Modified Condition/Decision Coverage (MC/DC) Tutorial

Kelly Hayhurst
NASA Langley Research Center
Phone: 757.864.6215
k.j.hayhurst@larc.nasa.gov

Dan Veerhusen
Rockwell Collins, Inc.
Phone: 319.295.4874
dsveerhu@rockwellcollins.com

MC/DC tutorial available through
http://shemesh.larc.nasa.gov/people/kjh/

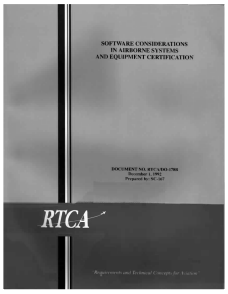
Objectives

- **Discuss the intent of Structural Coverage Analysis in the context of MC/DC**
- **Describe a method for evaluating requirements-based test cases for MC/DC**
- **Discuss the implications of short falls in Structural Coverage**

Coverage

Coverage is a measure -- not a test

- **Coverage is the extent to which a verification activity has satisfied its objectives**
  - **for testing, coverage can be used as an exit criteria**
- **DO-178B calls out 2 coverage measures**
  - **requirements coverage**
  - **software structure coverage**
    - **structural coverage**



Intent of Structural Coverage

Structural Coverage Analysis provides a means to confirm that the requirements-based tests exercised the code structure

- **The intent of structural coverage is to:**
  - **provide evidence that the code structure was verified to the degree required for the applicable software level**
  - **provide a means to support demonstration of absence of unintended functions**
  - **establish the thoroughness of requirements-based testing**

- from FAQ #43, DO-248B

Types of Structural Coverage

- Statement Coverage
- Decision Coverage
- Condition Coverage
- Condition/Decision Coverage
- Modified Condition/Decision Coverage
- Multiple Condition Coverage

Structural Coverage in DO-178B

Table A-7
Verification of Verification Process Results

	Objective		Applicability by SW Level			
	Description	Ref.	A	B	C	D
1	Test procedures are correct.	6.3.6b	○	○	○	
5	Test coverage of software structure (modified condition/decision) is achieved.	6.4.4.2	○			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.2a 6.4.4.2b	○	○		
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.2a 6.4.4.2b	○	○	○	

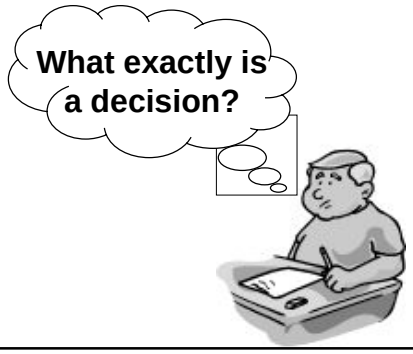
Statement & Decision Coverage

Statement Coverage:

Has every statement been invoked at least once?

Decision Coverage:

Has every decision taken on all possible outcomes at least once?



FAA National Software Conference - May 15, 2002 7

Defining Decision & Condition

Decision:

- **a Boolean expression composed of zero or more Boolean operators. A decision without a Boolean operator is a condition.**
- **if a condition appears more than once in a decision, each occurrence is a distinct condition**

Condition:

- **a Boolean expression containing no Boolean operators**
- **includes Boolean valued expressions with relational operators, such as, > , < , =**
  - **for example, x > y**

FAA National Software Conference - May 15, 2002 8

This is a Test

How many conditions are in the expression

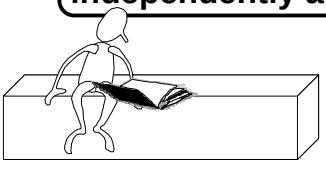
(A *and* B) *or* (B *and* C) *or* (A *and* C)?

Is Z:= ((x > y) or B); considered a decision?

Note: Certification Authorities Software Team (CAST) is working on a paper about decision coverage

MC/DC Description

- **every point of entry & exit in the program has been invoked at least once**
- **every condition in a decision in the program has taken all possible outcomes at least once**
- **every decision in the program has taken all possible outcomes at least once**
- **each condition in a decision has been shown to independently affect that decision's outcome**

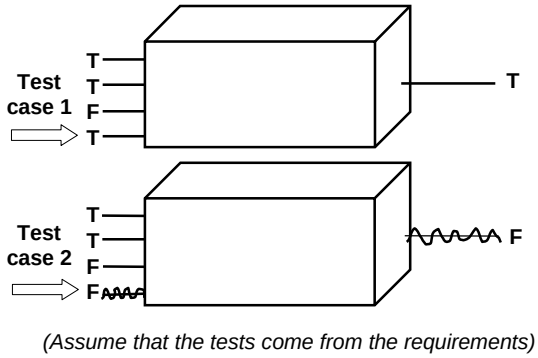


Independent Effect

- **A condition independently affects a decision's outcome if that condition *alone* determines the outcome of the decision**
- **Two methods for showing the independent effect of a condition are:**
  - **unique cause**
  - **masking**
- **Unique cause may be implied by the MC/DC description in the Glossary of DO-178B**

Unique Cause

- **A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions**



(Assume that the tests come from the requirements)

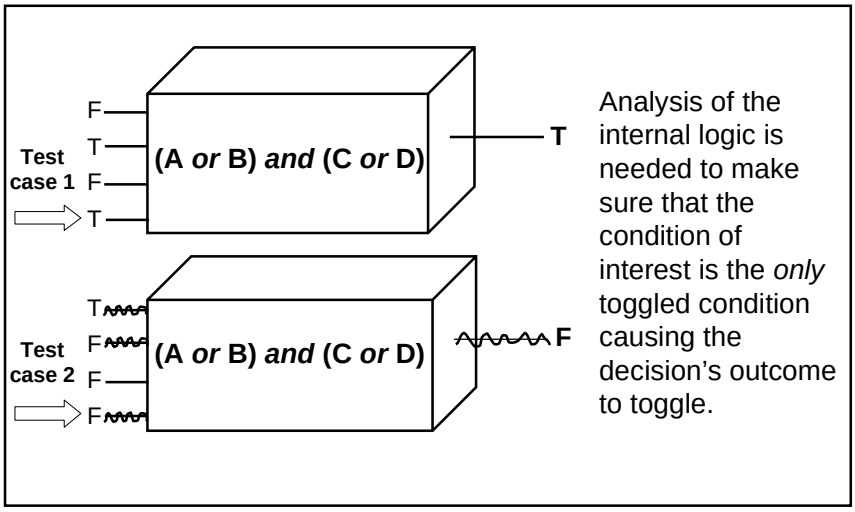
If the value of only one input toggles and the outcome toggles in response, then the cause is assumed to be the toggled input

- you don't need to see the internal logic of the expression to show independent effect

Masking

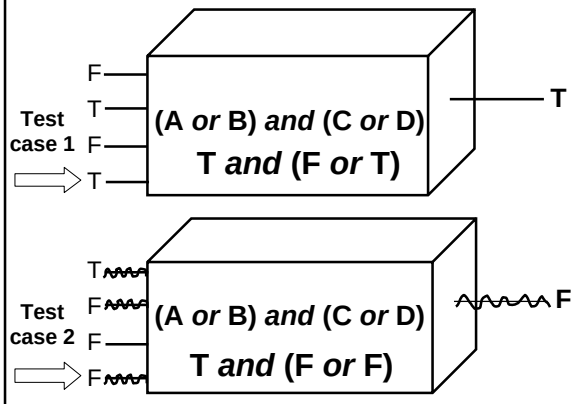
- **A condition is shown to independently affect a decision's outcome by using basic logic principles to assure that no other condition influences the outcome**
  - even though more than one condition may change value
- **In logical expressions, some inputs may hide or mask the effect of other inputs; for example,**
  - *false* ***and*** **X** is always *false*
  - *true* ***or*** **X** is always *true*
- **Masking principles are the converse**
  - *true* ***and*** **X** is **X**
  - *false* ***or*** **X** is **X**

Masking



Analysis of the internal logic is needed to make sure that the condition of interest is the *only* toggled condition causing the decision's outcome to toggle.

Masking (cont.)



Analysis of the internal logic is needed to make sure that the condition of interest is the *only* toggled condition causing the decision's outcome to toggle.

D is the only input that affects the outcome

Acceptability of Masking MC/DC

At the February 2001 meeting of the Certification Authorities Software Team (CAST), attendees concurred that masking MC/DC should be an acceptable means of meeting the MC/DC objective in DO-178B.

Position Paper CAST-6 titled "Rationale for Accepting Masking MC/DC in Certification Projects" has been published by CAST.

Masking Approach Overview

- **Defining Building Blocks for MC/DC**
  - **how to test basic logical constructs**
- **Using the Building Blocks for Decisions**
- **Building Block Approach to Evaluating MC/DC**
- **Examining Source Code**
  - **one line at a time**
  - **multiple lines**

Minimum Tests

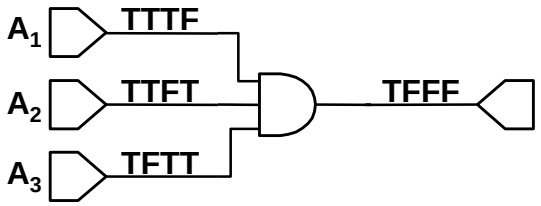
- **MC/DC is intended to assure that each condition within a decision has been shown to have the proper effect**
- **Showing independent effect of a condition requires specific minimum tests for each logical operator**
- **Minimum tests provide the building blocks for assessing MC/DC**

logical operator = logical gate

Testing an *n*-input *and* Gate

A_1 *and* A_2 *and* A_3 *and* … A_n

- **Minimum testing to provide MC/DC requires**
  - **all inputs *true*, output *true***
  - **each input individually *false*, output *false***
- **Example: 3-input *and* gate: TTT, TTF, TFT, FTT**

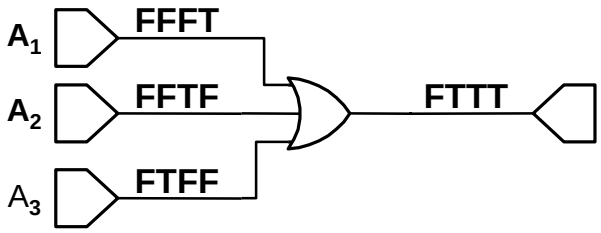


A_1 *and* A_2 *and* A_3

Testing an *n*-input *or* Gate

A_1 *or* A_2 *or* … A_n

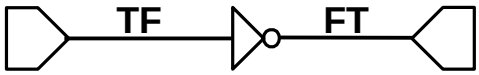
- **Minimum testing to provide MC/DC requires**
  - **all inputs *false*, output *false***
  - **each input individually *true*, output *true***
- **Example: 3-input *or* gate: FFF, FFT, FTF, TFF**



A_1 *or* A_2 *or* A_3

Testing a *not* Gate

- **Minimum testing to provide MC/DC requires**
  - **input *true*, output *false***
  - **input *false*, output *true***
- **Example:**



***not* A**

Testing an *xor* Gate

It's not like the other gates ...

- **More than one test set will satisfy the MC/DC criteria for an *xor* gate**
- **Minimum testing to provide MC/DC requires**
  - **any of the following for a 2-input *xor***
    - **TT, TF, FT**
    - **TF, FT, FF**
    - **FT, FF, TT**
    - **FF, TT, TF**

Question

Situation:

The software requirements call for evaluating the expression A *xor* B

The requirement is incorrectly implemented in source code as A *or* B

What test case is needed to catch the coding error?

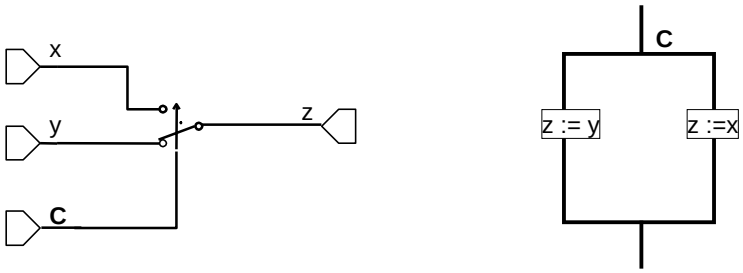
What do the Minimum Tests Provide?

- **For decisions with a common logical operator (e.g., A *and* B *and* C *and* …), the minimum tests guarantee that**
  - **the decision has taken all possible outcomes at least once**
  - **every condition has taken all possible outcomes at least once**
  - **every condition independently affects the decision outcome**

Testing an *if-then-else*

- **A Boolean expression controls the execution flow of an *if-then-else* statement**

Example: if C then z := x else z := y;



Minimum Testing for an *if-then-else*

(1) Inputs that force the execution of the *then* path

(2) Inputs that force the execution of the *else* path

- **Note that the decision must evaluate to *false* with confirmation that the *then* path did not execute, even if there is no *else* path.**

(3) Inputs to exercise any logical gates in the decision using the minimum tests

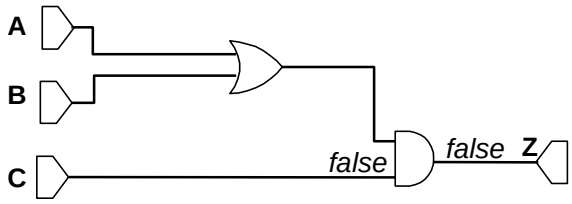
Applying the building blocks to determine whether a set of requirements-based test cases provide MC/DC of the source code

Evaluating MC/DC

- **For decisions with a common logical operator (A *or* B *or* C) or (A *and* B *and* C)**
  - **evaluating MC/DC requires checking the requirements-based tests to make sure they contain the minimum tests for that operator**
- **For decisions with mixed logical operators (A *or* B) *and* (C *or* D)**
  - **evaluating MC/DC is a bit more complicated**
  - **complications arise because one input to a logical operator may mask the effects of other inputs to that operator**

A Closer Look at Masking

Z:= (A or B) and C;



- ***false and*** X **is always *false***
- ***true or*** X **is always *true***

If you can't "see" the output of a gate for a particular test case, then that test case does not count towards coverage of that gate

Controllability & Observability

- **Basic concepts of testing logic circuits:**
  - **<u>controllability</u>: ability to control the inputs to a logical operator**
  - **<u>observability</u>: ability to observe the outputs of a logical operator at some end point**

5-Step Approach to Evaluating MC/DC

(1) Create a schematic representation of the source code

(2) Map the inputs of the requirements-based test cases to the schematic representation

(3) Eliminate masked test cases
- those cases where the results for a specific gate are hidden from the observed outcome

(4) Determine MC/DC based on the building blocks for each logical operator

(5) Examine the outputs of the tests to confirm correct operation of the software

First, we will look at assessing MC/DC for a single line of source code.

Second, we will look at assessing MC/DC for multiple lines of source code.

Single Source Line Example

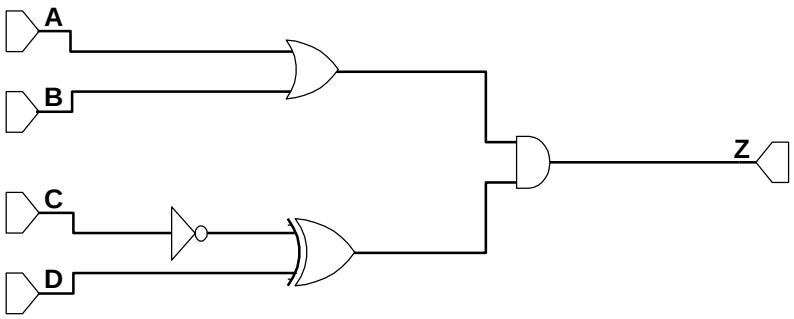
Z:= (A or B) and (not C xor D);

Requirements-based Test Cases

Test Case Number	1	2	3	4	5
A	T	F	F	T	T
B	F	F	T	T	T
C	T	T	T	F	F
D	T	T	T	T	F
Z	T	F	T	F	T

FAA National Software Conference - May 15, 2002 33

Step 1: Source Code Representation



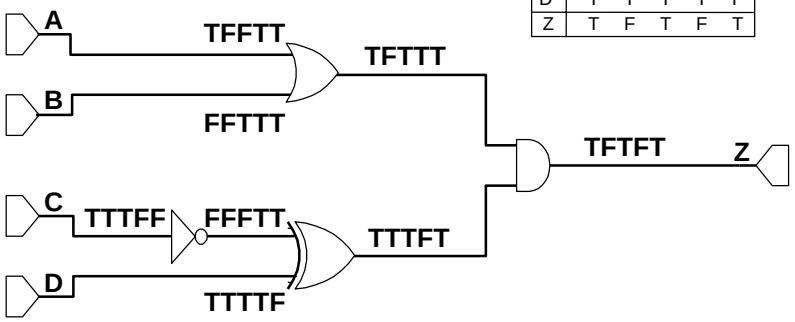
Z:= (A or B) and (not C xor D);

FAA National Software Conference - May 15, 2002 34

Step 2: Map Test Cases

Test Cases

	1	2	3	4	5
A	T	F	F	T	T
B	F	F	T	T	T
C	T	T	T	F	F
D	T	T	T	T	F
Z	T	F	T	F	T



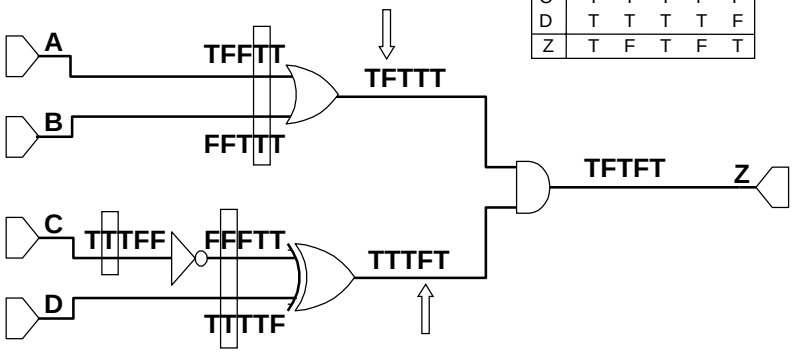
Z:= (A or B) and (not C xor D);

FAA National Software Conference - May 15, 2002 35

Step 3: Eliminate Masked Tests

Test Cases

	1	2	3	4	5
A	T	F	F	T	T
B	F	F	T	T	T
C	T	T	T	F	F
D	T	T	T	T	F
Z	T	F	T	F	T



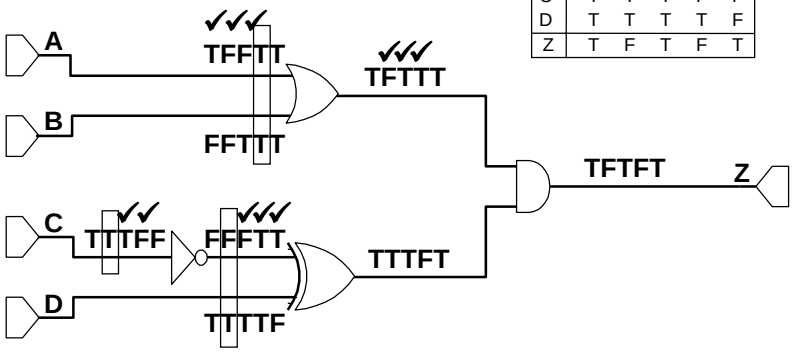
Z:= (A or B) and (not C xor D);

FAA National Software Conference - May 15, 2002 36

Step 4: Check for Minimum Tests

Test Cases

	1	2	3	4	5
A	T	F	F	T	T
B	F	F	T	T	T
C	T	T	T	F	F
D	T	T	T	T	F
Z	T	F	T	F	T

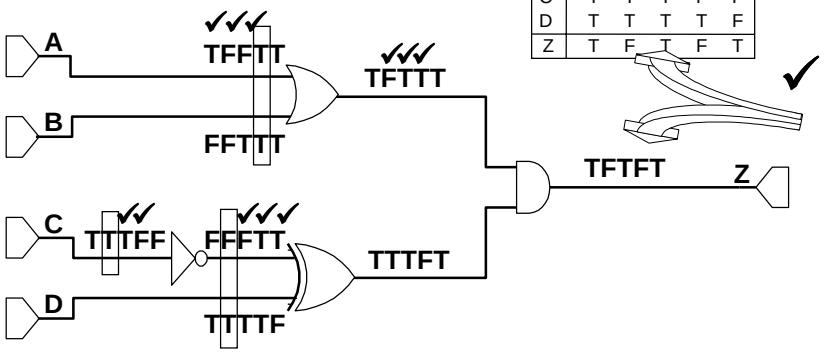


Z:= (A or B) and (not C xor D);

Step 5: Confirm Final Results

Test Cases

	1	2	3	4	5
A	T	F	F	T	T
B	F	F	T	T	T
C	T	T	T	F	F
D	T	T	T	T	F
Z	T	F	T	F	T



Z:= (A or B) and (not C xor D);

Multiple Source Lines Example

Requirement: Perform a voting operation on three input Booleans where the output is to be *true* whenever at least two of the inputs are *true*.

Source Code:

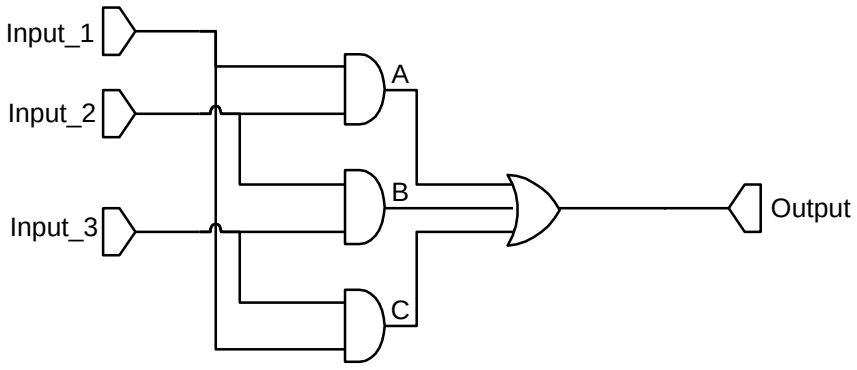
```
A := Input_1 and Input_2;
B := Input_2 and Input_3;
C := Input_3 and Input_1;
Output := A or B or C;
```

Requirements-based Test Cases

Test Case Number	1	2	3	4
Input_1	T	T	F	F
Input_2	T	F	T	F
Input _3	F	T	T	F
Output	T	T	T	F

FAA National Software Conference - May 15, 2002 39

Step 1: Source Code Representation

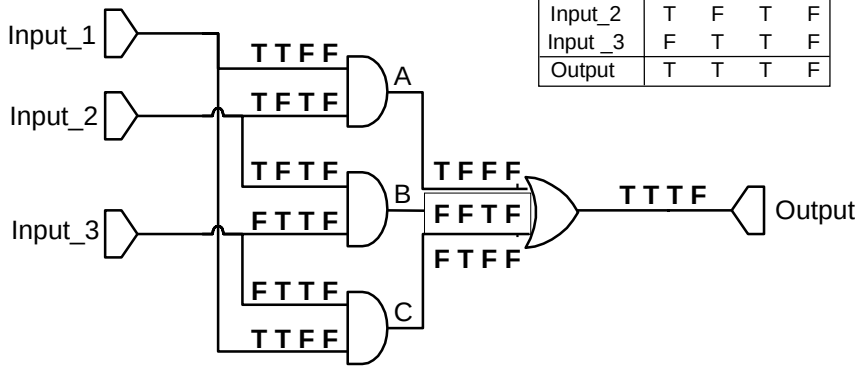


FAA National Software Conference - May 15, 2002 40

Step 2: Map Test Cases

Test Cases

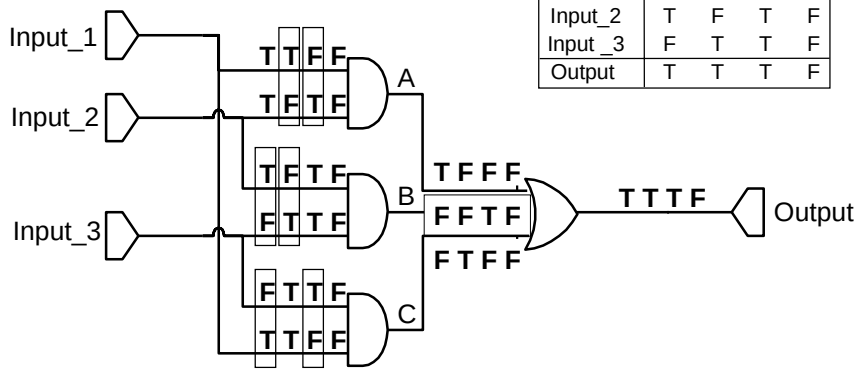
	1	2	3	4
Input_1	T	T	F	F
Input_2	T	F	T	F
Input _3	F	T	T	F
Output	T	T	T	F



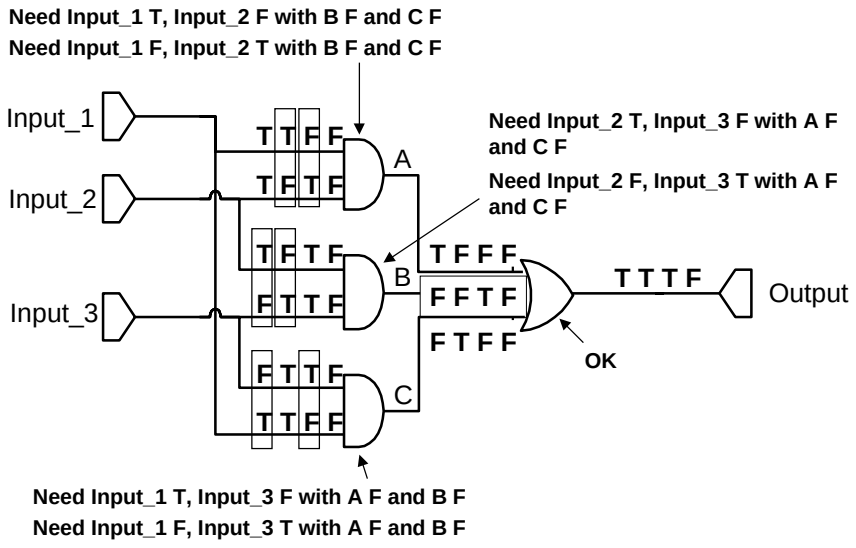
Step 3: Eliminate Masked Tests

Test Cases

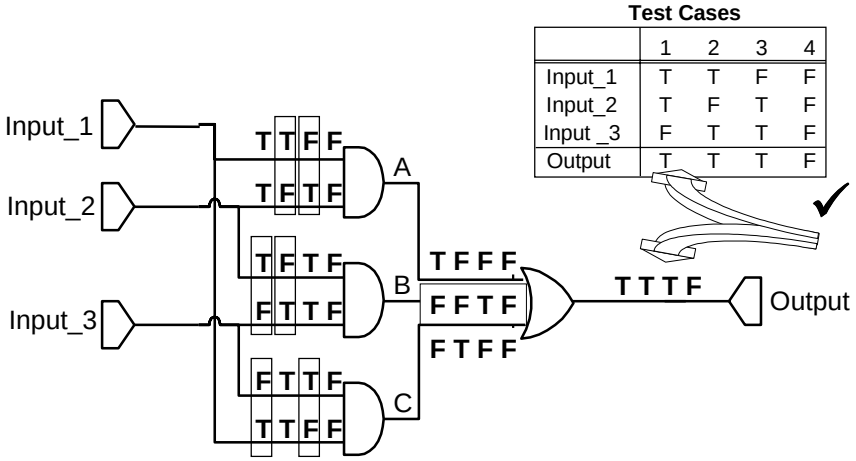
	1	2	3	4
Input_1	T	T	F	F
Input_2	T	F	T	F
Input _3	F	T	T	F
Output	T	T	T	F



Step 4: Check for Minimum Tests



Step 5: Confirm Results



Analysis Resolution

- **Coverage analysis can reveal that the code structure was not exercised sufficiently by the requirements-based test cases**
  - **inadequate requirements-based tests or procedures**
  - **inadequate software requirements**
  - **dead or deactivated code**
- **Section 6.4.4.3 of DO-178B provides guidance for each of these**

Analysis Resolution (cont.)

- **Coverage analysis may also identify errors in the source code**
  - **there may be an error even if the actual results match the expected results**

Example:

Requirement: **A and (B xor C)**
Requirements-based Test Cases

Test Case Number	1	2	3	4
A	F	T	T	T
B	F	F	T	T
C	T	F	F	T
Output	F	F	T	F

Source Code: **B and (B xor C)**

- **Expected results will match the actual results**
- **MC/DC analysis will show that the *xor* gate is not adequately tested**
  - **further analysis will show the mismatch between requirements and code**

Summary

- Structural coverage and its role in DO-178B
- MC/DC and its nuances
- Building blocks for coverage
- Controllability & observability
- 5 step approach to MC/DC
- Analysis resolution

Additional Information on MC/DC

- 2 Reports available through the FAA software site http://av-info.faa.gov/software/Research.htm
  - A Practical Tutorial on Modified Condition/ Decision Coverage, by Hayhurst, Veerhusen, Chilenski, & Rierson, NASA/TM-2001-210876, May 2001
  - An Investigation of Three Forms of the Modified Condition Decision Coverage (MCDC) Criterion, by John Chilenski, DOT/FAA/AR-01/18, April 2001
- DO-178B and DO-248B
- CAST Position Paper CAST-6, Rationale for Accepting Masking MC/DC in Certification Projects
  - http://av-info.faa.gov/software/CAST_frames.htm