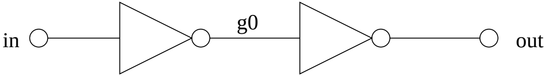
ECE570 Lecture 16: Diagnosis

Jeffrey Mark Siskind

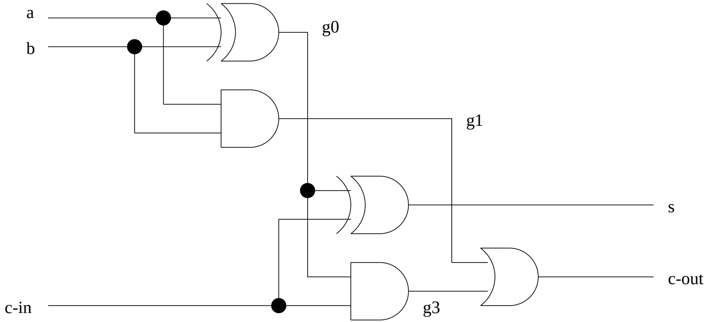
School of Electrical and Computer Engineering

Fall 2013

A Circuit



Another Circuit



No Fault Model

"Transistors can fall from the sky"

A faulty component of n inputs can behave like *any* Boolean function of n inputs

Component Models—No Fault Model

$$
\begin{aligned}
\textsc{Invert}(ab, x, out) &\triangleq \neg ab \rightarrow (out \leftrightarrow \neg x)\\
\textsc{And}(ab, x, y, out) &\triangleq \neg ab \rightarrow (out \leftrightarrow (x \wedge y))\\
\textsc{Or}(ab, x, y, out) &\triangleq \neg ab \rightarrow (out \leftrightarrow (x \vee y))\\
\textsc{Xor}(ab, x, y, out) &\triangleq \neg ab \rightarrow (out \leftrightarrow \neg(x \leftrightarrow y))
\end{aligned}
$$

Component Models—Stuck-At-Zero Fault Model

$$
\begin{aligned}
\textsc{Invert}(ab, x, out) &\triangleq [\neg ab \rightarrow (out \leftrightarrow \neg x)] \wedge [ab \rightarrow \neg out]\\
\textsc{And}(ab, x, y, out) &\triangleq [\neg ab \rightarrow (out \leftrightarrow (x \wedge y))] \wedge [ab \rightarrow \neg out]\\
\textsc{Or}(ab, x, y, out) &\triangleq [\neg ab \rightarrow (out \leftrightarrow (x \vee y))] \wedge [ab \rightarrow \neg out]\\
\textsc{Xor}(ab, x, y, out) &\triangleq [\neg ab \rightarrow (out \leftrightarrow \neg(x \leftrightarrow y))] \wedge [ab \rightarrow \neg out]
\end{aligned}
$$

Component Models in Scheme—I

```
(define (clauses p atomic-formulas)
 (all-values
  (let ((clause
         (map (lambda (atomic-formula)
               (list (a-boolean) atomic-formula))
              atomic-formulas)))
   (when (apply p (map first clause)) (fail))
   (map negate-literal clause))))
```

Component Models in Scheme—II

```
(define (clauses p atomic-formulas)
 (prime-implicates
  (all-values
   (let ((clause
          (map (lambda (atomic-formula)
                (list (a-boolean) atomic-formula))
               atomic-formulas)))
    (when (apply p (map first clause)) (fail))
    (map negate-literal clause)))))
```

Component Models in Scheme—III

```
(define (make-inverter fault-model ab x out)
 (clauses (lambda (ab x out)
           (case fault-model
            ((none) (implies (not ab) (eq? out (not x))))
            ((stuck-at-zero)
             (and (implies (not ab) (eq? out (not x)))
                  (implies ab (not out))))
            (else (panic "Unrecognized fault model"))))
          (list ab x out)))
```

Component Models in Scheme—IV

```
(define (make-and-gate fault-model ab x y out)
 (clauses (lambda (ab x y out)
           (case fault-model
            ((none) (implies (not ab) (eq? out (and x y))))
            ((stuck-at-zero)
             (and (implies (not ab) (eq? out (and x y)))
                  (implies ab (not out))))
            (else (panic "Unrecognized fault model"))))
          (list ab x y out)))
```

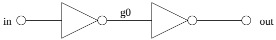
Component Models in Scheme—V

```
(define (make-or-gate fault-model ab x y out)
 (clauses (lambda (ab x y out)
           (case fault-model
            ((none) (implies (not ab) (eq? out (or x y))))
            ((stuck-at-zero)
             (and (implies (not ab) (eq? out (or x y)))
                  (implies ab (not out))))
            (else (panic "Unrecognized fault model"))))
          (list ab x y out)))
```

Component Models in Scheme—VI

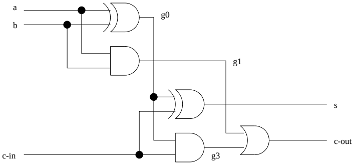
```
(define (make-xor-gate fault-model ab x y out)
 (clauses (lambda (ab x y out)
           (case fault-model
            ((none) (implies (not ab) (eq? out (not (eq? x y)))))
            ((stuck-at-zero)
             (and (implies (not ab) (eq? out (not (eq? x y))))
                  (implies ab (not out))))
            (else (panic "Unrecognized fault model"))))
          (list ab x y out)))
```

System Description—I



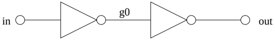
$$\textsc{Inverter}(ab(g_0), in, g_0) \wedge \textsc{Inverter}(ab(g_1), g_0, out)$$

System Description—II



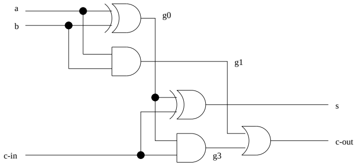
$$
\begin{array}{l}
\text{XOR}(ab(g_0), a, b, g_0)\wedge \\
\text{AND}(ab(g_1), a, b, g_1)\wedge \\
\text{XOR}(ab(g_2), g_0, c_{in}, s)\wedge \\
\text{AND}(ab(g_3), g_0, c_{in}, g_3)\wedge \\
\text{OR}(ab(g_4), g_3, c_{in}, c_{out})
\end{array}
$$

System Description in Scheme—I




```
(define (make-double-inverter fault-model test-vector)
 (append (make-inverter fault-model '(ab g0) 'in 'g0)
         (make-inverter fault-model '(ab g1) 'g0 'out)
         test-vector))
```

System Description in Scheme—II




```
(define (make-full-adder fault-model test-vector)
 (append (make-xor-gate fault-model '(ab g0) 'a 'b 'g0)
         (make-and-gate fault-model '(ab g1) 'a 'b 'g1)
         (make-xor-gate fault-model '(ab g2) 'g0 'c-in 's)
         (make-and-gate fault-model '(ab g3) 'g0 'c-in 'g3)
         (make-or-gate fault-model '(ab g4) 'g3 'g1 'c-out)
         test-vector))
```

Four Classes of Atomic Formulas

AB formulas: $ab(g_0), ab(g_1), ab(g_2), ab(g_3), ab(g_4)$

inputs: a, b, c_{in}

outputs: s, c_{out}

internal nodes: g_0, g_1, g_3

Vectors—I

A *vector* is a CNF formula where each clause contains a single literal.
(A *vector* is also a DNF formula that contains a single minterm.)
An *input literal* is a true or negated input.
An *output literal* is a true or negated output.
An *AB literal* is a true or negated AB formula.
An *input vector* is a vector that contains only input literals.
An *output vector* is a vector that contains only output literals.
A *test vector* is a vector that contains only input or output literals.
A *diagnosis* is a vector that contains only AB literals.

Vectors—II

An input vector is a (partial) specification of the inputs to a circuit.
An output vector is a (partial) specification of the outputs of a circuit.
A diagnosis is a (partial) specification of which components are operational and which are faulty.

Diagnosis

Let Σ be a system description.
Let i be an input vector.
Let o be an output vector.
Let t be a test vector.
Let d be a diagnosis.

simulation Given Σ, i, d, find o such that $\Sigma \cup \{i, d\} \models o$

inverse simulation Given Σ, o, d, find i such that $\Sigma \cup \{o, d\} \models i$

diagnosis Given Σ, i, o, find d such that $\Sigma \cup \{i, o\} \models d$
Given Σ, t, find d such that $\Sigma \cup \{t\} \models d$

General Problem

Let Σ contain both the system description and some vectors.
Find a vector Φ that contains atomic formulas of a given class such that $\Sigma \models \Phi$ and that Φ is not covered by some other Ψ such that $\Sigma \models \Psi$.

General Algorithm

1. find the set Π all of the prime implicates of Σ
2. remove from Π any clause that contains atomic formulas that are not of the desired class
3. find all of the prime implicants of Π

Terminology

A *minimal conflict* is a prime implicate that contains only AB literals.
A *kernel diagnosis* is a prime implicant of the set of all minimal conflicts.