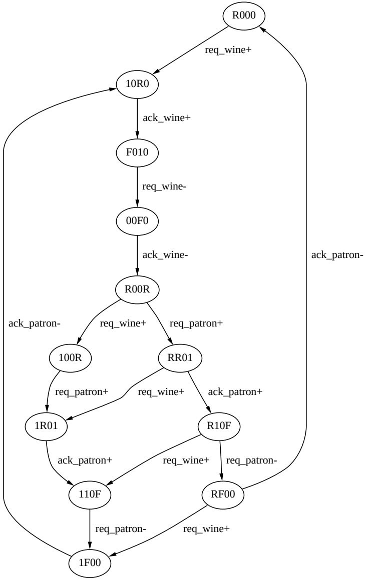


$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{\text{req\_wine}, \text{ack\_patron}\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \,.\, t \notin T_I$
3. $\forall t_1, t_2 \in t_s \,.\, t_1 \parallel t_2$
   $\forall t_1, t_2 \in t_e \,.\, t_1 \parallel t_2$

$TP = (\{req\_patron+\}, \{ack\_wine+, req\_patron-\})$

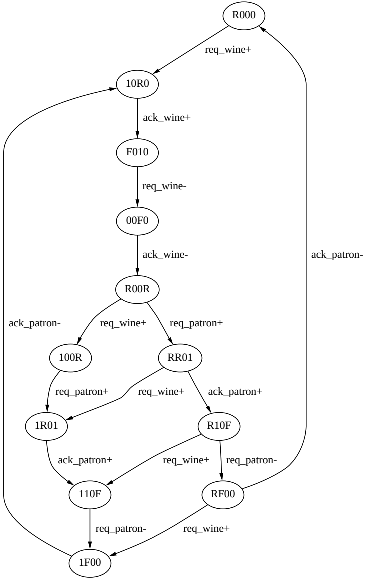


$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{\text{req\_wine}, \text{ack\_patron}\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \;.\; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \;.\; t_1 \parallel t_2$
   $\forall t_1, t_2 \in t_e \;.\; t_1 \parallel t_2$

$TP = (\{req\_wine+, ack\_patron-\}, \{ack\_wine+\})$

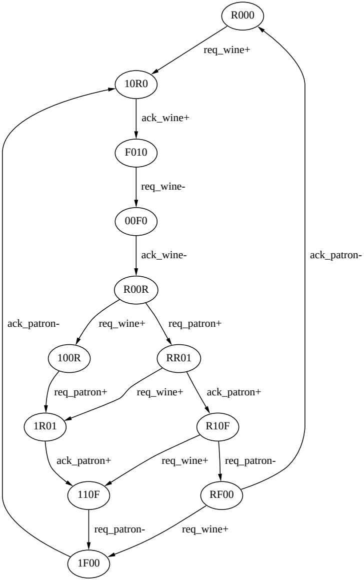


$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \; . \; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \; . \; t_1 \parallel t_2$
   $\forall t_1, t_2 \in t_e \; . \; t_1 \parallel t_2$

$TP = (\{req\_patron+\}, \{req\_patron+\})$



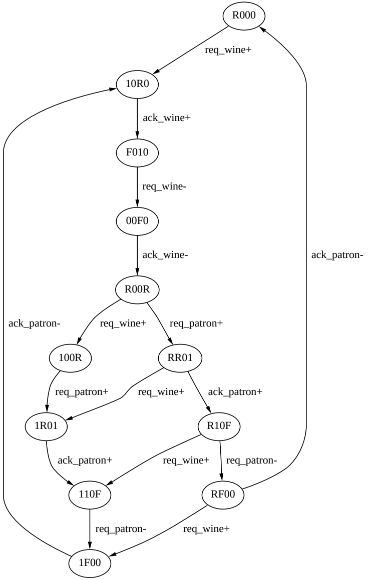
$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \;.\; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \;.\; t_1 \parallel t_2$

   $\forall t_1, t_2 \in t_e \;.\; t_1 \parallel t_2$

$TP = (\{req\_wine+, ack\_patron-\}, \{ack\_wine-\})$



$\langle$req_wine, ack_patron, ack_wine, req_patron$\rangle$

$T_I = \{$req_wine, ack_patron$\}$

1. $t_s \cap t_e = \emptyset$
2. $\forall t \in t_e \;.\; t \notin T_I$
3. $\forall t_1, t_2 \in t_s \;.\; t_1 \parallel t_2$

   $\forall t_1, t_2 \in t_e \;.\; t_1 \parallel t_2$

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:

$$TP_R(t_s) \cap TP_F(t_s) \neq \emptyset$$
$$TP_R(t_e) \cap TP_F(t_e) \neq \emptyset$$

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

$$\begin{aligned} IP \quad = \quad & (\{ack\_wine+\}, \{ack\_wine-\}), \\ & (\{req\_wine+, req\_patron-\}, \{ack\_wine-\}) \end{aligned}$$

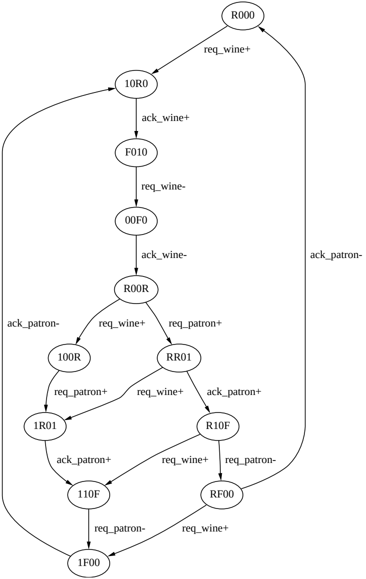
State Graph Coloring

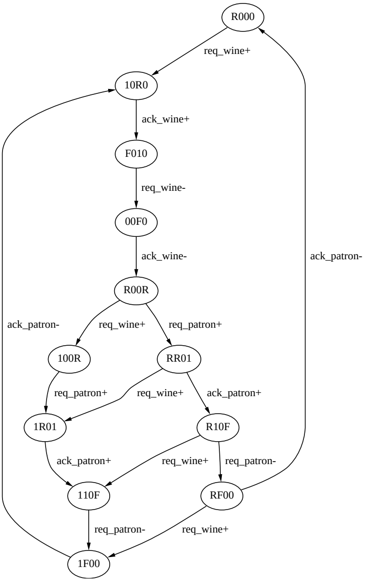
- Need to determine effect of inserting a state variable in an IP.
- Can be done by inserting the state signal and finding new SG.
- This approach is unnecessarily time consuming and may produce a SG with an inconsistent state assignment.
- Instead, SG is partitioned into four parts corresponding to the *rising*, *falling*, *high*, and *low* sets for the new state signal.

State Graph Coloring Procedure

- States in $S(TP_R)$ 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.

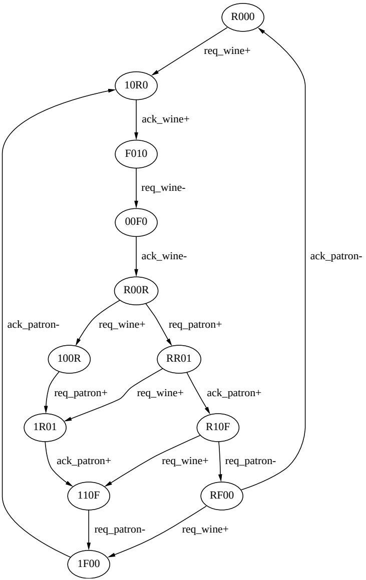
IP(({$req\_patron+$}, {$req\_patron-$}), ({ack_wine-}, {ack_wine+}))





$$\text{Rising} = \{\langle RR01\rangle, \langle 1R01\rangle, \langle R10F\rangle, \langle 110F\rangle\}$$

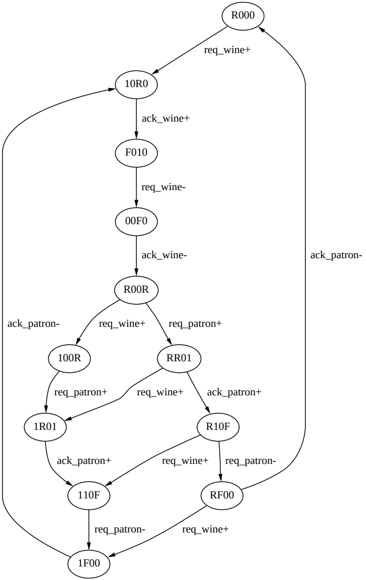
IP(({$req\_patron+$}, {$req\_patron-$}), ({ack_wine-}, {ack_wine+}))



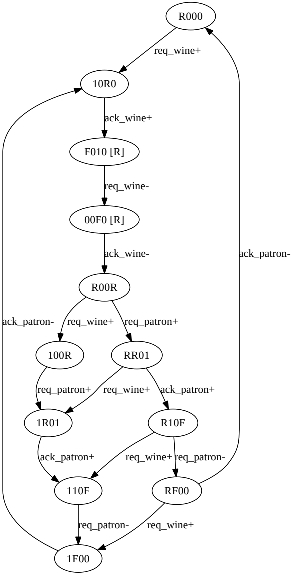
$$\text{Rising} = \{\langle RR01\rangle, \langle 1R01\rangle, \langle R10F\rangle, \langle 110F\rangle\}$$

$$\text{Falling} = \{\langle R00R\rangle, \langle 100R\rangle, \langle RR01\rangle, \langle 1R01\rangle, \langle R10F\rangle, \langle 110F\rangle, \langle RF00\rangle, \langle 1F00\rangle, \langle R000\rangle, \langle 10F0\rangle\}$$

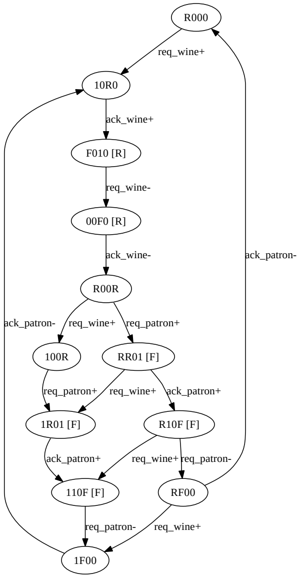
IP(({ack_wine+}, {ack_wine-}), ({$req\_patron+$}, {$req\_patron-$}))



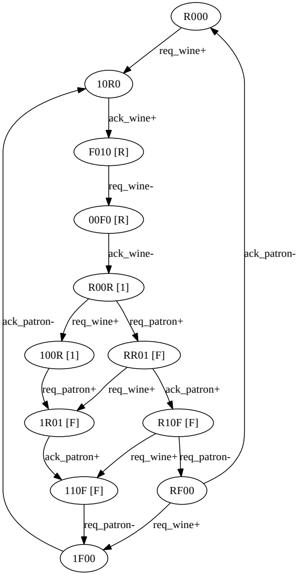
IP(({ack_wine+}, {ack_wine-}), ({$req\_patron+$}, {$req\_patron-$}))



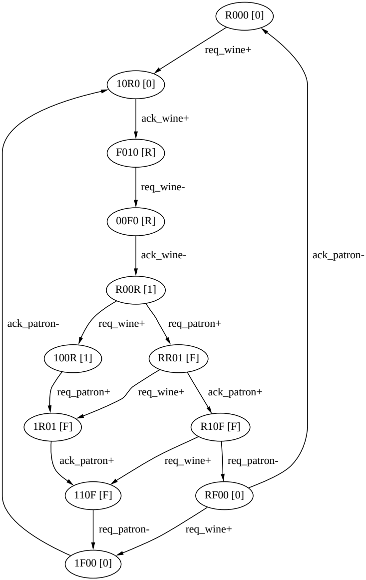
IP(({*ack_wine*+}, {*ack_wine*−}), ({*req_patron*+}, {*req_patron*−}))



IP(({*ack_wine*+}, {*ack_wine*−}), ({*req_patron*+}, {*req_patron*−}))



$\mathrm{IP}((\{ack\_wine+\}, \{ack\_wine-\}), (\{req\_patron+\}, \{req\_patron-\}))$



Insertion Point Primary Cost Function

- 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.

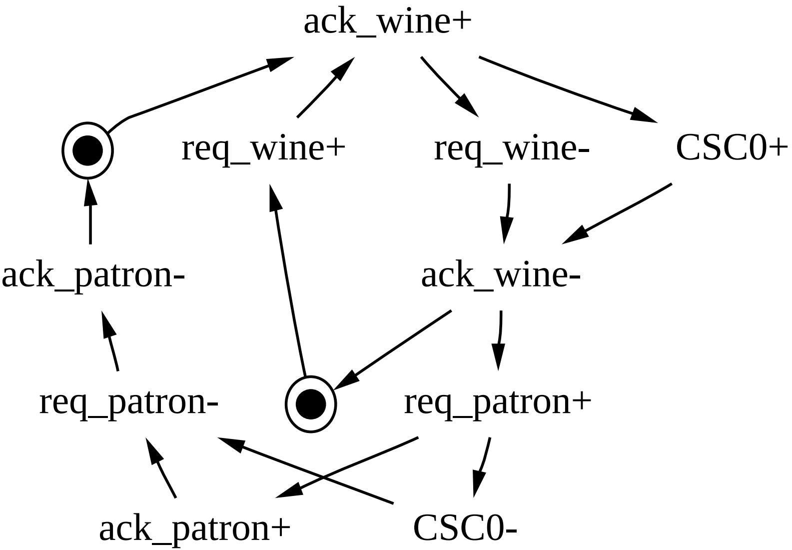
Insertion Point Secondary Cost Functions

- The IP with the smallest sum $|TP_R(t_e)| + |TP_F(t_e)|$.
- The IP with the smallest sum $|TP_R(t_s)| + |TP_F(t_s)|$.

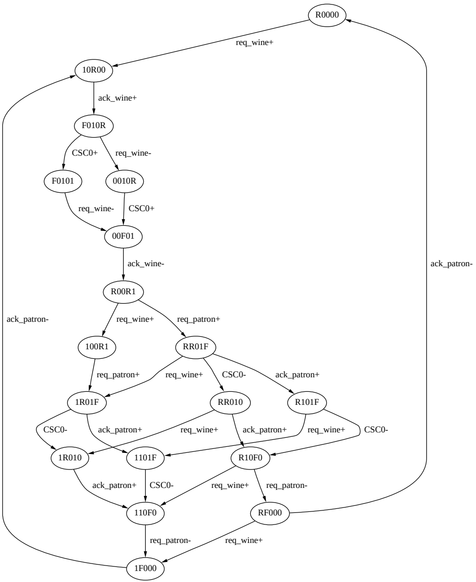
State Signal Insertion: Petri-net

- State signal can be inserted into a Petri-net by adding arcs from each transition in t_s to the new state signal transition.
- Arcs are also 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.

IP(({ack_wine+}, {ack_wine-}), ({$req\_patron+$}, {$req\_patron-$}))



IP(({ack_wine+}, {ack_wine-}), ({$req\_patron+$}, {$req\_patron-$}))



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’.

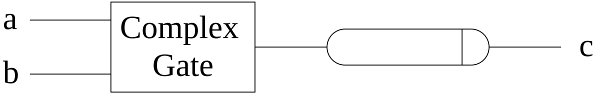
```
csc_solver(SG)
  CSCV = find_csc_violations(SG);
  if (|CSCV| = 0) return SG;
  best = |CSCV|;
  best_IP = (∅,∅);
  TP = find_all_transition_points(SG);
  foreach TP_R ∈ TP
    foreach TP_F ∈ TP
      if IP = (TP_R, TP_F) is legal then
        CSG = color_state_graph(SG, TP_R, TP_F);
        CSCV = find_csc_violations(CSG);
        if (CSG is consistent) and ((|CSCV| < best) or
          ((|CSCV|=best) and (cost(IP)<cost(best_IP)))) then
          best = |CSCV|;
          best_IP = (TP_R, TP_F);
  SG = insert_state_signal(SG, best_IP);
  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:
  1. Complex gates
  2. Generalized C-elements
  3. 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.



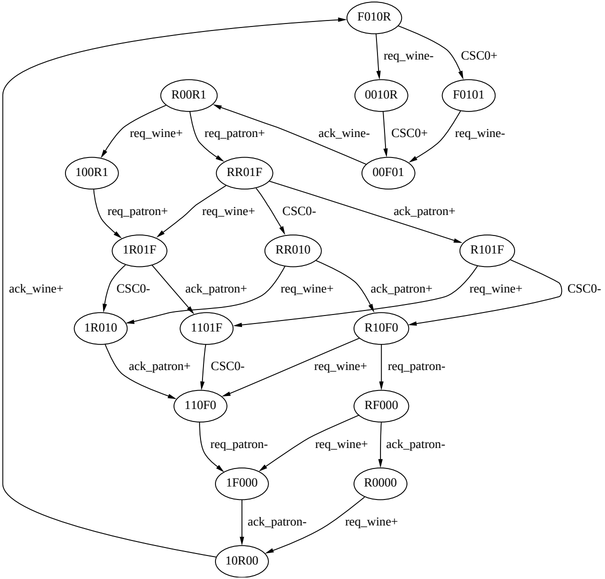
Atomic Gate Logic Synthesis

- On-set for a signal u is the set of states in which u is excited to rise or stable high.
- Off-set is 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.

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

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

Passive/Active Wine Shop: Atomic Gate



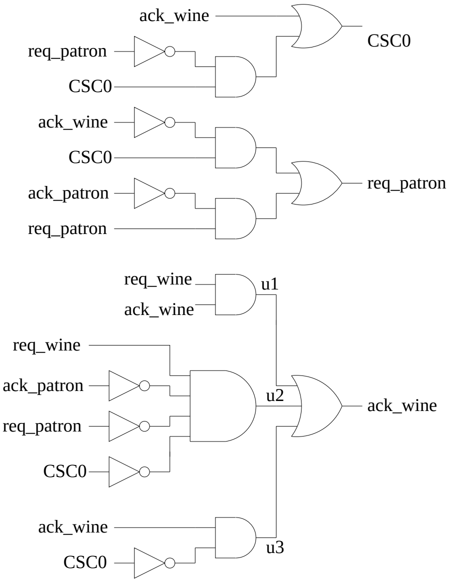
Atomic Gate: Example (Ack_Wine)

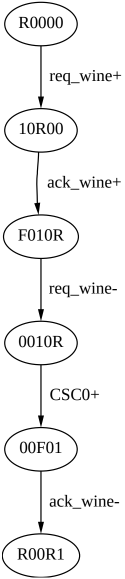
$$
\begin{aligned}
\textit{ON-set} &= \{10000, 10100, 00100, 10101\} \\
\textit{OFF-set} &= \{00101, 00001, 10001, 00011, 10011, 01011, 00010, \\
&\quad 10010, 01010, 11010, 01000, 11000, 11011, 00000\} \\
\textit{DC-set} &= \{00110, 00111, 01001, 01100, 01101, 01110, 01111, \\
&\quad 10110, 10111, 11001,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

$$
P = \{\texttt{1-1--}, \texttt{-11--}, \texttt{--11-}, \texttt{--1-0}, \texttt{-1-01}, \texttt{10-00}\}
$$

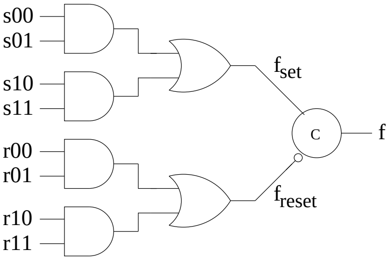
	1-1--	-11--	--11-	--1-0	-1-01	10-00
10000	–	–	–	–	–	1
10100	1	–	–	1	–	1
00100	–	–	–	1	–	–
10101	1	–	–	–	–	–

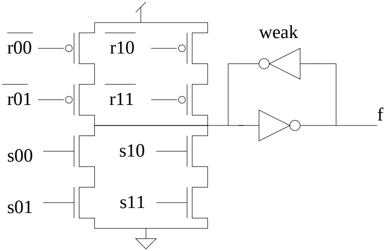
Passive/Active Shop: Atomic Gate Circuit

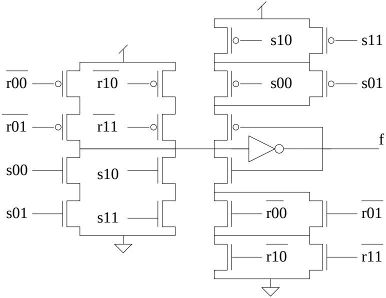




Generalized C-Elements







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

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

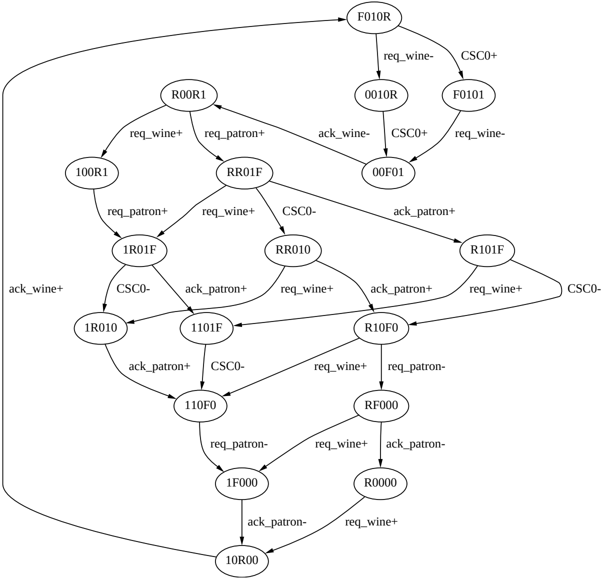
gC Logic Synthesis

- For *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.

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

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

Passive/Active Wine Shop: gC

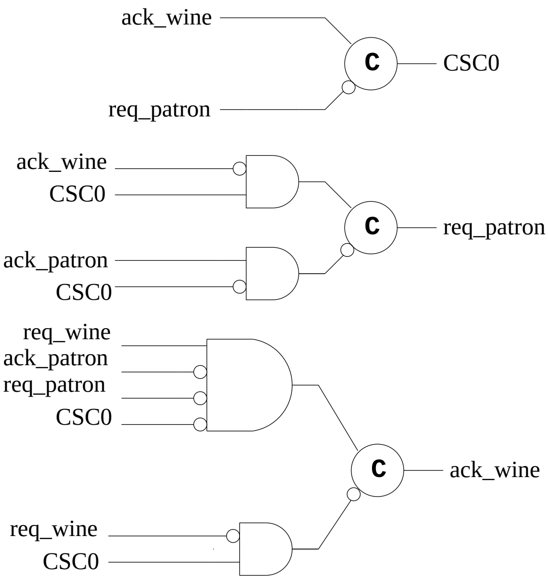


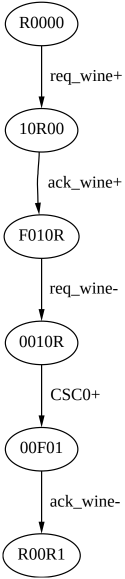
gC Circuit: Example

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

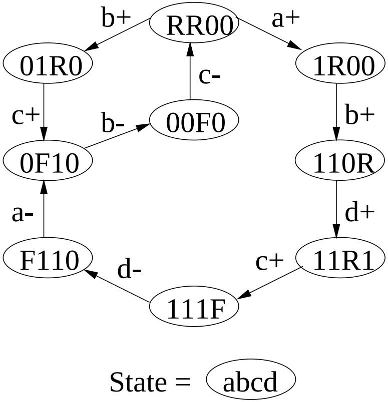
$$
P = \{\texttt{1-1--}, \texttt{-11--}, \texttt{--11-}, \texttt{--1-0}, \texttt{-1-01}, \texttt{10-00}\}
$$

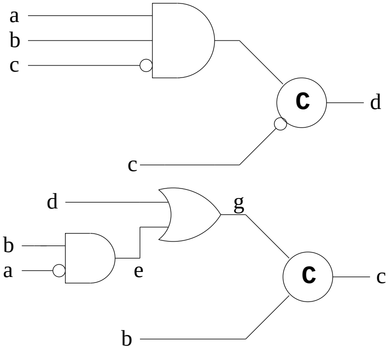
Passive/Active Shop: gC Circuit





A Simple Example

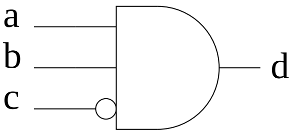




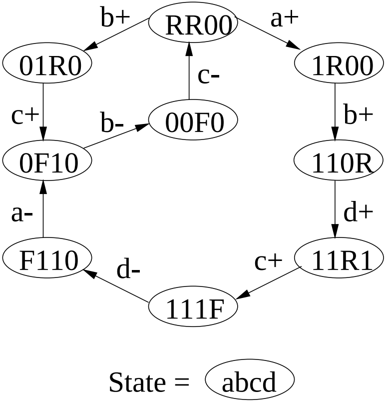
Combinational Optimization

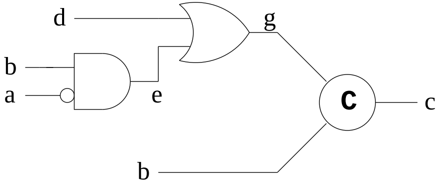
- If *set(u)* is on in all states in which *u* should be rising or high, then the state holding element can be removed.
- Implementation for *u* is equal to the logic for *set(u)*.
- If *reset(u)* is on in all states in which *u* should be falling or low, then the signal *u* can be implemented with $\overline{reset(u)}$.

Combinational Optimization



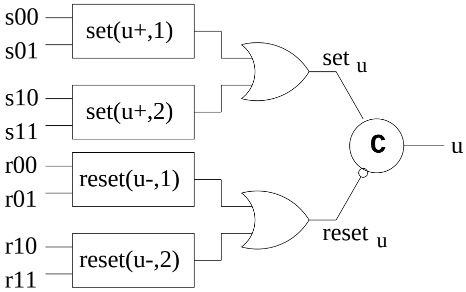
Gate-Level Hazard





$$\langle F110\rangle \rightarrow \langle 0F10\rangle \rightarrow \langle 00F0\rangle$$

Standard C-implementation



- Structure similar to gC-implemenation, 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-implemenation, an excitation region can be implemented by multiple product terms.
- A region function may need to be implemented using SOP.

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.

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.

$$ER(u*, k) \subseteq [C(u*, k) \cap S] \subseteq [ER(u*, k) \cup QS(u*)]$$

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

$$[(s_i, t, s_j) \in \delta \wedge s_i \notin C(u*, k) \wedge s_j \in C(u*, k)] \Rightarrow s_j \in ER(u*, k)$$

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.

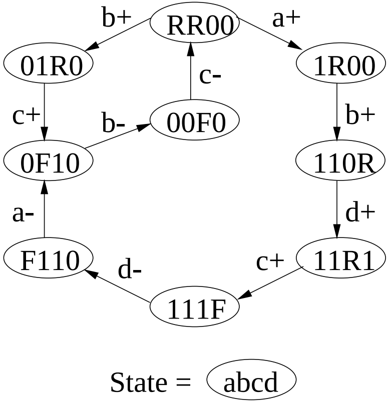
$$
\begin{aligned}
\textit{ON-set} &= \{\lambda_S(s) \mid s \in (ER(u+, k)\} \\
\textit{OFF-set} &= \{\lambda_S(s) \mid s \in (ES(u-) \cup QS(u-)) \cup \\
&\quad (ES(u+) - ER(u+, k))\} \\
\textit{DC-set} &= \{0, 1\}^{|N|} - (\textit{ON-set} \cup \textit{OFF-set})
\end{aligned}
$$

Gate Level Logic Synthesis: Reset Regions

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

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

Gate Level Circuit: Example



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 $ER(c+,1)$.

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

- The primes found are as follows:

$$
P = \{\texttt{01--}, \texttt{1-1-}, \texttt{-11-}, \texttt{0--1}, \texttt{-0-1}, \texttt{--11}\}
$$

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 $ER(u*, k)$.

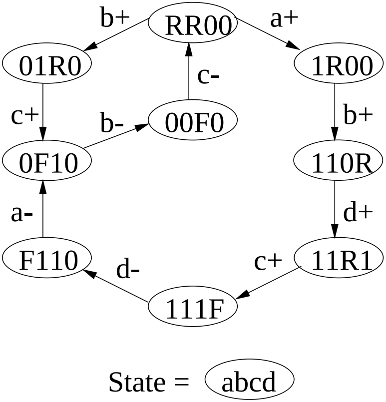
$$IS(c) = \{s_i \mid s_i \notin c \land \exists s_j.(s_i, t, s_j) \in \delta \land (s_j \in c) \land (s_j \notin ER(u*, k))\}$$

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

Existence of a Prime Cover

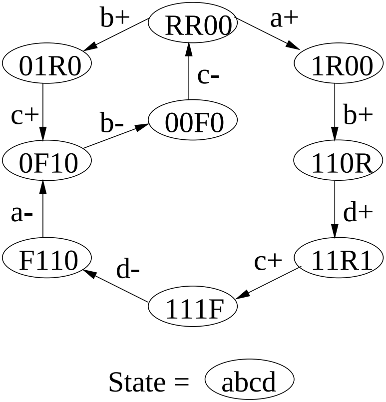
- An implicant may have implied states that are outside the excitation region and the cooresponding quiescent states.
- Implied states may not be covered by any other implicant.
- If this implicant is the only prime which covers some excitation region state, then no cover can be found using only primes.

Existence of a Prime Cover: Example



Consider prime `01--`

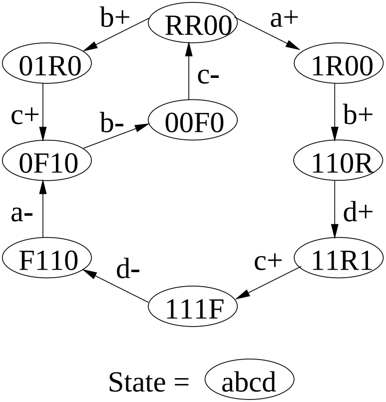
Existence of a Prime Cover: Example



Consider prime `01--`

Entered by $(F110, a-, 0F10)$

Existence of a Prime Cover: Example

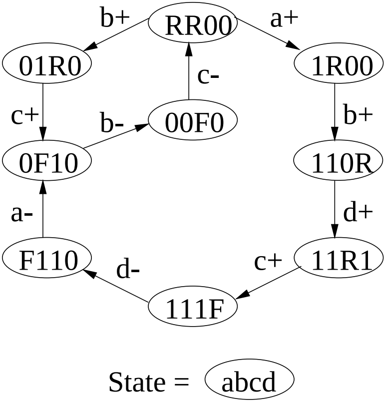


Consider prime `01--`

Entered by $(F110, a-, 0F10)$

F110 is implied state

Existence of a Prime Cover: Example



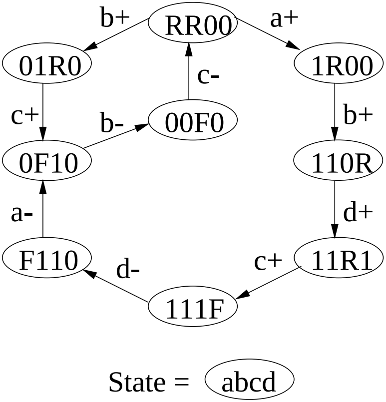
Consider prime 01- -

Entered by $(F110, a-, 0F10)$

$F110$ is implied state

Cover with 1-1- or -11-

Existence of a Prime Cover: Example



Consider prime `01--`

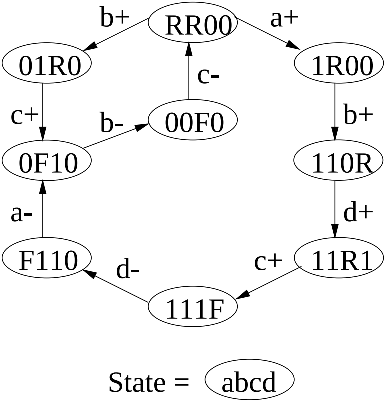
Entered by $(F110, a-, 0F10)$

$F110$ is implied state

Cover with `1-1-` or `-11-`

Entered by $(11R1, c+, 111F)$

Existence of a Prime Cover: Example



Consider prime `01--`

Entered by $(F110, a-, 0F10)$

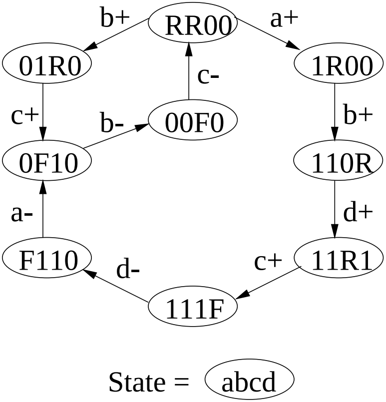
$F110$ is implied state

Cover with `1-1-` or `-11-`

Entered by $(11R1, c+, 111F)$

$11R1$ is implied state

Existence of a Prime Cover: Example



Consider prime `01--`

Entered by $(F110, a-, 0F10)$

$F110$ is implied state

Cover with `1-1-` or `-11-`

Entered by $(11R1, c+, 111F)$

$11R1$ is implied state

But it is in the OFF-set

Candidate Implicants

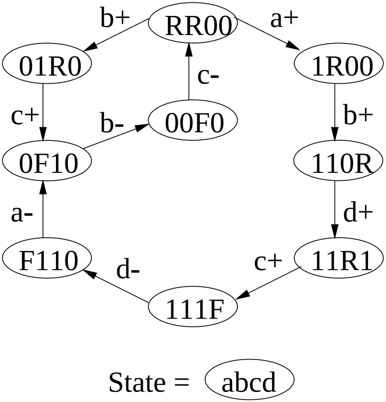
- Implicant is a *candidate implicant* if there does not exist one which properly contains it with a subset of the implied states.
- c_i is a candidate implicant if there *does not exist* an implicant c_j that satisfies the following two conditions:

$$\begin{aligned} c_j &\supset c_i \\ IS(c_j) &\subseteq IS(c_i). \end{aligned}$$

- 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.

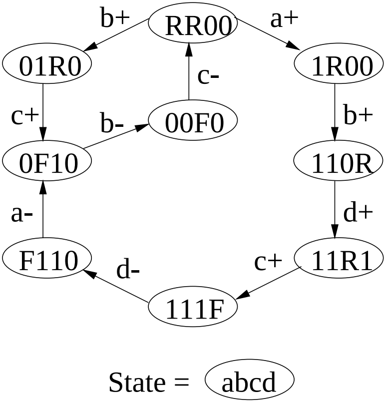
Candidate Implicant Algorithm

```
candidate_implicants(SG, P)
    done = ∅
    for(k = |largest(P)|; k ≥ 1; k − −)
        foreach(q ∈ P;  |q| = k)  enqueue(C, q)
        foreach(c ∈ C;  |c| = k)
            if(IS(SG, c) = ∅) continue
            foreach(s ∈ lit_extend(c))
                if(s ∈ done) continue
                Γ_s = IS(SG, s)
                prime = true
                foreach(q ∈ C; |q| ≥ k)
                    if (s ⊂ q)
                        Γ_q = IS(SG, q)
                        if (Γ_s ⊇ Γ_q) {
                            prime = false;
                            break
                if(prime = 1) enqueue(C, s)
                done = done ∪ {s}
```



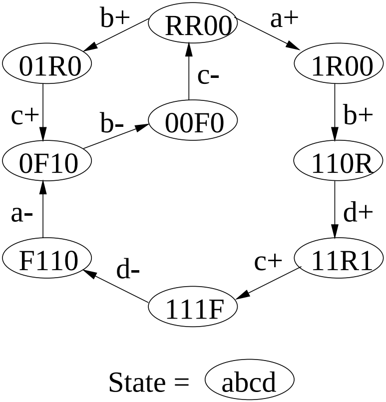
Primes	Implied States
01--	
1-1-	
-11-	
0--1	
-0-1	
--11	

Candidate Implicant Algorithm Example



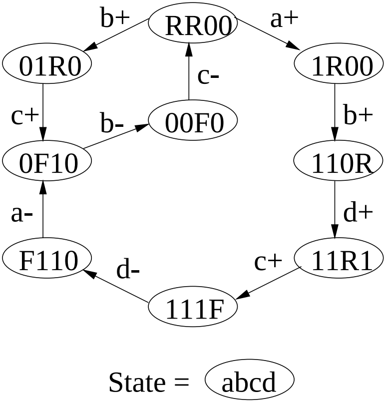
Primes	Implied States
01--	F110
1-1-	
-11-	
0--1	
-0-1	
--11	

Candidate Implicant Algorithm Example



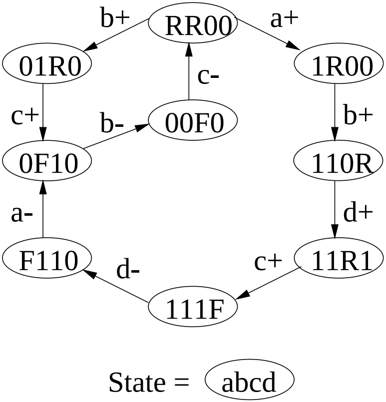
Primes	Implied States
01--	F110
1-1-	11R1
-11-	
0--1	
-0-1	
--11	

Candidate Implicant Algorithm Example



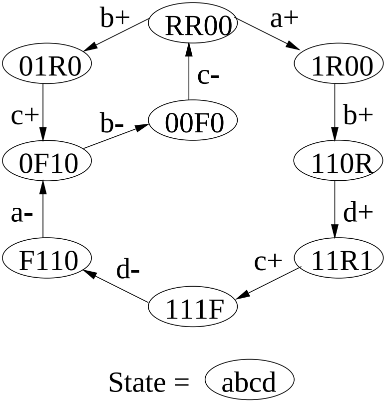
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	
-0-1	
--11	

Candidate Implicant Algorithm Example



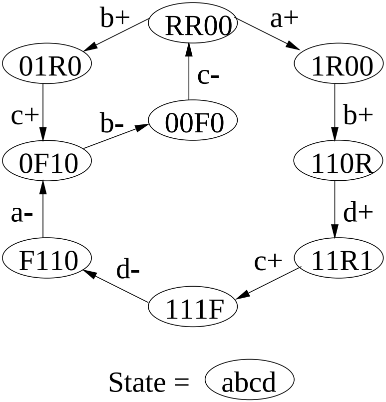
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	
--11	

Candidate Implicant Algorithm Example



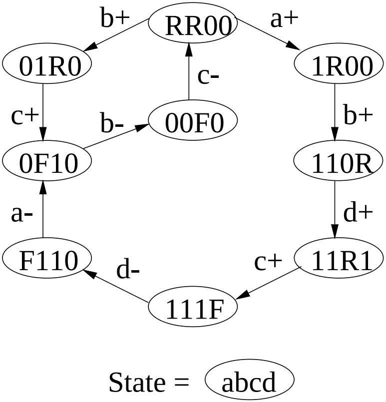
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	

Candidate Implicant Algorithm Example



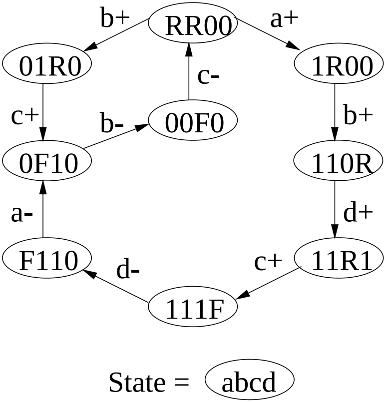
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	11R1

Candidate Implicant Algorithm Example



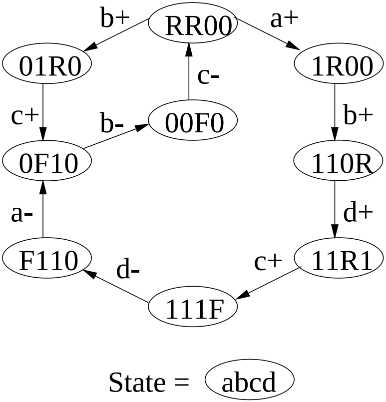
Primes	Implied States
`01--`	F110
`1-1-`	11R1
`-11-`	11R1, 01R0
`0--1`	$\emptyset$
`-0-1`	$\emptyset$
`--11`	11R1
`010-`	

Candidate Implicant Algorithm Example



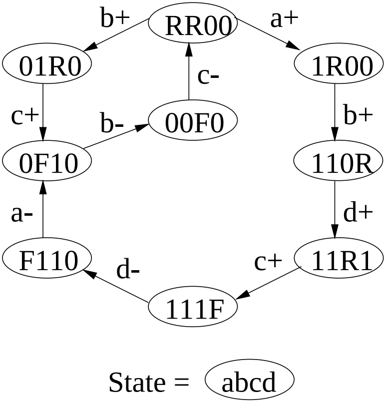
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	11R1
010-	$\emptyset$

Candidate Implicant Algorithm Example



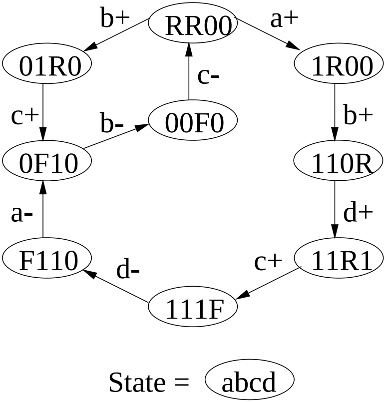
Primes	Implied States
`01--`	F110
`1-1-`	11R1
`-11-`	11R1, 01R0
`0--1`	∅
`-0-1`	∅
`--11`	11R1
`010-`	∅
`011-`	

Candidate Implicant Algorithm Example



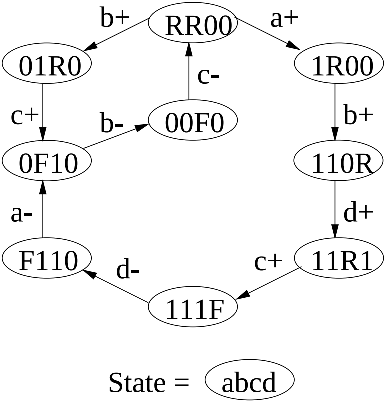
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	11R1
010-	$\emptyset$
011-	F110

Candidate Implicant Algorithm Example

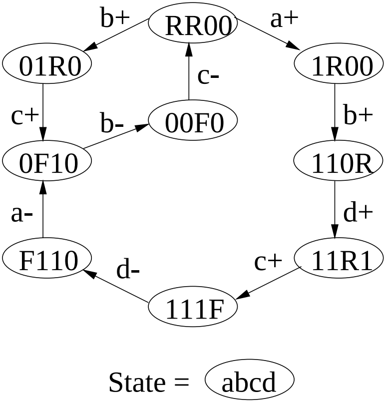


Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	∅
-0-1	∅
--11	11R1
010-	∅
0110	

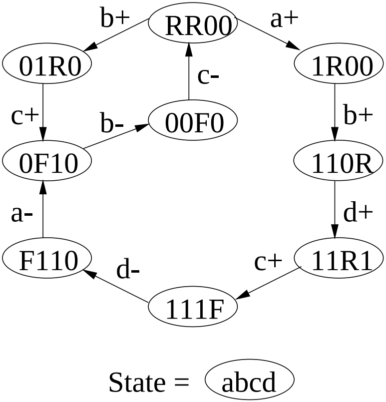
Candidate Implicant Algorithm Example



Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	11R1
010-	$\emptyset$
0110	F110

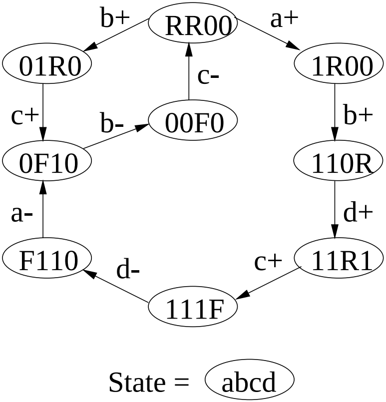


Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	∅
-0-1	∅
--11	11R1
010-	∅
0111	



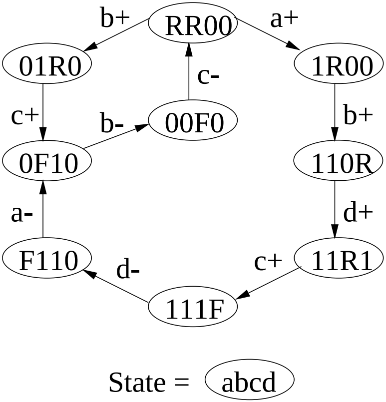
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	∅
-0-1	∅
--11	11R1
010-	∅
0111	∅

Candidate Implicant Algorithm Example



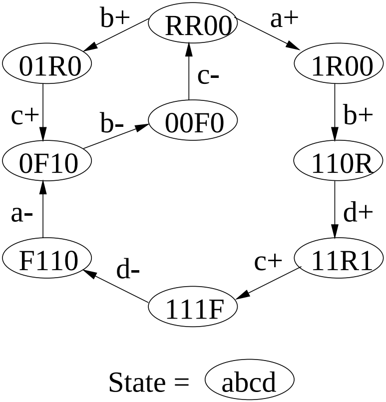
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	11R1
010-	$\emptyset$
101-	

Candidate Implicant Algorithm Example



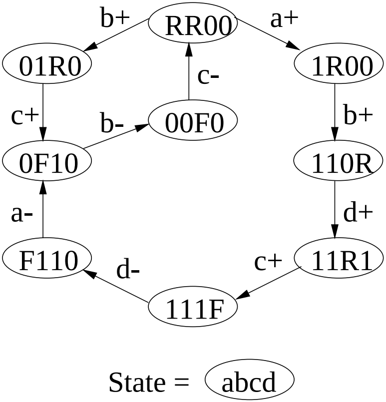
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	∅
-0-1	∅
--11	11R1
010-	∅
101-	∅

Candidate Implicant Algorithm Example



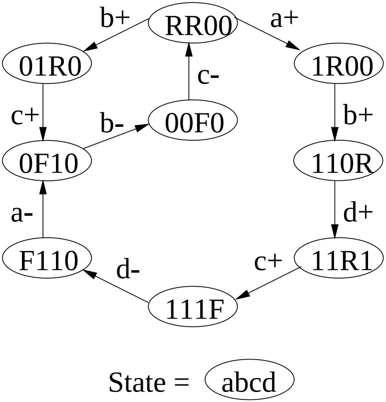
Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	11R1
010-	$\emptyset$
101-	$\emptyset$
1-10	

Candidate Implicant Algorithm Example

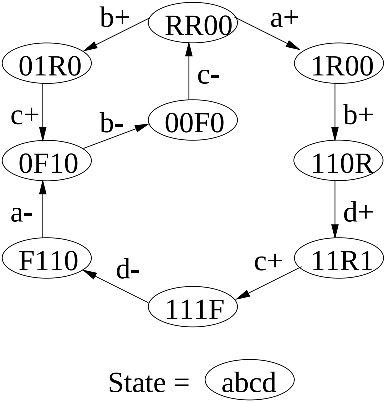


Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	11R1
010-	$\emptyset$
101-	$\emptyset$
1-10	111F

Candidate Implicant Algorithm Example



Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	$\emptyset$
-0-1	$\emptyset$
--11	11R1
010-	$\emptyset$
101-	$\emptyset$
1-10	111F
-110	



Primes	Implied States
01--	F110
1-1-	11R1
-11-	11R1, 01R0
0--1	∅
-0-1	∅
--11	11R1
010-	∅
101-	∅
1-10	111F
-110	111F, 01R0

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.

$$\bigvee_{i:s \in c_i} x_i.$$

- $ER(u*, k) = 0100$ 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.

$$\overline{x_i} \vee \bigvee_{j:s \in c_j} x_j .$$

- 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: $(x_1 + x_2)(\overline{x_1} + x_3 + x_5)\overline{x_3}\ \overline{x_5}\ \overline{x_8}$

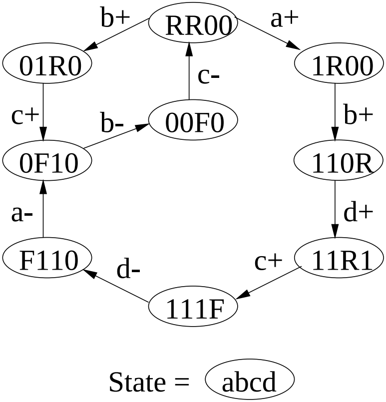
Setting Up the Constraint Matrix

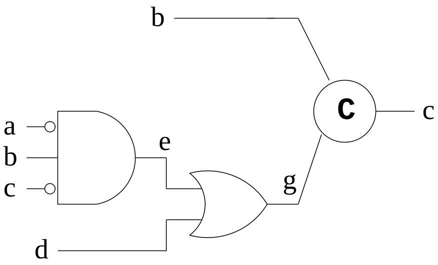
- Find x_i's that satisfy function with minimum cost.
- Since negated variables, the covering problem is binate.
- The constraint matrix has one row for each clause and one column for each candidate implicant.
- Rows divided into a *covering section* and a *closure section*.
- Covering section: row for 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 $ER(c+, 1)$

	01--	010-	1-1-	101-	-11-	0--1	-0-1	--11	1-10	-110
1	1	1	–	–	–	–	–	–	–	–
2	0	–	1	–	1	–	–	–	1	1
3	–	–	0	–	–	–	–	–	–	–
4	–	–	–	–	0	–	–	–	–	–
5	–	–	–	–	–	–	–	0	–	–
6	–	–	1	–	1	–	–	1	0	–
7	–	–	1	–	1	–	–	1	–	0

A Simple Example





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.

$$\bigcup_{l} C(u+, l) \supseteq ER(u+, l) \cup QS(u+)$$

- 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.

$$ER(u*,k) \subseteq [C(u*,k) \cap S] \subseteq \left[\bigcup_{l} ER(u*,l) \cup QS(u*)\right]$$

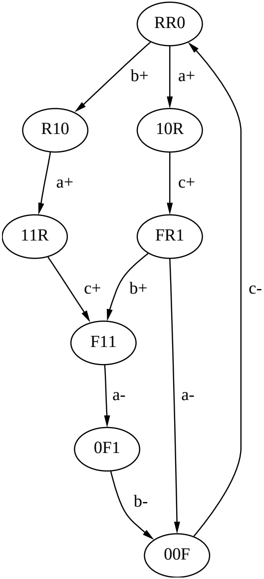
- Entrance constraint must be modified to allow the cover to be entered from any corresponding excitation region state.

$$[(s_i,t,s_j) \in \delta \wedge s_i \notin C(u*,k) \wedge s_j \in C(u*,k)] \Rightarrow s' \in \bigcup_{l} ER(u*,l)$$

- 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



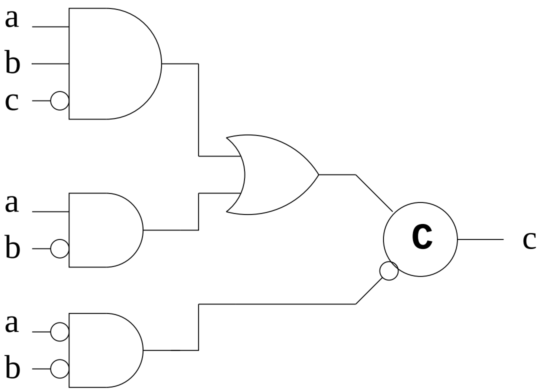
Example: No Sharing

- $ER(c+,1) = 100$ and $ER(c+,2) = 110$.
- Using the earlier constraints, the primes are found to be:

$$\begin{aligned} P(c+,1) &= \{\texttt{10-},\texttt{1-1},\texttt{-11}\} \\ P(c+,2) &= \{\texttt{11-},\texttt{1-1},\texttt{-11}\} \end{aligned}$$

- `10-` has no implied states.
- `11-` has implied state FR1 which can be covered by `1-1`, but this has implied state 10R which is an OFF-set state.
- Prime `11-` must be expanded to `110`.

Gate Sharing Example: Original Circuit

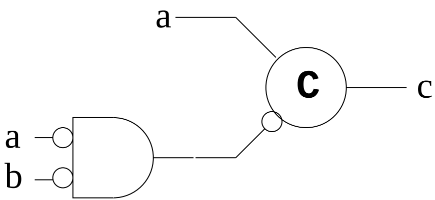


Example: Sharing

- $ER(c+, 1) = 100$ and $ER(c+, 2) = 110$.
- Using the new constraints, the primes are found to be:

$$\begin{aligned} P(c+, 1) &= \{\texttt{1--}, \texttt{-11}\} \\ P(c+, 2) &= \{\texttt{1--}, \texttt{-11}\} \end{aligned}$$

Gate Sharing Example: Optimized 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.

$\textbf{single\_cube}(SG, \texttt{technology})$
$\quad\textbf{foreach}\ \ u \in O$
$\quad\quad\textbf{EC} = \texttt{find\_excitation\_cubes}(SG);$
$\quad\quad\textbf{foreach}\ \ EC(u*, k) \in \textbf{EC}$
$\quad\quad\quad TC(u*, k) = \texttt{trigger\_cube}(SG, EC(u*, k));$
$\quad\quad\quad CS(u*, k) = \texttt{context\_signals}(SG, EC(u*, k), TC(u*, k));$
$\quad\quad\quad V(u*, k) = \texttt{violations}(SG, EC(u*, k), TC(u*, k), \texttt{tech});$
$\quad\quad\quad CC = \texttt{build\_cover\_table}(CS(u*, k), V(u*, k));$
$\quad\quad\quad C(u*, k) = \texttt{solve\_cover\_table}(CC, TC(u*, k));$
$\quad\quad solution(u) = \texttt{optimize\_logic}(C);$
$\quad\textbf{return}\ \ solution;$

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:

$$EC(u*, k)(v) \equiv \begin{cases} 0 & \text{if } \forall s \in ER(u*, k) \ . \ s(v) = 0 \\ 1 & \text{if } \forall s \in ER(u*, k) \ . \ s(v) = 1 \\ - & \text{otherwise} \end{cases}$$

- 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.

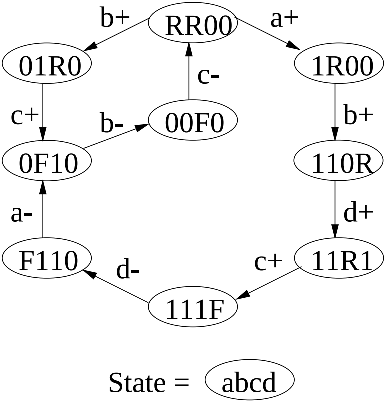
Trigger Cubes

- 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:

$$TC(u*,k)(v) \equiv \begin{cases} s_j(v) & \text{If } \exists (s_i, t, s_j) \in \delta \,.\, (t = v+ \vee t = v-) \wedge \\ & (s_i \notin ER(u*,k)) \wedge (s_j \in ER(u*,k)) \\ - & \text{otherwise} \end{cases}$$

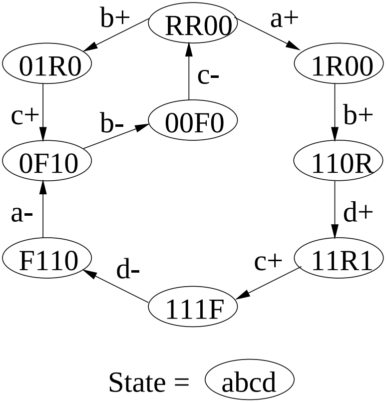
- The single cube cover of an excitation region must contain all its trigger signals (i.e., $C(u*, k) \subseteq TC(u*, k)$).
- Therefore, all trigger signals must be stable (i.e., $EC(u*, k) \subseteq TC(u*, k)$).

Example: Excitation and Trigger Cubes



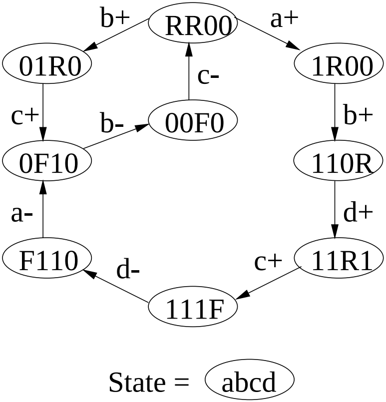
$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$		
$c+, 2$		
$c-, 1$		
$d+, 1$		
$d-, 1$		

Example: Excitation and Trigger Cubes



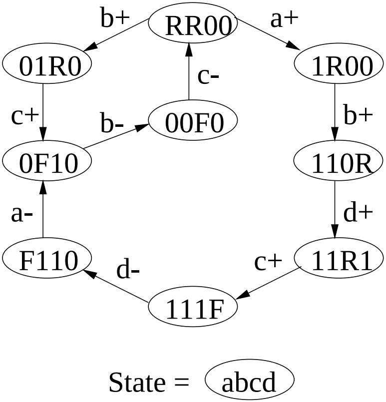
$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	
$c+, 2$		
$c-, 1$		
$d+, 1$		
$d-, 1$		

Example: Excitation and Trigger Cubes



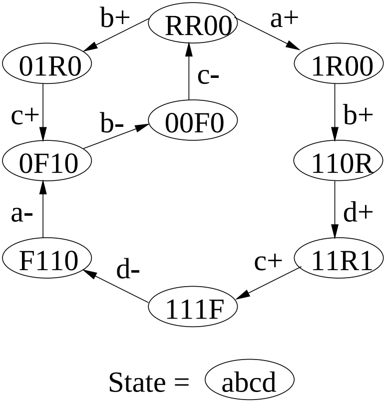
$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$		
$c-, 1$		
$d+, 1$		
$d-, 1$		

Example: Excitation and Trigger Cubes



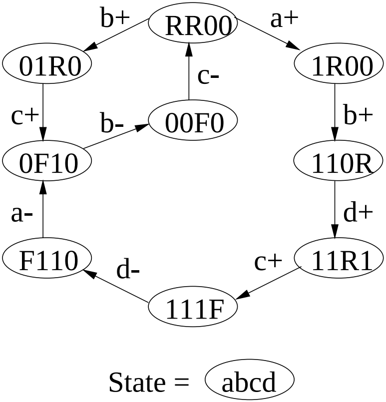
$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$	1101	
$c-, 1$		
$d+, 1$		
$d-, 1$		

Example: Excitation and Trigger Cubes



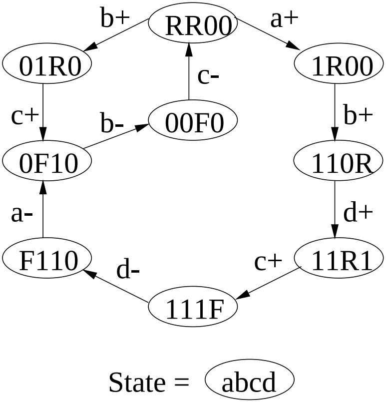
$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$	1101	---1
$c-, 1$		
$d+, 1$		
$d-, 1$		

Example: Excitation and Trigger Cubes



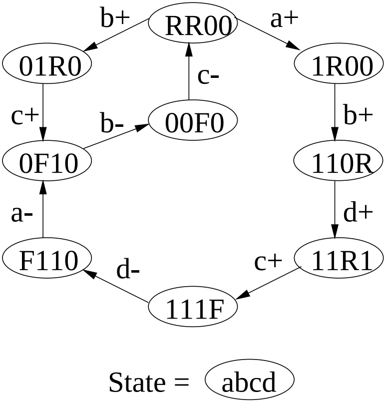
$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$	1101	---1
$c-, 1$	0010	
$d+, 1$		
$d-, 1$		

Example: Excitation and Trigger Cubes



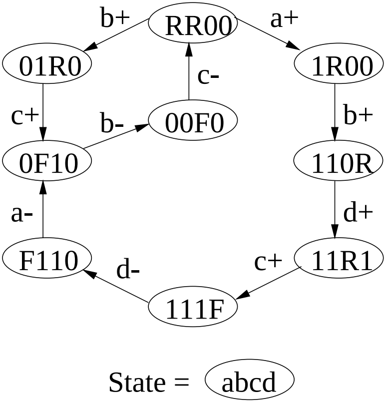
$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$	1101	---1
$c-, 1$	0010	-0--
$d+, 1$		
$d-, 1$		

Example: Excitation and Trigger Cubes

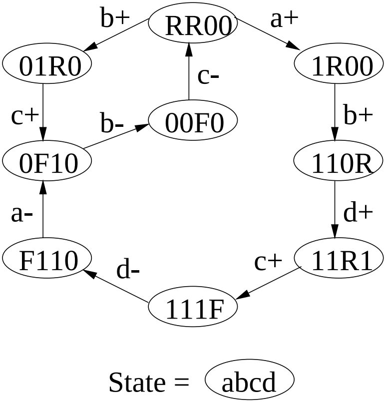


$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$	1101	---1
$c-, 1$	0010	-0--
$d+, 1$	1100	
$d-, 1$		

Example: Excitation and Trigger Cubes

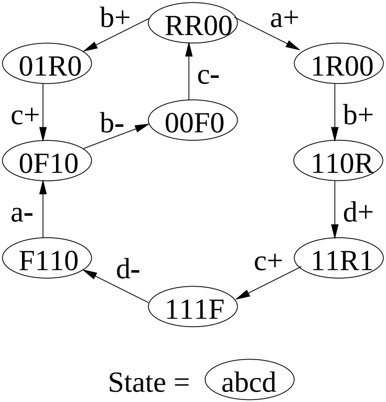


$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$	1101	---1
$c-, 1$	0010	-0--
$d+, 1$	1100	-1--
$d-, 1$		



$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$	1101	---1
$c-, 1$	0010	-0--
$d+, 1$	1100	-1--
$d-, 1$	1111	

Example: Excitation and Trigger Cubes



$u*, k$	$EC(u*, k)$	$TC(u*, k)$
$c+, 1$	0100	-1--
$c+, 2$	1101	---1
$c-, 1$	0010	-0--
$d+, 1$	1100	-1--
$d-, 1$	1111	--1-

Violating States

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

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

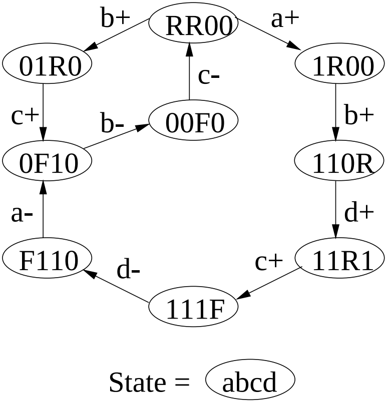
  and satisfies the required correctness constraints.
- Begin with a cube consisting only of the trigger signals.
- If this cover contains no states that violate the required correctness constraints, we are done.
- If not, context signals must be added to the cube to remove any *violating states*.
- For 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.

Violating States: gC Circuits

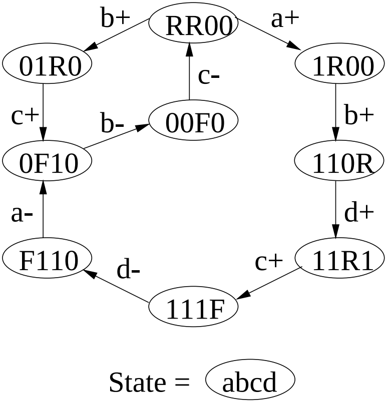
- In gC circuits, for a set region a state is a violating state when the trigger cube intersects the *falling* or *low* sets.
- Similarly, for a reset region, a state is a violating state when the trigger cube intersects the *rising* or *high* sets.

$$
\begin{aligned}
V(u+,k) &= \{s \in S \mid s \in TC(u+,k) \wedge s \in ES(u-) \cup QS(u-)\} \\
V(u-,k) &= \{s \in S \mid s \in TC(u-,k) \wedge s \in ES(u+) \cup QS(u+)\}
\end{aligned}
$$

Example: Violating States



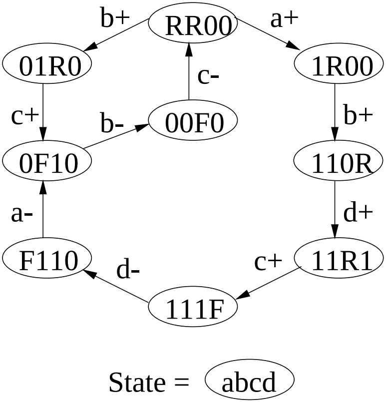
Example: Violating States



$TC(c+,1)$ -1- -

$V(c+,1)$

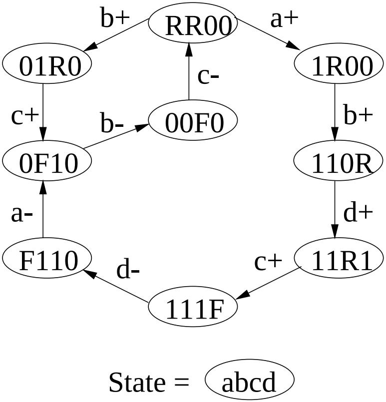
Example: Violating States



$TC(c+, 1)$ $\quad$ `-1--`

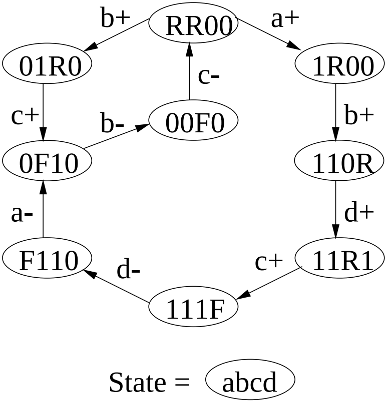
$V(c+, 1)$ $\quad$ $\{110R\}$

Example: Violating States



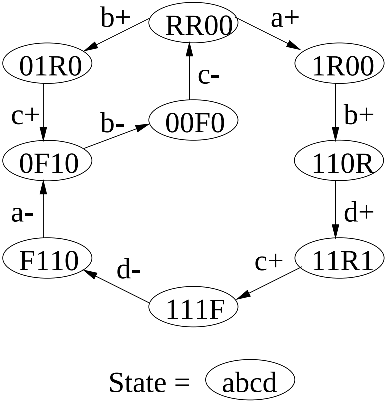
$TC(c+,1)$	$\texttt{-1--}$
$V(c+,1)$	$\{110R\}$
$TC(c+,2)$	$\texttt{---1}$
$V(c+,2)$	

Example: Violating States

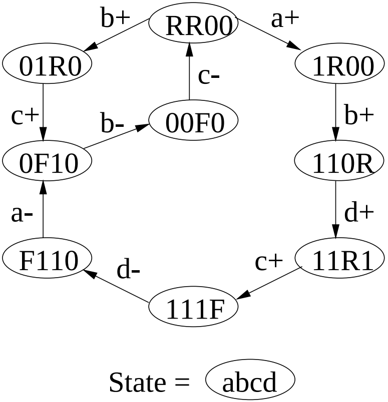


$$
\begin{array}{cc}
TC(c+,1) & \texttt{-1--} \\
V(c+,1) & \{110R\} \\
TC(c+,2) & \texttt{---1} \\
V(c+,2) & \emptyset
\end{array}
$$

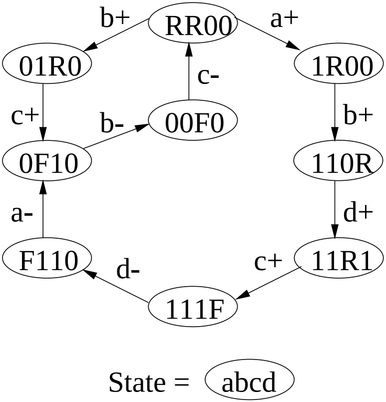
Example: Violating States



$TC(c+,1)$	-1--
$V(c+,1)$	$\{110R\}$
$TC(c+,2)$	---1
$V(c+,2)$	$\emptyset$
$TC(c-,1)$	-0--
$V(c-,1)$	

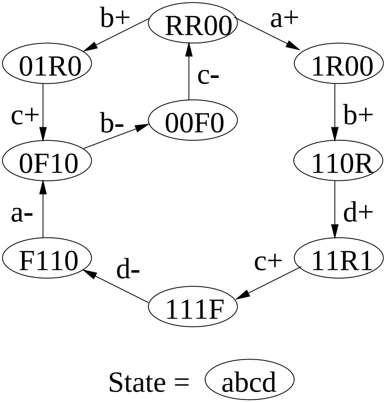


$TC(c+,1)$	`-1--`
$V(c+,1)$	$\{110R\}$
$TC(c+,2)$	`---1`
$V(c+,2)$	$\emptyset$
$TC(c-,1)$	`-0--`
$V(c-,1)$	$\emptyset$



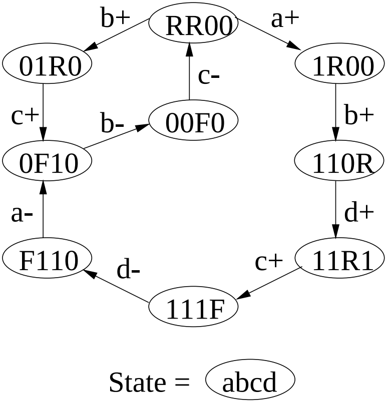
$TC(c+,1)$	`-1--`
$V(c+,1)$	$\{110R\}$
$TC(c+,2)$	`---1`
$V(c+,2)$	$\emptyset$
$TC(c-,1)$	`-0--`
$V(c-,1)$	$\emptyset$
$TC(d+,1)$	`-1--`
$V(d+,1)$	

Example: Violating States



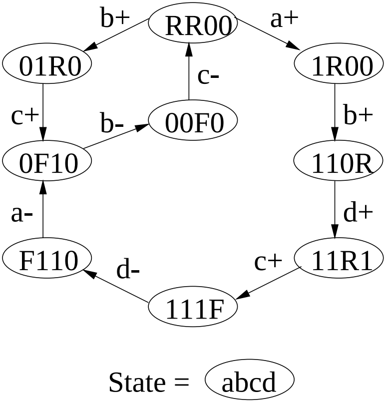
$TC(c+,1)$	`-1--`
$V(c+,1)$	$\{110R\}$
$TC(c+,2)$	`---1`
$V(c+,2)$	$\emptyset$
$TC(c-,1)$	`-0--`
$V(c-,1)$	$\emptyset$
$TC(d+,1)$	`-1--`
$V(d+,1)$	$\{111F, F110, 0F10, 01R0\}$

Example: Violating States



$TC(c+,1)$	-1--
$V(c+,1)$	$\{110R\}$
$TC(c+,2)$	---1
$V(c+,2)$	$\emptyset$
$TC(c-,1)$	-0--
$V(c-,1)$	$\emptyset$
$TC(d+,1)$	-1--
$V(d+,1)$	$\{111F, F110, 0F10, 01R0\}$
$TC(d-,1)$	--1-
$V(d-,1)$	

Example: Violating States

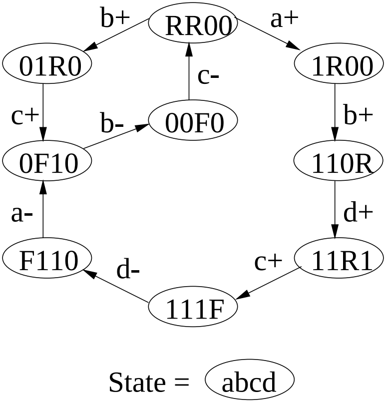


$TC(c+,1)$	`-1--`
$V(c+,1)$	$\{110R\}$
$TC(c+,2)$	`---1`
$V(c+,2)$	$\emptyset$
$TC(c-,1)$	`-0--`
$V(c-,1)$	$\emptyset$
$TC(d+,1)$	`-1--`
$V(d+,1)$	$\{111F, F110, 0F10, 01R0\}$
$TC(d-,1)$	`--1-`
$V(d-,1)$	$\emptyset$

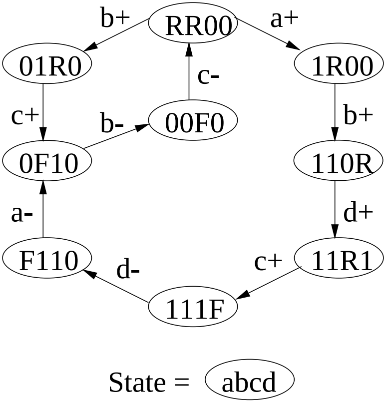
Context Signal Choices

- 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 $EC(u*, k)(v) = \overline{s(v)}$.

Example: Context Signals



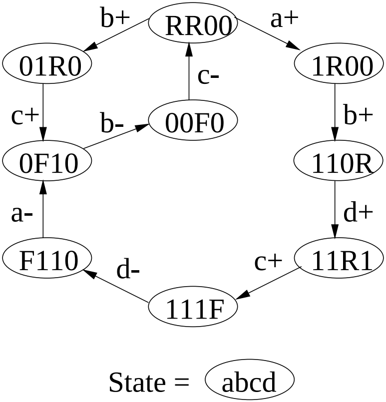
Example: Context Signals



$EC(c+, 1)$ `0100`

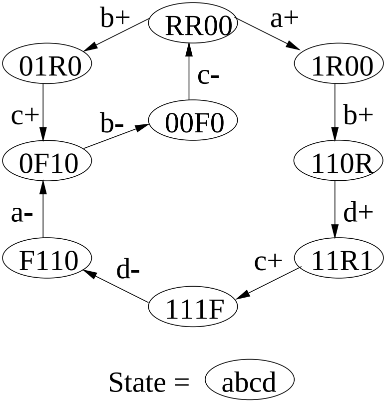
$TC(c+, 1)$ `-1--`

Example: Context Signals



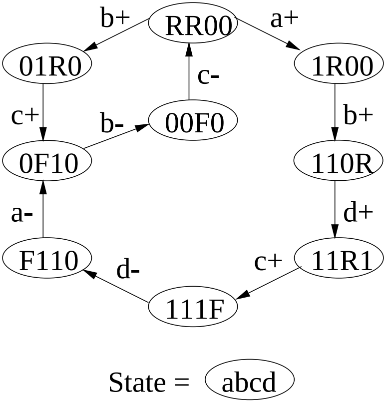
$EC(c+, 1)$	0100
$TC(c+, 1)$	-1--
$110R$	

Example: Context Signals



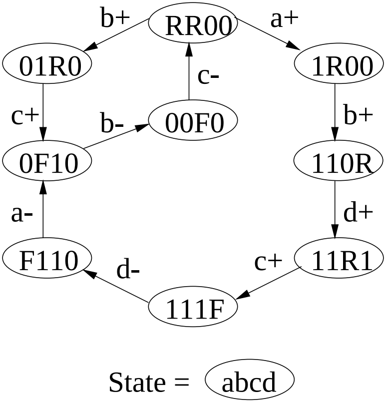
$EC(c+, 1)$	`0100`
$TC(c+, 1)$	`-1--`
$110R$	a

Example: Context Signals



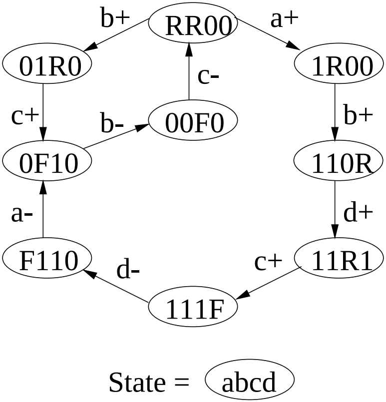
$EC(c+,1)$	`0100`
$TC(c+,1)$	`-1--`
$110R$	a
$EC(d+,1)$	`1100`
$TC(d+,1)$	`-1--`

Example: Context Signals



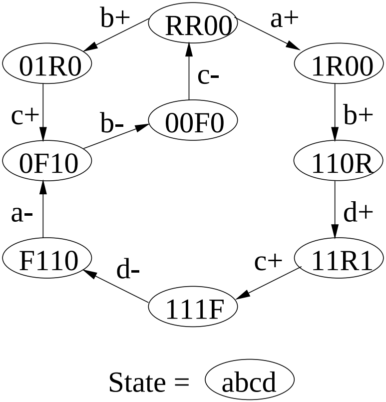
$EC(c+, 1)$	`0100`
$TC(c+, 1)$	`-1--`
$110R$	a
$EC(d+, 1)$	`1100`
$TC(d+, 1)$	`-1--`
$111F$	

Example: Context Signals

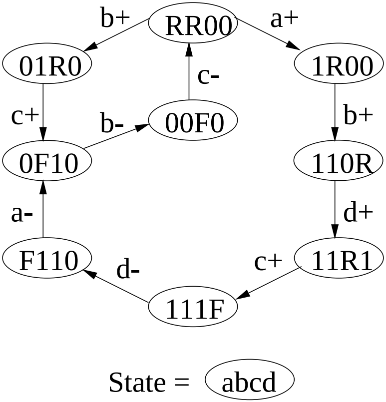


$EC(c+,1)$	0100
$TC(c+,1)$	-1--
$110R$	a
$EC(d+,1)$	1100
$TC(d+,1)$	-1--
$111F$	c,d

Example: Context Signals

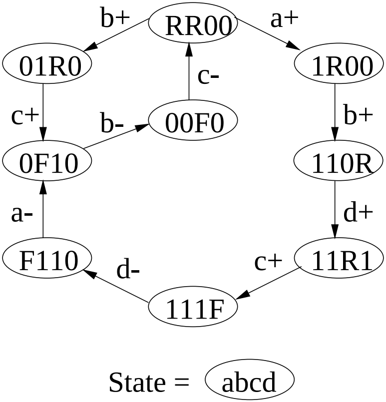


$EC(c+,1)$	0100
$TC(c+,1)$	-1--
$110R$	a
$EC(d+,1)$	1100
$TC(d+,1)$	-1--
$111F$	c, d
$F110$	

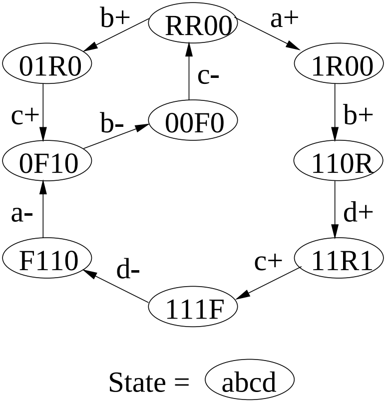


$EC(c+, 1)$	`0100`
$TC(c+, 1)$	`-1--`
$110R$	a
$EC(d+, 1)$	`1100`
$TC(d+, 1)$	`-1--`
$111F$	c, d
$F110$	c

Example: Context Signals

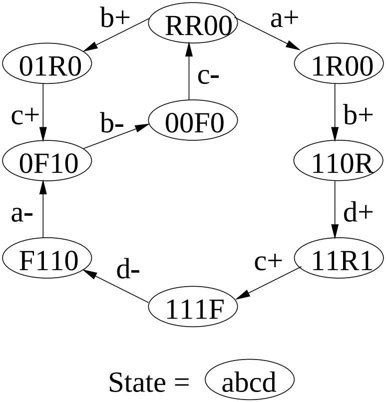


$EC(c+,1)$	`0100`
$TC(c+,1)$	`-1--`
$110R$	a
$EC(d+,1)$	`1100`
$TC(d+,1)$	`-1--`
$111F$	c, d
$F110$	c
$0F10$	



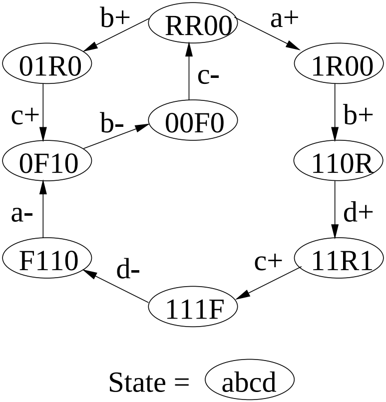
$EC(c+,1)$	0100
$TC(c+,1)$	-1--
$110R$	a
$EC(d+,1)$	1100
$TC(d+,1)$	-1--
$111F$	c,d
$F110$	c
$0F10$	a,c

Example: Context Signals



$EC(c+,1)$	`0100`
$TC(c+,1)$	`-1--`
$110R$	a
$EC(d+,1)$	`1100`
$TC(d+,1)$	`-1--`
$111F$	c,d
$F110$	c
$0F10$	a,c
$01R0$	

Example: Context Signals



$EC(c+,1)$	0100
$TC(c+,1)$	-1--
$110R$	a
$EC(d+,1)$	1100
$TC(d+,1)$	-1--
$111F$	c, d
$F110$	c
$0F10$	a, c
$01R0$	a

Setting Up the Covering Problem

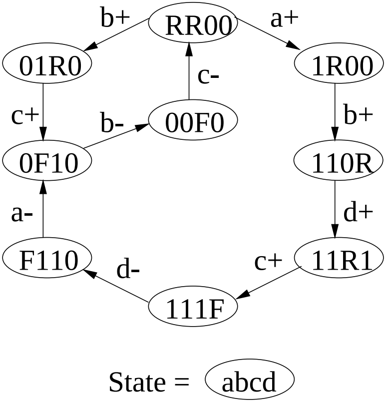
- The constraint matrix has a row for each violating state and a column for each context signal.
- The constraint matrix for $ER(d+, 1)$ is shown below:

$$\begin{array}{cccc} & a & c & d \\ 111F & - & 1 & 1 \\ F110 & - & 1 & - \\ 0F10 & 1 & 1 & - \\ 01R0 & 1 & - & - \end{array}$$

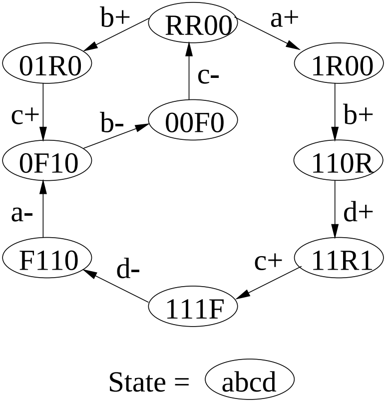
Gate Level Circuits: Cover Violations

- 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



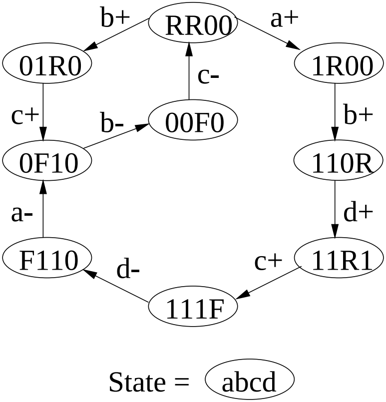
Example: Cover Violations



$TC(c+, 1)$ -1- -

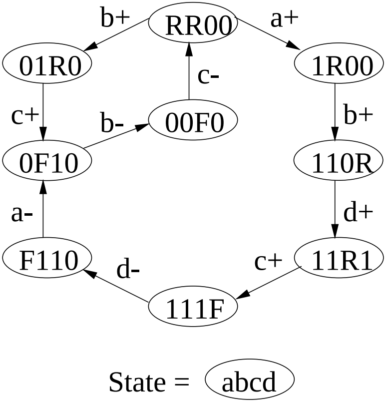
$CV(c+, 1)$

Example: Cover Violations



$TC(c+,1)$	-1- -
$CV(c+,1)$	$\{110R, 11R1\}$

Example: Cover Violations



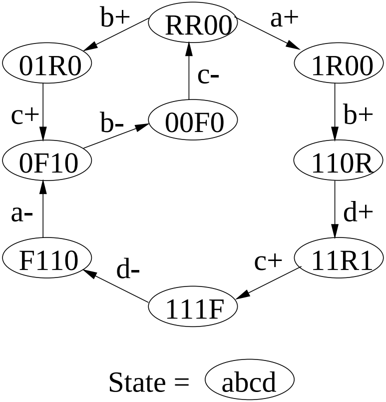
$TC(c+,1)$	-1--
$CV(c+,1)$	$\{110R, 11R1\}$
$TC(c+,2)$	---1
$CV(c+,2)$	$\emptyset$
$TC(c-,1)$	-0--
$CV(c-,1)$	$\emptyset$
$TC(d+,1)$	-1--
$CV(d+,1)$	$\{111F, F110, 0F10, 01R0\}$
$TC(d-,1)$	--1-
$CV(d-,1)$	$\emptyset$

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.

$$
\begin{aligned}
EV(u+,k) &= \{s_j \in S \mid (s_i, v*, s_j) \in \delta \wedge s_j \in QS(u+) \\
&\quad \wedge\ s_j \in TC(u+,k) \wedge\ EC(u+,k)(v) = \overline{s_i(v)} \\
EV(u-,k) &= \{s_j \in S \mid (s_i, v*, s_j) \in \delta \wedge s_j \in QS(u-) \\
&\quad \wedge\ s_j \in TC(u-,k) \wedge\ EC(u-,k)(v) = \overline{s_i(v)}
\end{aligned}
$$

- 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.

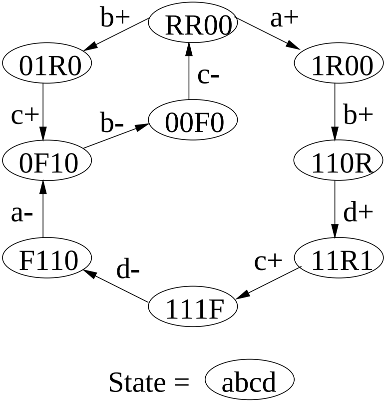


Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

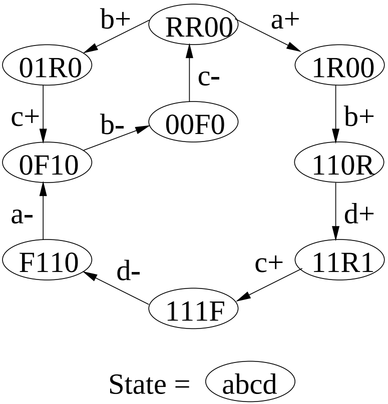
$EV(c+, 1) =$

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations

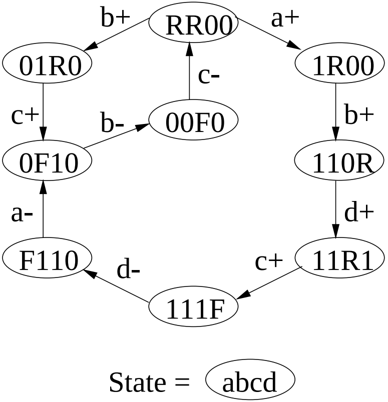


$EC(c+, 1) = 0100$
$TC(c+, 1) = -1--$
$EV(c+, 1) =$
$(RR00, a+, 1R00)$?

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



$EC(c+, 1) = 0100$

$TC(c+, 1) = -1 - -$

$EV(c+, 1) =$

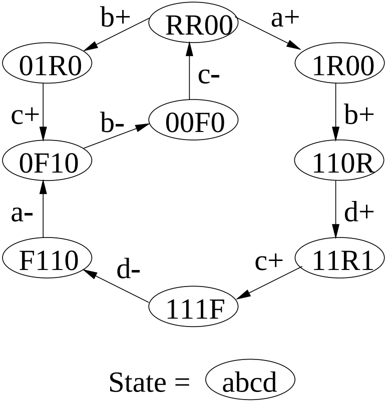
$(RR00, a+, 1R00)$? No 1

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



$EC(c+,1) = 0100$

$TC(c+,1) = -1--$

$EV(c+,1) =$

$(RR00, a+, 1R00)$? No 1

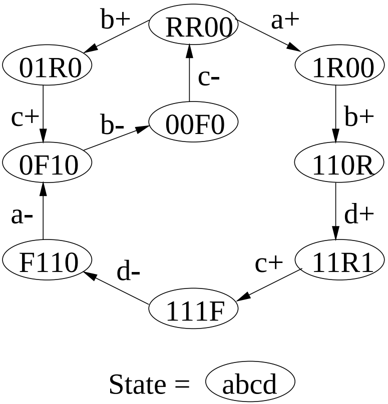
$(1R00, b+, 110R)$?

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*,k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*,k)$
3. $EC(u*,k)(v) = \overline{s_i(v)}$

Example: Entrance Violations

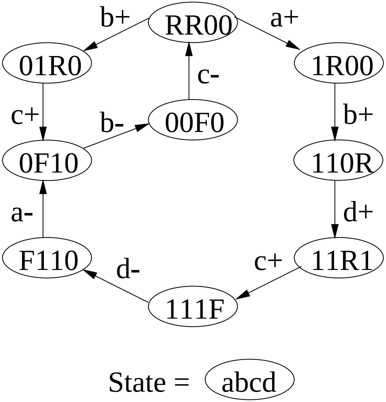


$EC(c+, 1) = 0100$
$TC(c+, 1) = -1 - -$
$EV(c+, 1) =$
$(RR00, a+, 1R00)$? No 1
$(1R00, b+, 110R)$? No 1

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



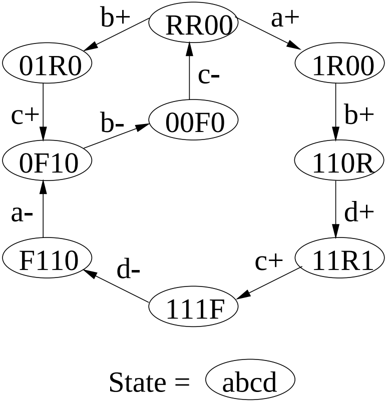
$EC(c+,1) = 0100$
$TC(c+,1) = -1--$
$EV(c+,1) =$
$(RR00, a+, 1R00)$? No 1
$(1R00, b+, 110R)$? No 1
$(110R, d+, 11R1)$?

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

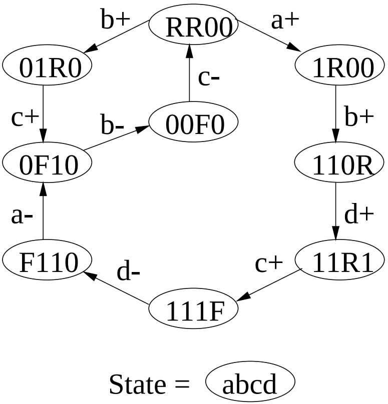
$EV(c+, 1) =$

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

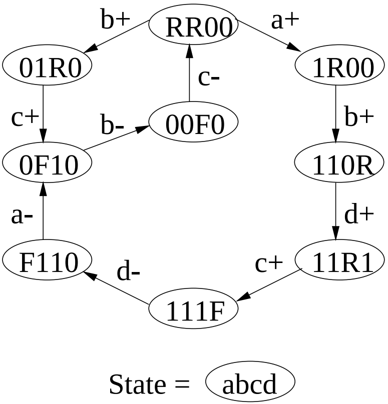
Example: Entrance Violations



$EC(c+, 1) = 0100$
$TC(c+, 1) = -1 - -$
$EV(c+, 1) =$
$(RR00, a+, 1R00)$? No 1
$(1R00, b+, 110R)$? No 1
$(110R, d+, 11R1)$? No 1
$(11R1, c+, 111F)$?

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

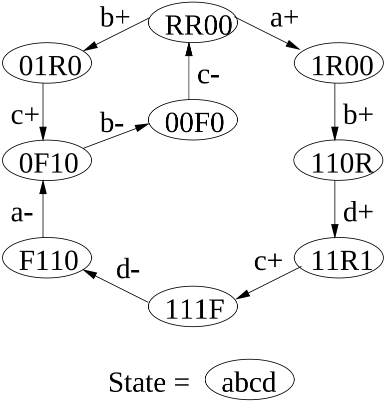


$EC(c+,1) = 0100$
$TC(c+,1) = -1--$
$EV(c+,1) =$
$(RR00, a+, 1R00)$? No 1
$(1R00, b+, 110R)$? No 1
$(110R, d+, 11R1)$? No 1
$(11R1, c+, 111F)$? No 3

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations

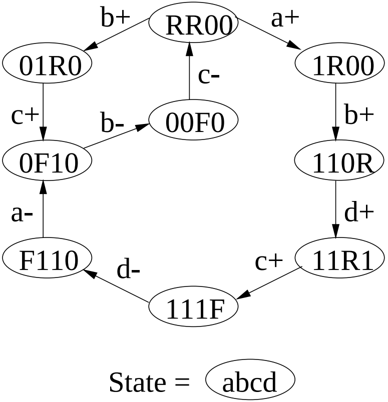


$EC(c+, 1) = 0100$
$TC(c+, 1) = -1--$
$EV(c+, 1) =$
$(RR00, a+, 1R00)$? No 1
$(1R00, b+, 110R)$? No 1
$(110R, d+, 11R1)$? No 1
$(11R1, c+, 111F)$? No 3
$(111F, d-, F110)$?

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



$EC(c+,1) = 0100$

$TC(c+,1) = -1--$

$EV(c+,1) = \{$ F110

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

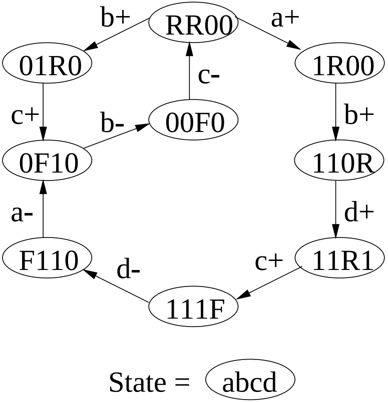
$(111F, d-, F110)$? Yes

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

$EV(c+, 1) = \{$ F110

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

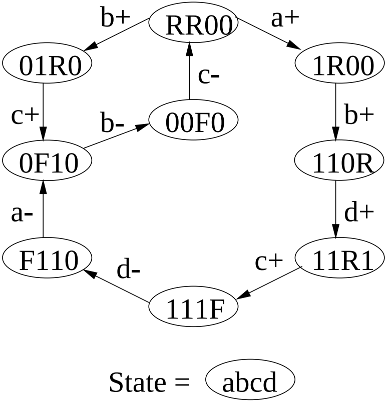
$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

$(111F, d-, F110)$? Yes

$(F110, a-, 0F10)$?

Example: Entrance Violations



$EC(c+, 1) = 0100$

$TC(c+, 1) = -1 - -$

$EV(c+, 1) = \{$ F110, 0F10

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

$(111F, d-, F110)$? Yes

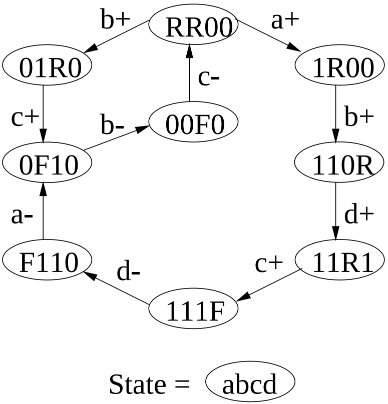
$(F110, a-, 0F10)$? Yes

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

$EV(c+, 1) =$ { F110, 0F10

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

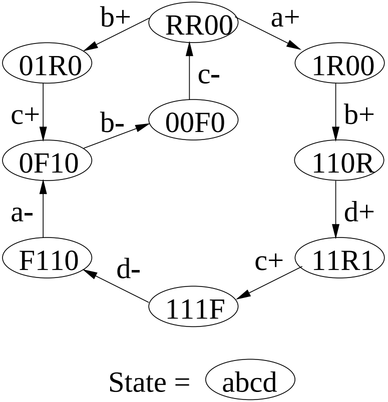
$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

$(111F, d-, F110)$? Yes

$(F110, a-, 0F10)$? Yes

$(0F10, b-, 00F0)$?



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1 - -$

$EV(c+, 1) = \{$ F110, 0F10

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

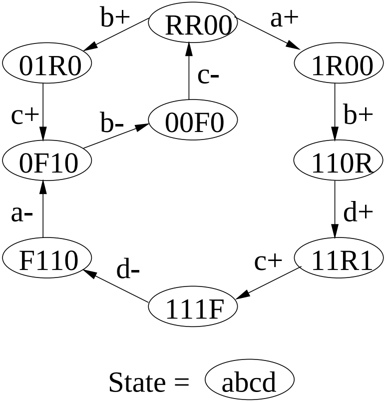
$(11R1, c+, 111F)$? No 3

$(111F, d-, F110)$? Yes

$(F110, a-, 0F10)$? Yes

$(0F10, b-, 00F0)$? No 1

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

$EV(c+, 1) = \{$ F110, 0F10

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

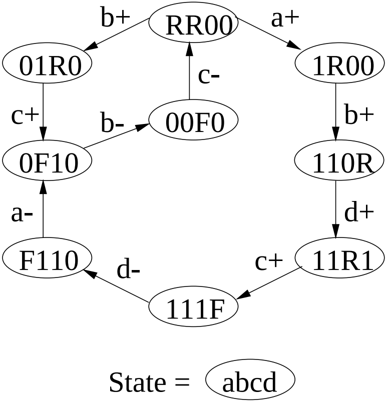
$(11R1, c+, 111F)$? No 3

$(111F, d-, F110)$? Yes

$(F110, a-, 0F10)$? Yes

$(0F10, b-, 00F0)$? No 1

$(00F0, c-, RR00)$?



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

$EV(c+, 1) = \{$ F110, 0F10

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

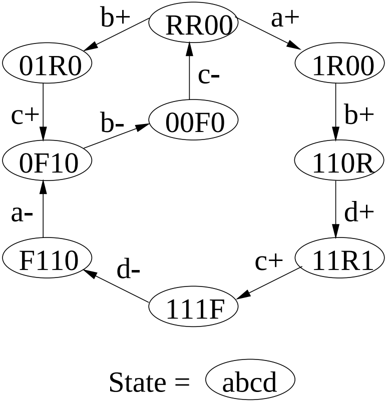
$(111F, d-, F110)$? Yes

$(F110, a-, 0F10)$? Yes

$(0F10, b-, 00F0)$? No 1

$(00F0, c-, RR00)$? No 1

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1 - -$

$EV(c+, 1) = \{$ F110, 0F10

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

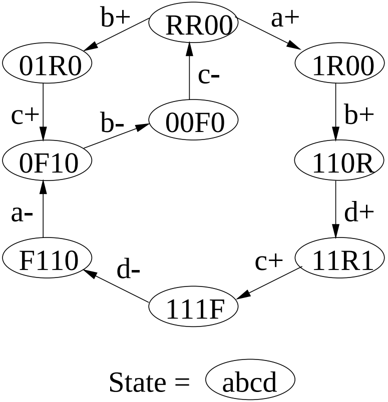
$(111F, d-, F110)$? Yes

$(F110, a-, 0F10)$? Yes

$(0F10, b-, 00F0)$? No 1

$(00F0, c-, RR00)$? No 1

$(RR00, b+, 01R0)$?



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

$EV(c+, 1) = \{$ F110, 0F10

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

$(111F, d-, F110)$? Yes

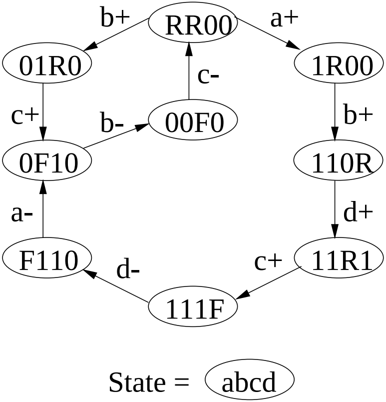
$(F110, a-, 0F10)$? Yes

$(0F10, b-, 00F0)$? No 1

$(00F0, c-, RR00)$? No 1

$(RR00, b+, 01R0)$? No 1

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

$EV(c+, 1) = \{$ F110, 0F10

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

$(111F, d-, F110)$? Yes

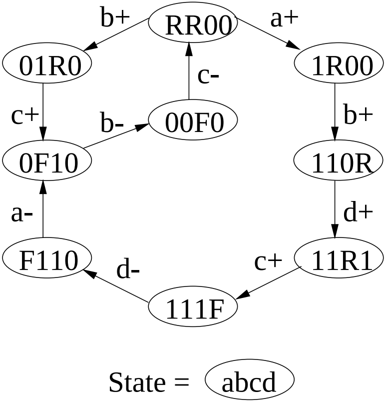
$(F110, a-, 0F10)$? Yes

$(0F10, b-, 00F0)$? No 1

$(00F0, c-, RR00)$? No 1

$(RR00, b+, 01R0)$? No 1

$(01R0, c+, 0F10)$?



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(c+, 1) = 0100$

$TC(c+, 1) = -1--$

$EV(c+, 1) = \{$ F110, 0F10 $\}$

$(RR00, a+, 1R00)$? No 1

$(1R00, b+, 110R)$? No 1

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 3

$(111F, d-, F110)$? Yes

$(F110, a-, 0F10)$? Yes

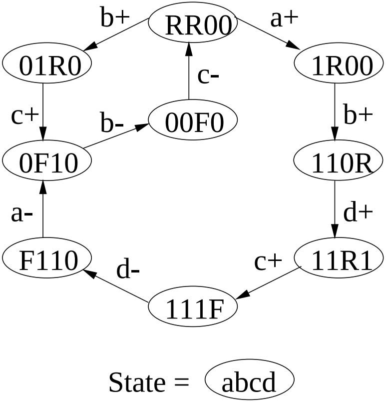
$(0F10, b-, 00F0)$? No 1

$(00F0, c-, RR00)$? No 1

$(RR00, b+, 01R0)$? No 1

$(01R0, c+, 0F10)$? No 3

Example: Entrance Violations

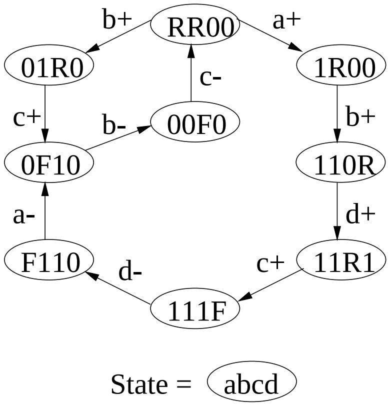


Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



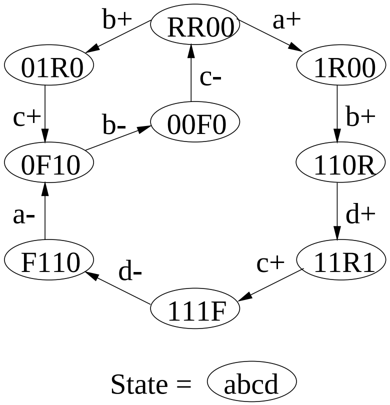
$EC(d-, 1) = 1111$
$TC(d-, 1) = --1-$
$EV(d-, 1) =$

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations

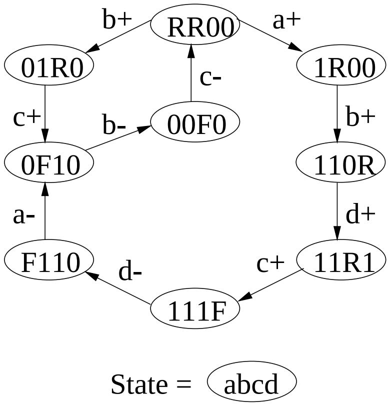


$EC(d-,1) = 1111$
$TC(d-,1) = --1-$
$EV(d-,1) =$
$(RR00, a+, 1R00)$?

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations

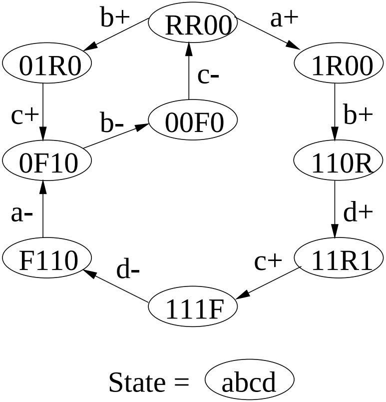


$EC(d-, 1) = 1111$
$TC(d-, 1) = --1-$
$EV(d-, 1) =$
$(RR00, a+, 1R00)$? No 2

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

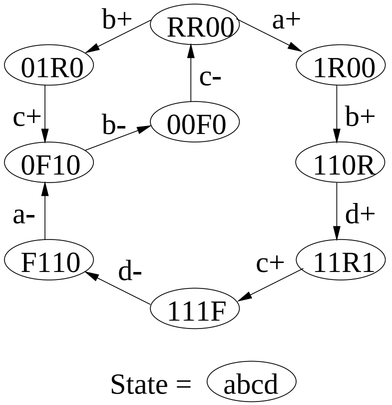
$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$?



$EC(d-,1) = 1111$

$TC(d-,1) = --1-$

$EV(d-,1) =$

$(RR00, a+, 1R00)$? No 2

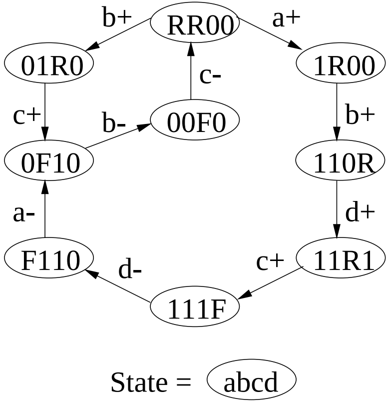
$(1R00, b+, 110R)$? No 2

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations

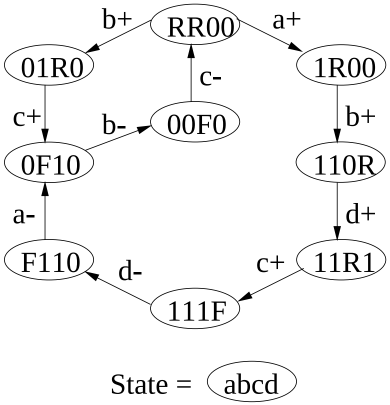


$EC(d-,1) = 1111$
$TC(d-,1) = --1-$
$EV(d-,1) =$
$(RR00, a+, 1R00)$? No 2
$(1R00, b+, 110R)$? No 2
$(110R, d+, 11R1)$?

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



$EC(d-,1) = 1111$

$TC(d-,1) = --1-$

$EV(d-,1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

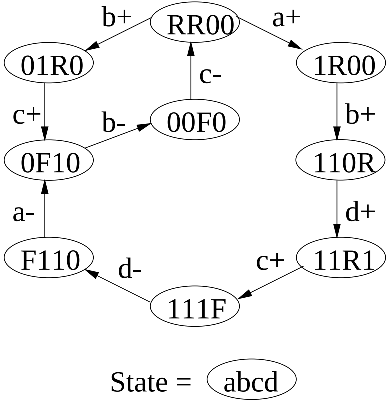
$(110R, d+, 11R1)$? No 1

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations

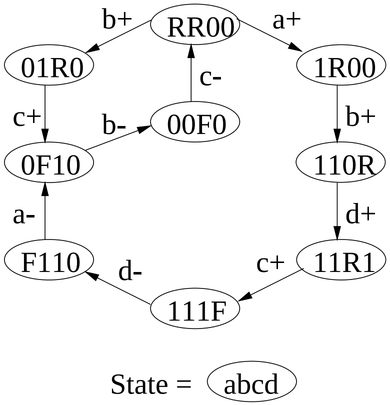


$EC(d-,1) = 1111$
$TC(d-,1) = --1-$
$EV(d-,1) =$
$(RR00, a+, 1R00)$? No 2
$(1R00, b+, 110R)$? No 2
$(110R, d+, 11R1)$? No 1
$(11R1, c+, 111F)$?

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



$EC(d-,1) = 1111$

$TC(d-,1) = --1-$

$EV(d-,1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

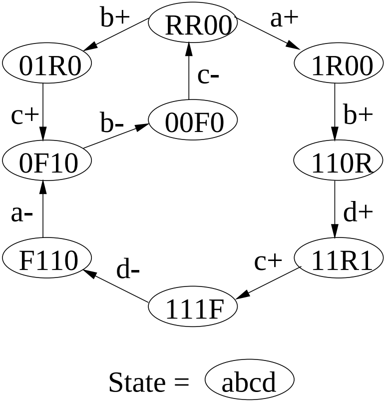
$(11R1, c+, 111F)$? No 1

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations

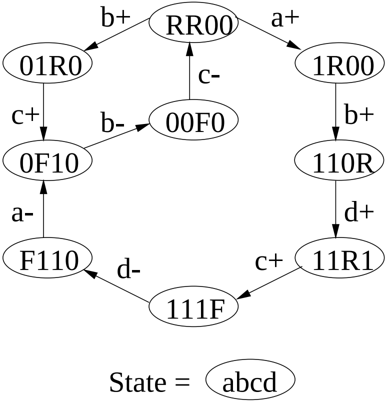


$EC(d-,1) = 1111$
$TC(d-,1) = --1-$
$EV(d-,1) =$
$(RR00, a+, 1R00)$? No 2
$(1R00, b+, 110R)$? No 2
$(110R, d+, 11R1)$? No 1
$(11R1, c+, 111F)$? No 1
$(111F, d-, F110)$?

Consider each $(s_i, v*, s_j)$
s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



$$EC(d-,1) = 1111$$
$$TC(d-,1) = --1-$$
$$EV(d-,1) =$$
$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

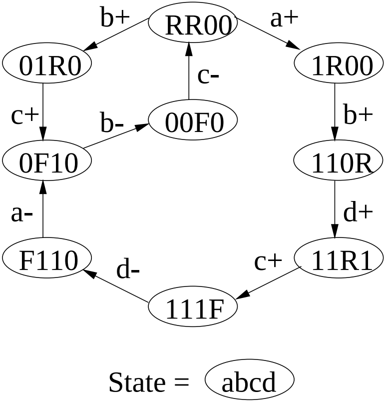
$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$



$EC(d-,1) = 1111$

$TC(d-,1) = --1-$

$EV(d-,1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

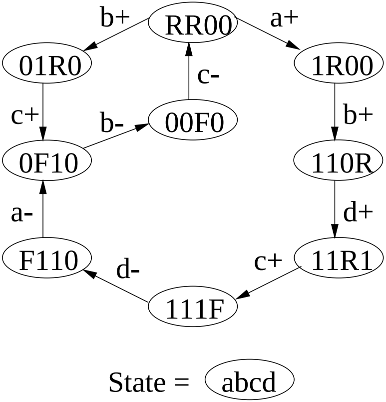
$(F110, a-, 0F10)$?

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



$EC(d-,1) = 1111$

$TC(d-,1) = --1-$

$EV(d-,1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

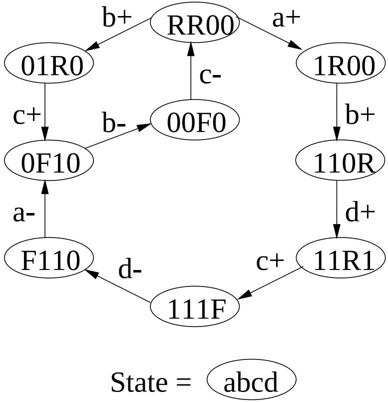
$(F110, a-, 0F10)$? No 3

Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

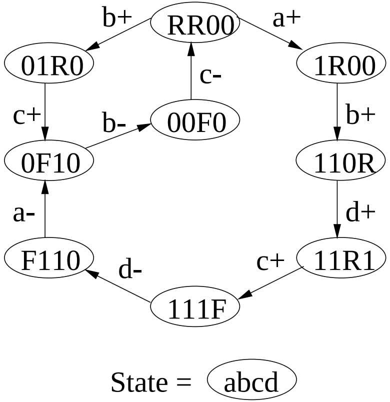
$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

$(F110, a-, 0F10)$? No 3

$(0F10, b-, 00F0)$?

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

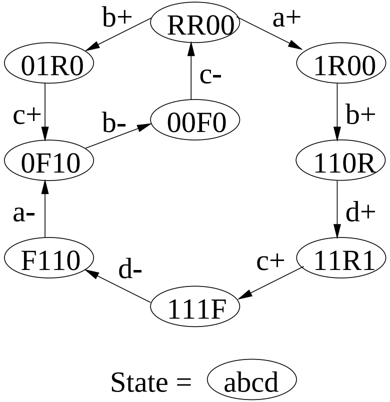
$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

$(F110, a-, 0F10)$? No 3

$(0F10, b-, 00F0)$? No 3

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 1

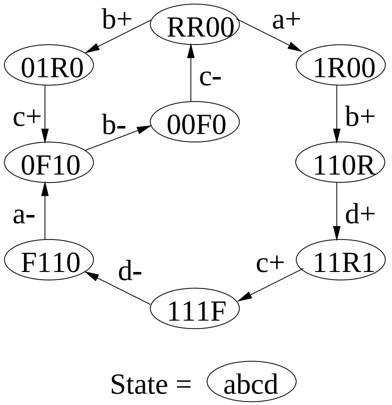
$(111F, d-, F110)$? No 3

$(F110, a-, 0F10)$? No 3

$(0F10, b-, 00F0)$? No 3

$(00F0, c-, RR00)$?

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

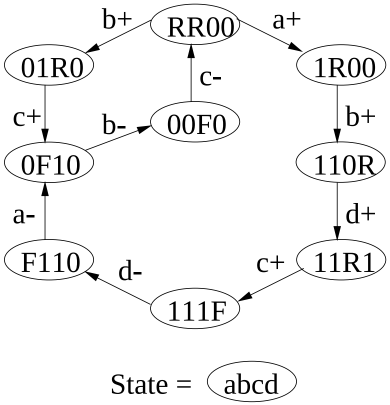
$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

$(F110, a-, 0F10)$? No 3

$(0F10, b-, 00F0)$? No 3

$(00F0, c-, RR00)$? No 3



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

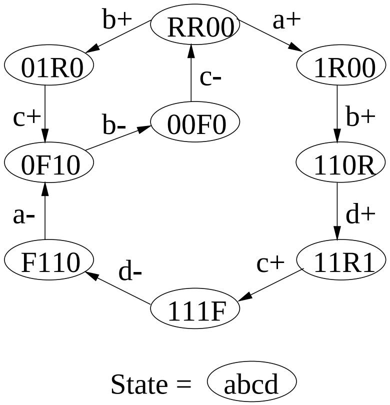
$(F110, a-, 0F10)$? No 3

$(0F10, b-, 00F0)$? No 3

$(00F0, c-, RR00)$? No 3

$(RR00, b+, 01R0)$?

Example: Entrance Violations



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 1

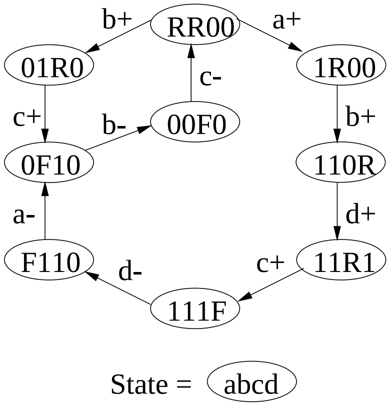
$(111F, d-, F110)$? No 3

$(F110, a-, 0F10)$? No 3

$(0F10, b-, 00F0)$? No 3

$(00F0, c-, RR00)$? No 3

$(RR00, b+, 01R0)$? No 2



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) =$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

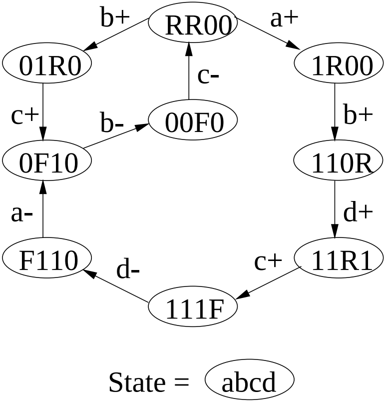
$(F110, a-, 0F10)$? No 3

$(0F10, b-, 00F0)$? No 3

$(00F0, c-, RR00)$? No 3

$(RR00, b+, 01R0)$? No 2

$(01R0, c+, 0F10)$?



Consider each $(s_i, v*, s_j)$

s_j is in $EV(u*, k)$ when:

1. $s_j \in QS(u*)$
2. $s_j \in TC(u*, k)$
3. $EC(u*, k)(v) = \overline{s_i(v)}$

$EC(d-, 1) = 1111$

$TC(d-, 1) = --1-$

$EV(d-, 1) = \{\ 0F10\ \}$

$(RR00, a+, 1R00)$? No 2

$(1R00, b+, 110R)$? No 2

$(110R, d+, 11R1)$? No 1

$(11R1, c+, 111F)$? No 1

$(111F, d-, F110)$? No 3

$(F110, a-, 0F10)$? No 3

$(0F10, b-, 00F0)$? No 3

$(00F0, c-, RR00)$? No 3

$(RR00, b+, 01R0)$? No 2

$(01R0, c+, 0F10)$? Yes!

Setting Up the Covering Problem

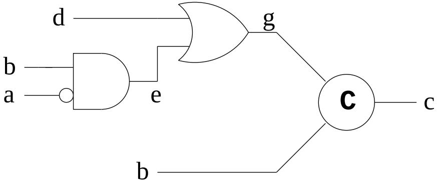
- Since inclusion of certain context signals cause some states to have entrance violations, the covering problem is binate.
- There is a row in the constraint matrix for each violation and each violation that could arise from a context signal choice.
- There is a column for each context signal.
- The 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

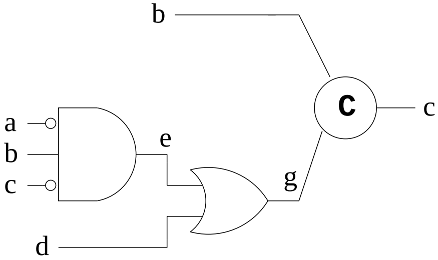
- The constraint matrix for $ER(c+, 1)$ is shown below:

$$\begin{array}{cccc} & a & c & d \\ 110R & 1 & - & - \\ 11R1 & 1 & - & 1 \\ 0F10 & 0 & 1 & - \\ F110 & 1 & 1 & 0 \end{array}$$

Example: gC versus Standard-C

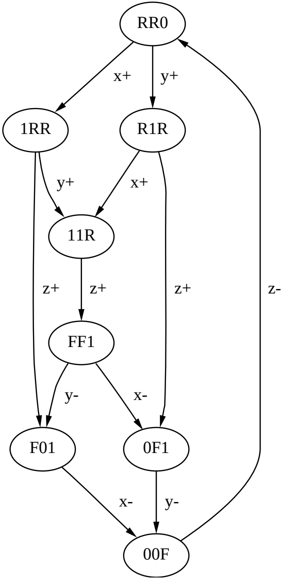


gC Implementation

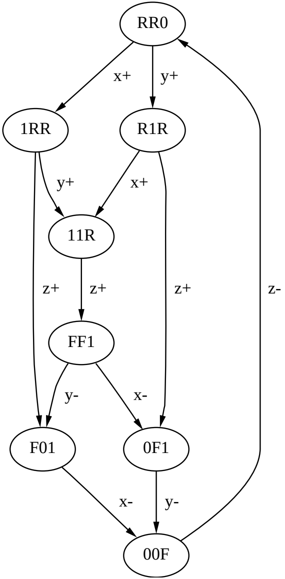


Standard C-Implementation

Example: Non-Persistent Trigger Signals

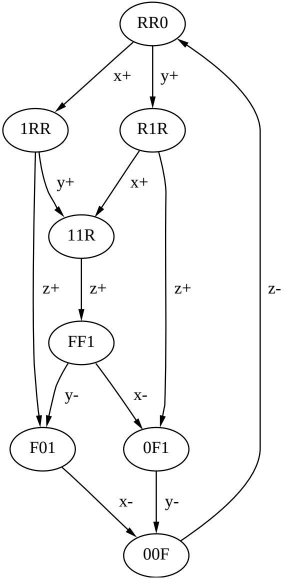


Example: Non-Persistent Trigger Signals



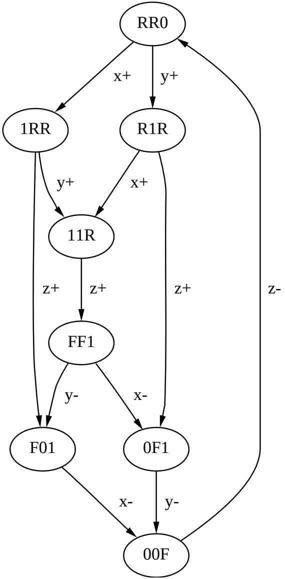
$EC(z+, 1) =$

Example: Non-Persistent Trigger Signals



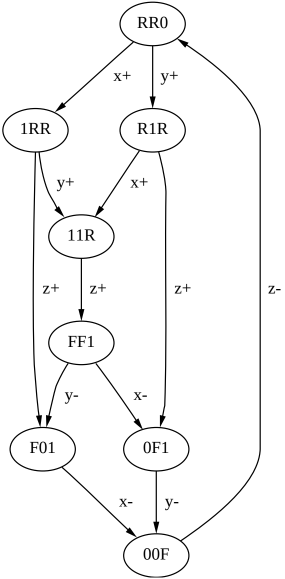
$$EC(z+, 1) = {-}{-}0$$

Example: Non-Persistent Trigger Signals



$$EC(z+,1) = --0$$
$$TC(z+,1) =$$

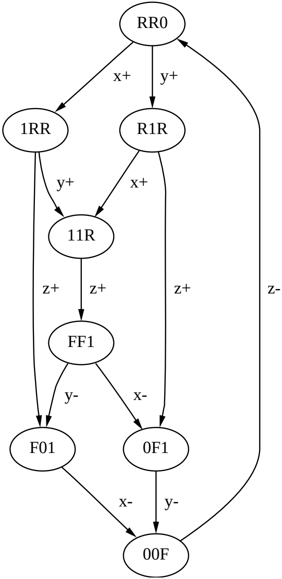
Example: Non-Persistent Trigger Signals



$EC(z+,1) = \texttt{--0}$

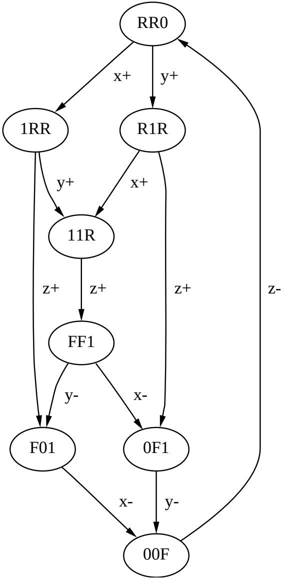
$TC(z+,1) = \texttt{11-}$

Example: Non-Persistent Trigger Signals



$EC(z+,1) = \texttt{--0}$
$TC(z+,1) = \texttt{11-}$
Trigger signals are not stable

Example: Non-Persistent Trigger Signals



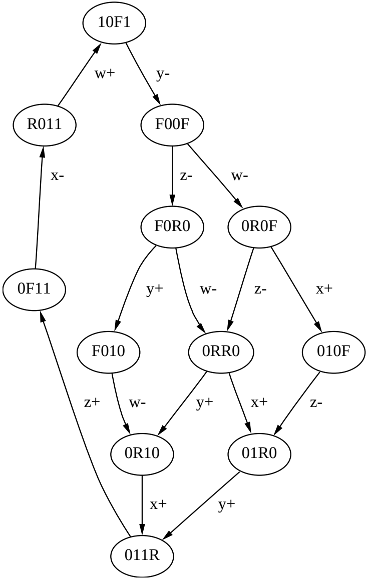
$EC(z+,1) = \texttt{--0}$

$TC(z+,1) = \texttt{11-}$

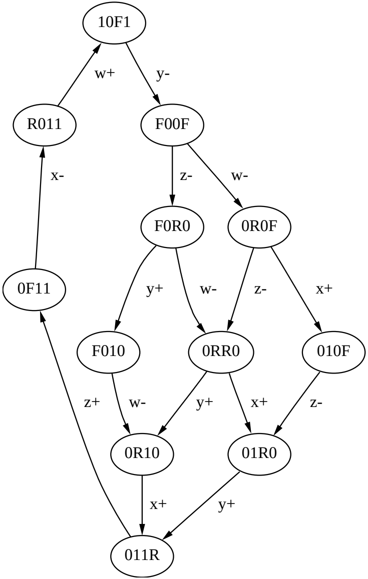
Trigger signals are not stable

No single cube cover exists

Example: Non-Persistent Trigger Signals

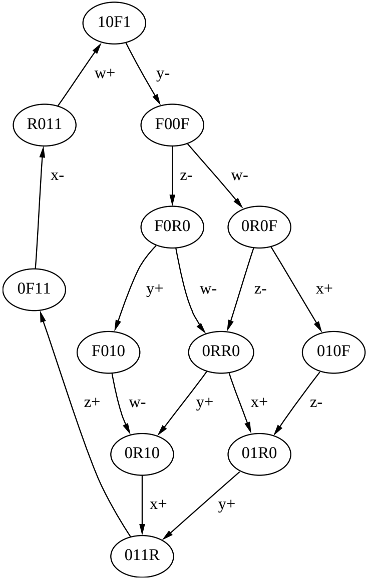


Example: Non-Persistent Trigger Signals



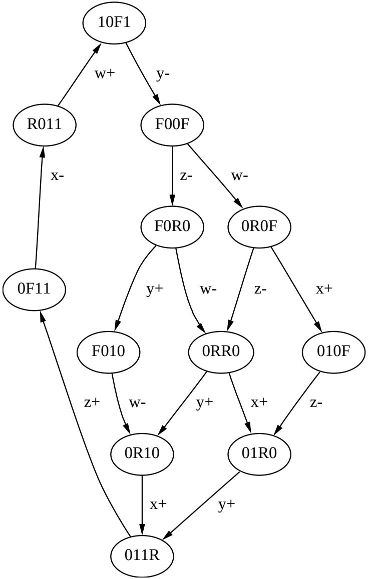
$$EC(w-, 1) =$$

Example: Non-Persistent Trigger Signals



$$EC(w-,1) = \texttt{10--}$$

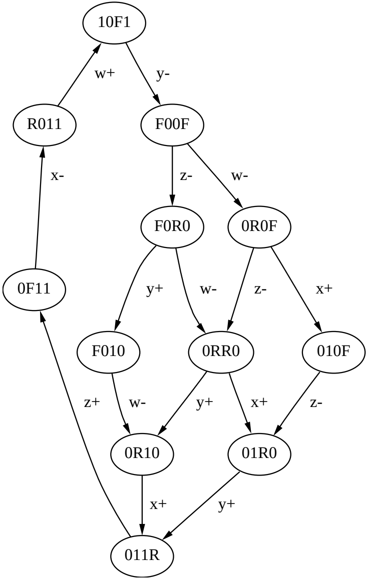
Example: Non-Persistent Trigger Signals



$EC(w-, 1) = \mathtt{10--}$

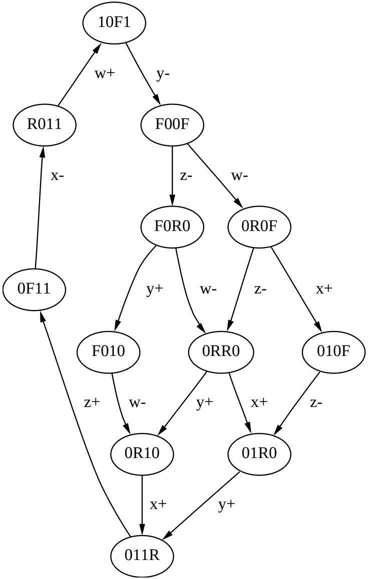
$TC(w-, 1) =$

Example: Non-Persistent Trigger Signals



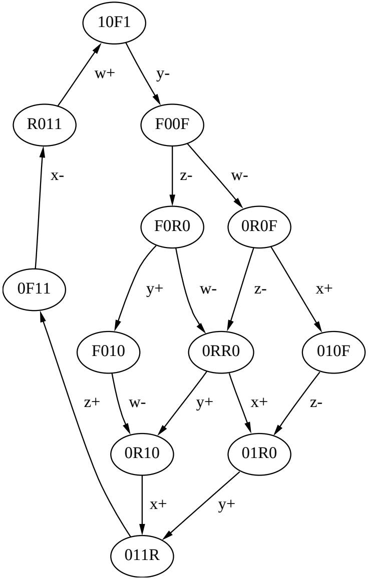
$$EC(w-,1) = 10--$$
$$TC(w-,1) = --0-$$

Example: Non-Persistent Trigger Signals



$EC(w-,1) = 10--$
$TC(w-,1) = --0-$
Trigger signals are not stable

Example: Non-Persistent Trigger Signals



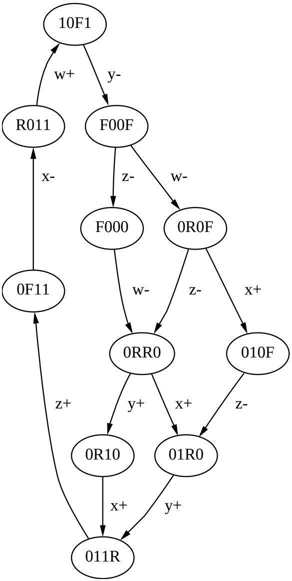
$EC(w-,1) = \texttt{10--}$

$TC(w-,1) = \texttt{--0-}$

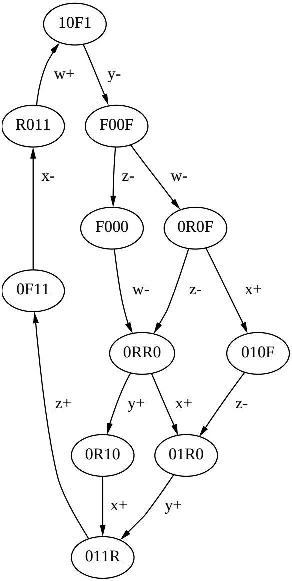
Trigger signals are not stable

No single cube cover exists

Example: Unresolvable Violations

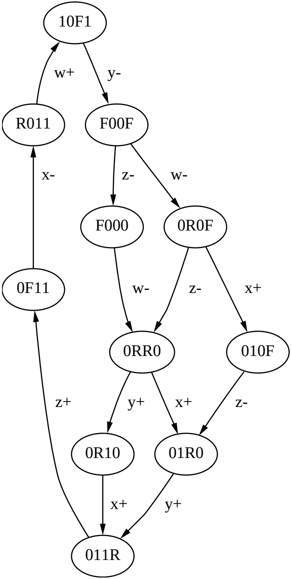


Example: Unresolvable Violations



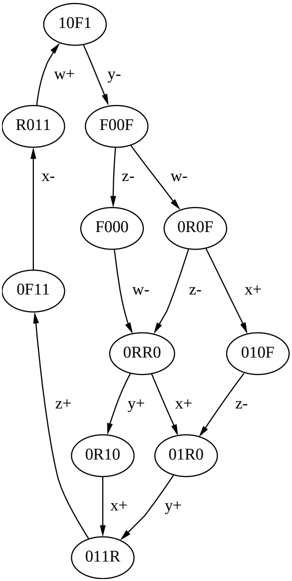
$EC(x+, 1) =$

Example: Unresolvable Violations



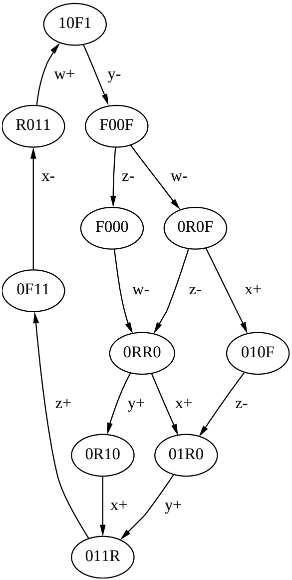
$EC(x+, 1) = 00--$

Example: Unresolvable Violations



$EC(x+,1) = 00--$
$TC(x+,1) =$

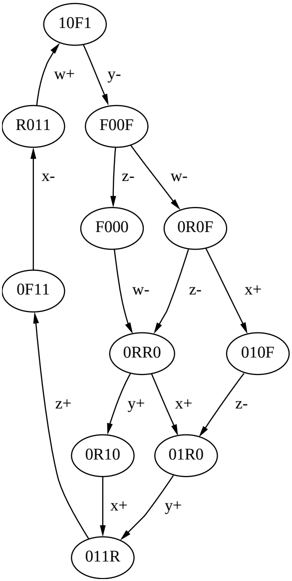
Example: Unresolvable Violations



$EC(x+, 1) = 00--$

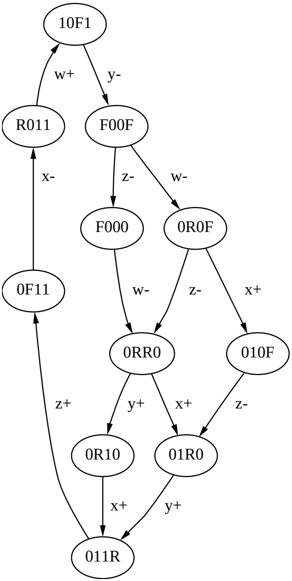
$TC(x+, 1) = 0---$

Example: Unresolvable Violations



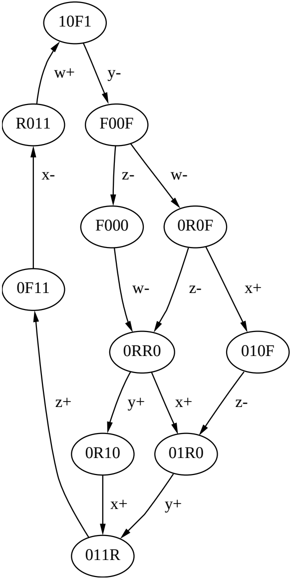
$$EC(x+,1) = \mathtt{00--}$$
$$TC(x+,1) = \mathtt{0---}$$
$$V(x+,1) =$$

Example: Unresolvable Violations



$$EC(x+,1) = \mathtt{00--}$$
$$TC(x+,1) = \mathtt{0---}$$
$$V(x+,1) = \{0F11, R011\}$$

Example: Unresolvable Violations

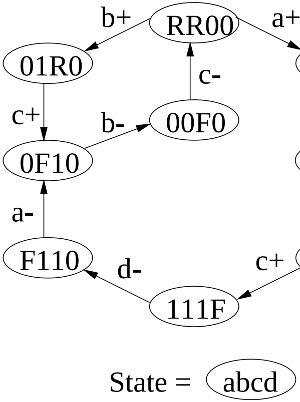


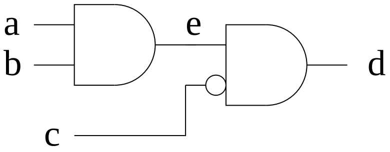
$EC(x+,1) = \texttt{00--}$
$TC(x+,1) = \texttt{0---}$
$V(x+,1) = \{0F11, R011\}$
No context signal to remove $R011$

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.

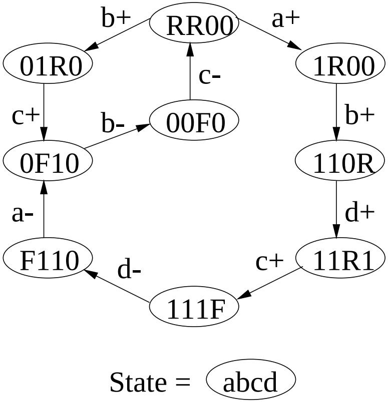
Example: Decomposition I

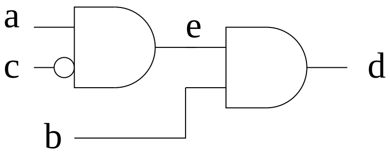




$$\langle F110\rangle \rightarrow \langle 0F10\rangle \rightarrow \langle 00F0\rangle \rightarrow \langle RR00\rangle$$

Example: Decomposition II

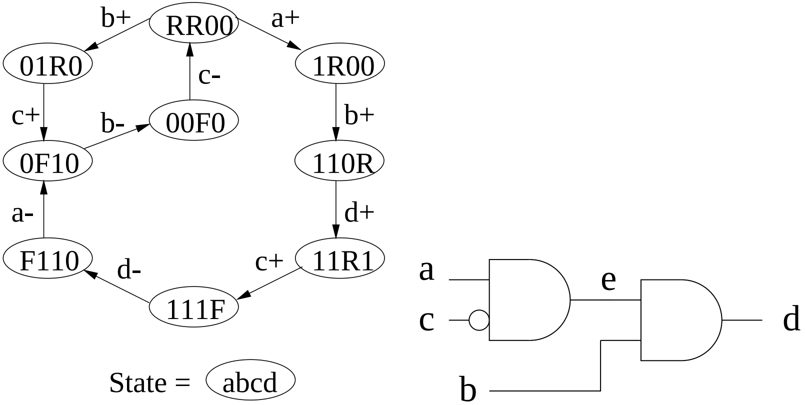




$$\langle F110 \rangle \rightarrow \langle 0F10 \rangle \rightarrow \langle 00F0 \rangle \rightarrow \langle RR00 \rangle$$

Hazard-Free Decomposition Overview

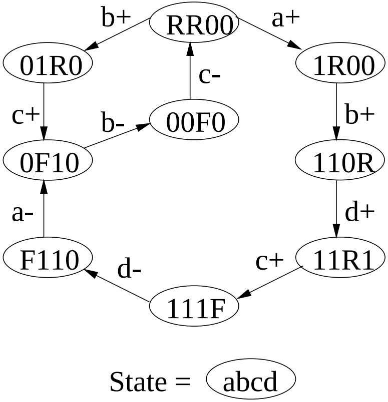
- Special care needed to guarantee a hazard-free decomposition.
- Need to find new internal signal that produces simpler circuit.
- Present here a simple technique for finding hazard-free decompositions using insertion points.

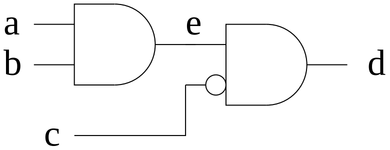


RR00
b+
a+
01R0
1R00
c-
c+
b+
b-
00F0
0F10
110R
a-
d+
F110
11R1
d-
c+
111F
State =
abcd
a
c
e
d
b

$$IP = ((\{a+\};\{d+\}),(\{c+\};\{d-\}))$$

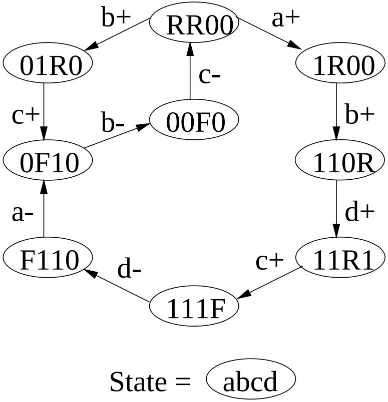
Example: Insertion Points

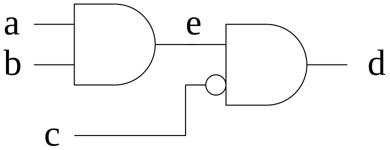




$$IP = ((\{b+\};\{d+\}),(\{a-\};\emptyset))$$

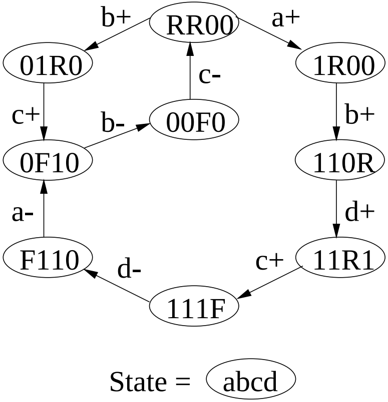
Example: Insertion Points

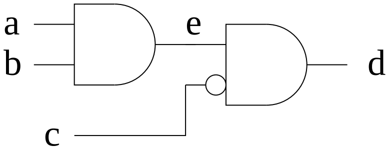




$$IP = ((\{b+\};\{d+\}),(\{a-\};\emptyset))$$

Cannot use $b-$ as end transition as it is an input





$$IP = ((\{b+\};\{d+\}),(\{a-\};\emptyset))$$

Cannot use $b-$ as end transition as it is an input

Using $c-$ makes e a three-input gate!

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., $\forall t \in t_e \;.\; t \notin T_I$)
  3. Start and end sets should only include concurrent transitions.
     (i.e., $\forall t_1, t_2 \in t_s \;.\; t_1 \parallel t_2$ and $\forall t_1, t_2 \in t_e \;.\; t_1 \parallel t_2$)

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.
  - 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.

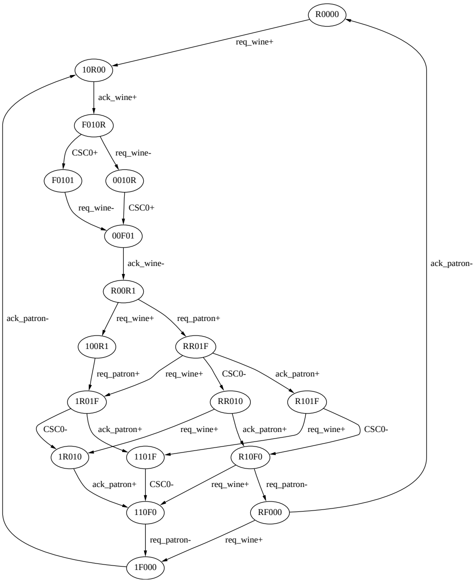
Transition Point Filters for Decomposition

- 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.

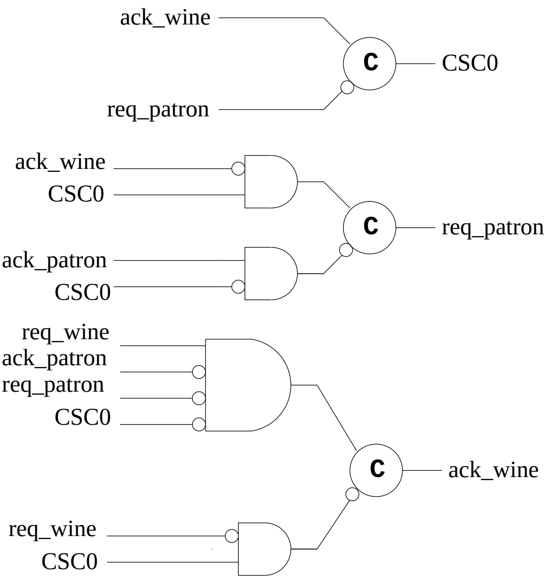
Algorithm for Decomposition

```
decomposition(SG, design, maxsize)
  HF = find_high_fanin_gates(design, maxsize);
  if (|HF| = 0) return design;
  best = |HF|; best_IP = design;
  TP = find_all_transition_points(SG, design, HF);
  foreach TP_R ∈ TP
    foreach TP_F ∈ TP
      if IP = (TP_R, TP_F) is legal then
        CSG = color_state_graph(SG, TP_R, TP_F);
        if (CSG is consistent) then
          SG' = insert_state_signal(SG, IP);
          design = synthesis(SG');
          HF = find_high_fanin_gates(design, maxsize);
          if ((|HF| < best) or ((|HF| = best) and
            (cost(design) < cost(best_IP)))) then
            best = |HF|; best_IP = design;
  design = decomposition(SG, design);
  return design;
```

Passive/Active Shop: State Graph



Passive/Active Shop: gC Circuit



Rising Transition Point Choices

Note that *req_wine* and *ack_patron* are trigger signals for *ack_wine*$+$.

Rising Transition Point Choices

Note that *req_wine* and *ack_patron* are trigger signals for *ack_wine*$+$.

Transition points using context signals:

Rising Transition Point Choices

Note that *req_wine* and *ack_patron* are trigger signals for *ack_wine*$+$.

Transition points using context signals:

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

Rising Transition Point Choices

Note that *req_wine* and *ack_patron* are trigger signals for *ack_wine*$+$.

Transition points using context signals:

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

Transition points using trigger signals:

Rising Transition Point Choices

Note that *req_wine* and *ack_patron* are trigger signals for *ack_wine*$+$.

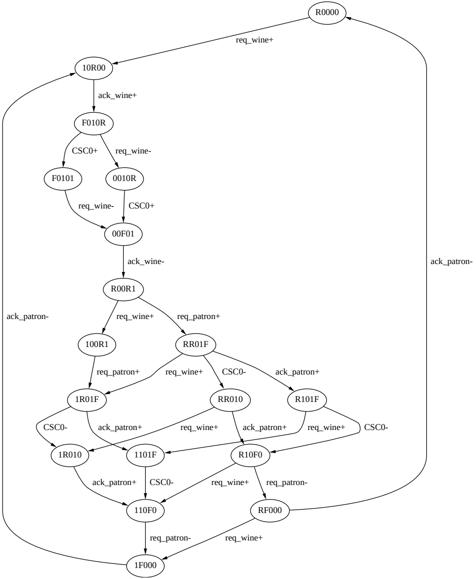
Transition points using context signals:

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

Transition points using trigger signals:

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

Passive/Active Shop: State Graph



Falling Transition Point Choices

Consider only ack_wine+, $CSC0+$, req_wine-, and ack_wine-

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$$(\{\textit{ack\_wine}+\}, \{\textit{CSC0}+\})$$

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$$(\{\mathit{ack\_wine}+\}, \{\mathit{CSC0}+\}) \qquad \text{OK}$$

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$$(\{\mathit{ack\_wine}+\},\{\mathit{CSC0}+\}) \qquad \text{OK}$$
$$(\{\mathit{ack\_wine}+\},\{\mathit{req\_wine}-\})$$

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{\mathit{ack\_wine}+\},\{\mathit{CSC0}+\})$	OK
$(\{\mathit{ack\_wine}+\},\{\mathit{req\_wine}-\})$	No, input

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{ack\_wine+\},\{CSC0+\})$	OK
$(\{ack\_wine+\},\{req\_wine-\})$	No, input
$(\{ack\_wine+\},\{ack\_wine-\})$	

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{ack\_wine+\}, \{CSC0+\})$	OK
$(\{ack\_wine+\}, \{req\_wine-\})$	No, input
$(\{ack\_wine+\}, \{ack\_wine-\})$	OK

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{ack\_wine+\},\{CSC0+\})$	OK
$(\{ack\_wine+\},\{req\_wine-\})$	No, input
$(\{ack\_wine+\},\{ack\_wine-\})$	OK
$(\{CSC0+\},\{req\_wine-\})$	

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{\mathit{ack\_wine}+\},\{\mathit{CSC0}+\})$	OK
$(\{\mathit{ack\_wine}+\},\{\mathit{req\_wine}-\})$	No, input
$(\{\mathit{ack\_wine}+\},\{\mathit{ack\_wine}-\})$	OK
$(\{\mathit{CSC0}+\},\{\mathit{req\_wine}-\})$	No, input

Falling Transition Point Choices

Consider only ack_wine+, $CSC0+$, req_wine-, and ack_wine-

$(\{ack\_wine+\}, \{CSC0+\})$	OK
$(\{ack\_wine+\}, \{req\_wine-\})$	No, input
$(\{ack\_wine+\}, \{ack\_wine-\})$	OK
$(\{CSC0+\}, \{req\_wine-\})$	No, input
$(\{CSC0+\}, \{ack\_wine-\})$	

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{ack\_wine+\},\{CSC0+\})$	OK
$(\{ack\_wine+\},\{req\_wine-\})$	No, input
$(\{ack\_wine+\},\{ack\_wine-\})$	OK
$(\{CSC0+\},\{req\_wine-\})$	No, input
$(\{CSC0+\},\{ack\_wine-\})$	OK

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{ack\_wine+\},\{CSC0+\})$	OK
$(\{ack\_wine+\},\{req\_wine-\})$	No, input
$(\{ack\_wine+\},\{ack\_wine-\})$	OK
$(\{CSC0+\},\{req\_wine-\})$	No, input
$(\{CSC0+\},\{ack\_wine-\})$	OK
$(\{req\_wine-\},\{CSC0+\})$	

Falling Transition Point Choices

Consider only $\mathit{ack\_wine}+$, $\mathit{CSC0}+$, $\mathit{req\_wine}-$, and $\mathit{ack\_wine}-$

$(\{\mathit{ack\_wine}+\}, \{\mathit{CSC0}+\})$	OK
$(\{\mathit{ack\_wine}+\}, \{\mathit{req\_wine}-\})$	No, input
$(\{\mathit{ack\_wine}+\}, \{\mathit{ack\_wine}-\})$	OK
$(\{\mathit{CSC0}+\}, \{\mathit{req\_wine}-\})$	No, input
$(\{\mathit{CSC0}+\}, \{\mathit{ack\_wine}-\})$	OK
$(\{\mathit{req\_wine}-\}, \{\mathit{CSC0}+\})$	OK

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{ack\_wine+\},\{CSC0+\})$	OK
$(\{ack\_wine+\},\{req\_wine-\})$	No, input
$(\{ack\_wine+\},\{ack\_wine-\})$	OK
$(\{CSC0+\},\{req\_wine-\})$	No, input
$(\{CSC0+\},\{ack\_wine-\})$	OK
$(\{req\_wine-\},\{CSC0+\})$	OK
$(\{req\_wine-\},\{ack\_wine-\})$	

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{ack\_wine+\}, \{CSC0+\})$	OK
$(\{ack\_wine+\}, \{req\_wine-\})$	No, input
$(\{ack\_wine+\}, \{ack\_wine-\})$	OK
$(\{CSC0+\}, \{req\_wine-\})$	No, input
$(\{CSC0+\}, \{ack\_wine-\})$	OK
$(\{req\_wine-\}, \{CSC0+\})$	OK
$(\{req\_wine-\}, \{ack\_wine-\})$	OK

Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

$(\{ack\_wine+\}, \{CSC0+\})$	OK
$(\{ack\_wine+\}, \{req\_wine-\})$	No, input
$(\{ack\_wine+\}, \{ack\_wine-\})$	OK
$(\{CSC0+\}, \{req\_wine-\})$	No, input
$(\{CSC0+\}, \{ack\_wine-\})$	OK
$(\{req\_wine-\}, \{CSC0+\})$	OK
$(\{req\_wine-\}, \{ack\_wine-\})$	OK
$(\{CSC0+, req\_wine-\}, \{ack\_wine-\})$	

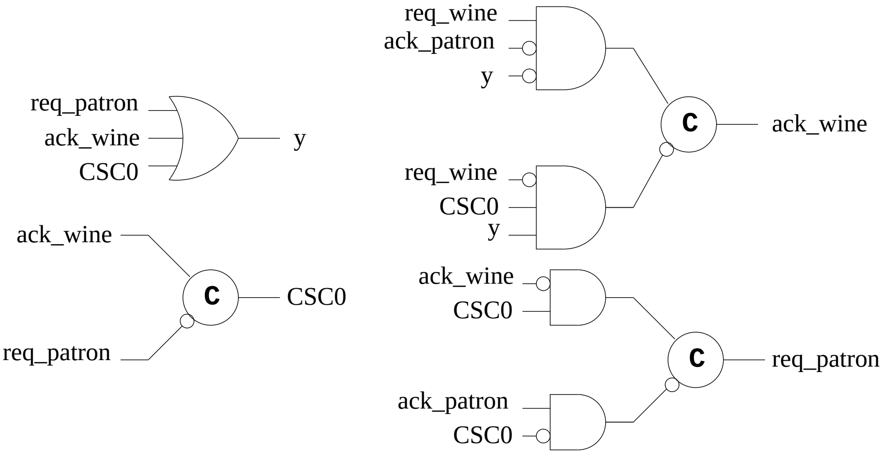
Falling Transition Point Choices

Consider only *ack_wine*$+$, *CSC0*$+$, *req_wine*$-$, and *ack_wine*$-$

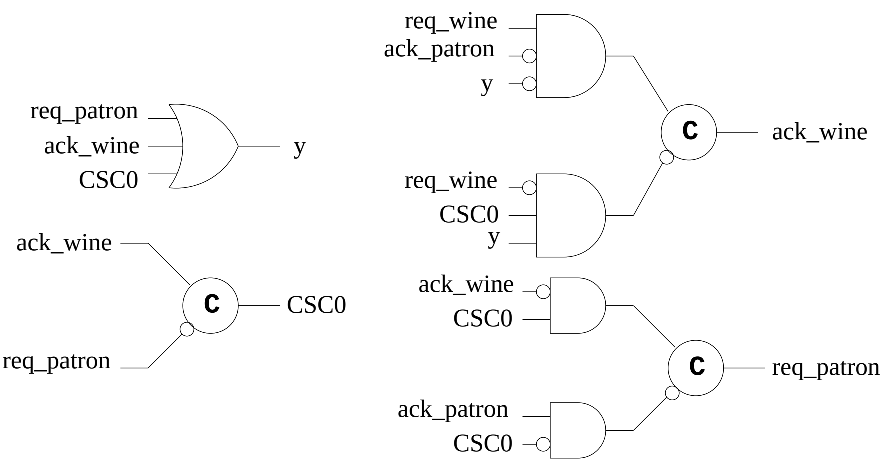
$(\{ack\_wine+\}, \{CSC0+\})$	OK
$(\{ack\_wine+\}, \{req\_wine-\})$	No, input
$(\{ack\_wine+\}, \{ack\_wine-\})$	OK
$(\{CSC0+\}, \{req\_wine-\})$	No, input
$(\{CSC0+\}, \{ack\_wine-\})$	OK
$(\{req\_wine-\}, \{CSC0+\})$	OK
$(\{req\_wine-\}, \{ack\_wine-\})$	OK
$(\{CSC0+, req\_wine-\}, \{ack\_wine-\})$	OK

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.

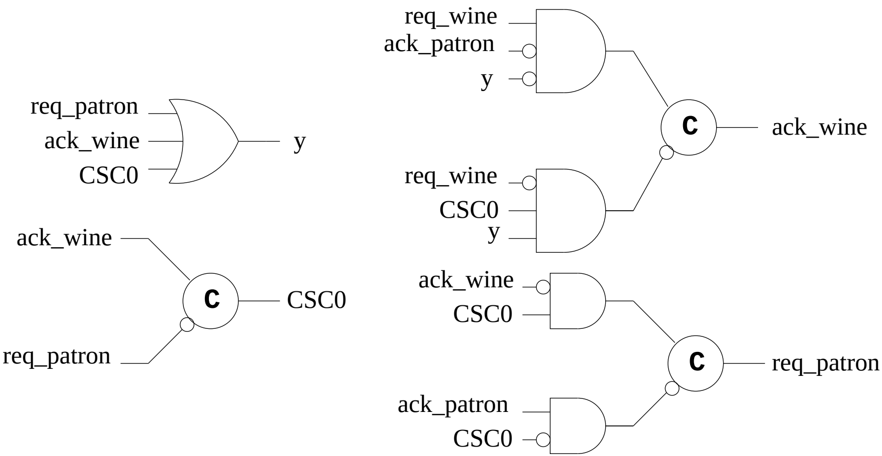

req_patron
ack_wine
CSC0
y
ack_wine
C
CSC0
req_patron
req_wine
ack_patron
y
C
ack_wine
req_wine
CSC0
y
ack_wine
CSC0
C
req_patron
ack_patron
CSC0

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



If bubble on *ack_patron* input to set AND gate for *ack_wine* is replaced with an inverter, this circuit is no longer hazard-free.

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



$\langle R00R1\rangle \rightarrow req\_patron+ \rightarrow \langle RR01F\rangle \rightarrow ack\_patron+ \rightarrow \langle R101F\rangle \rightarrow CSC0- \rightarrow \langle R10F0\rangle \rightarrow req\_patron- \rightarrow \langle RF000\rangle \rightarrow ack\_patron- \rightarrow \langle R0000\rangle$

Summary

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