CE653 – Asynchronous Circuit Design

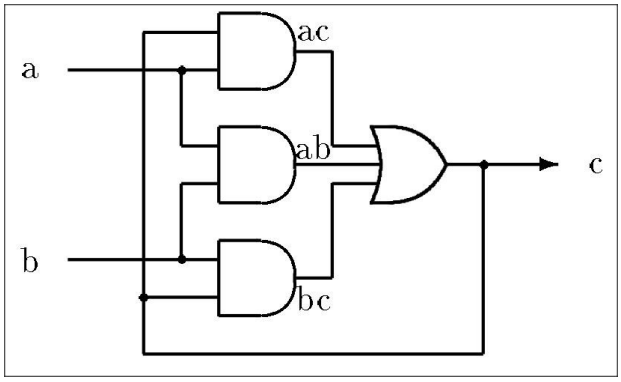
Instructor: C. Sotiriou

http://inf-server.inf.uth.gr/courses/CE653/

Contents

- Eichelberger's paper examples
- Abramovic's Book
- Methods for Hazard/Race Detection
  - Combinational Circuits
  - Sequential Circuits

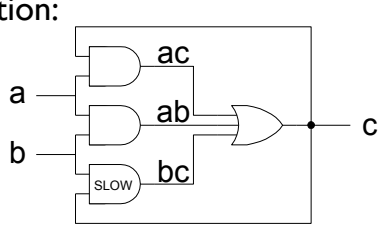
Motivation - C-Element Hazard Example

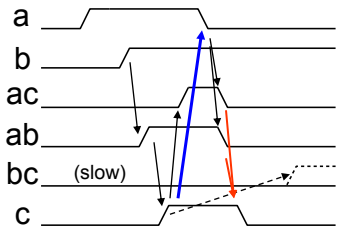


- Consider:
  1. (a, b) = 11 ➔ ab = 1 ➔ c = 1 (before ac, bc = 1)
  2. (a, b) = 10 ➔ ab = 0 ➔ c = 0, ac = 1 ➔ c = 1 (static 1 hazard)

Motivation - C-Element Hazard Example

- Hazard Animation:



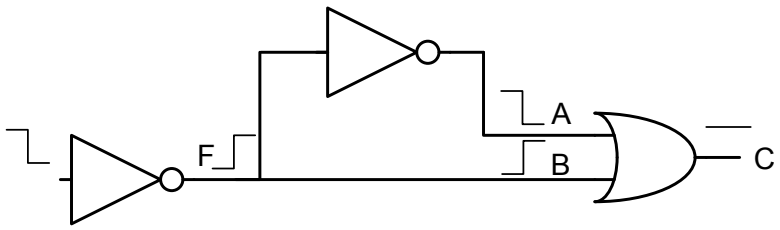


Asynchronous Circuits - Classes

- **Dimension 1: Delay Model**
  - Measure of robustness of control to variations in delays of gates and wires
  - Assumption about delays of gates necessary to ensure design works as dictated by specification
    - Most robust = Arbitrary gate and wire delay
      - Design will work as specified even if delays are random(0,infinity)
      - Larger delays just means control is slower
    - Least robust = Bounded delay on gates and wires
      - If delays are outside these bounds, glitches may occur at outputs or output simply may not transition as expected
- **Dimension 2: Environmental Model**
  - Essentially these are assumptions/restrictions on how fast environment can be for circuit to work

QDI Model – Isochronic Fork

- **Isochronic fork**
- If fork at F is isochronic
  - can assume B fires high before A
  - translates to relative timing assumption about long and short paths

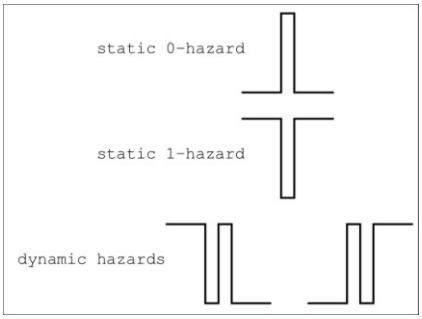


Asynchronous Circuits - Classes

- Timing Model (or Class) is used to define specific timing assumptions with respect to correct circuit operation
  - DI
    - Arbitrary gate and wire delays (unbounded)
  - QDI
    - DI except for Isochronic Forks
      - No need to acknowledge fanouts
  - SI (or Muller) circuits
    - Arbitrary gate delays, bounded wire delays
    - Closed system implementation (gate + environment)
  - Fundamental Mode (Huffman) circuits
    - "Fundamental Mode" Operation:
    - **Outputs and State (local) stabilise before new input change**

Hazard Types

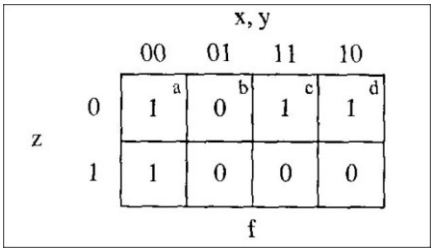
- Static 0 or 1
- Function Hazard
  - f(A) = f(B), where:
  - A = (a_1,…, a_p, a_{p+1}, …, a_n)
  - B = (a_1',…, a_p', a_{p+1}, …, a_n)
  - A➔B input vector transition contains 0's and 1's in function cubes



- Logic Hazard
  - Combinational Network static hazard caused by gate delays
- Dynamic
  - Static + Output Change

Function Hazard Example

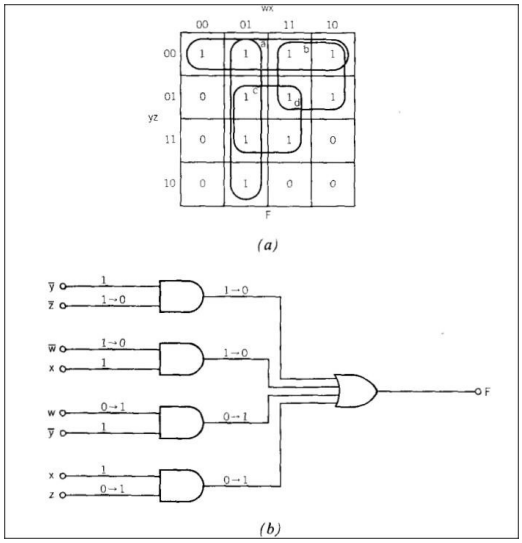
- Consider input transitions:
  000→110 for function f



- Function f contains potential function hazards:
  - for input changes between minterms
    a→c, a→d, etc. – static 1
  - for input changes between othrer minterms,
    e.g. 010→111 – static 0

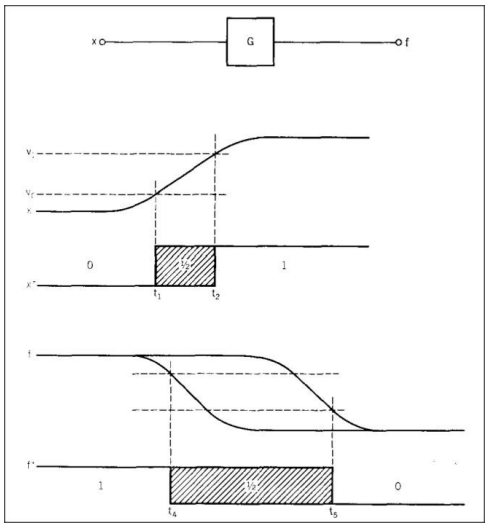
Logic Network Example

- Logic Networks may be free of function hazards but not of logic hazards
- Consider transitions between minterms:
  - a→d, d→a, c→b, b→c
- **Theorem:**
  A 2-level SOP function f is free of logic hazards, iff it contains all Primes (PIs) of f.



Ternary Approximation to Binary Signals

- A third value, **1/2**, or **X**, may be used to signify the transitive state of a signal
- 3-Valued Algebra may be used to detect and eliminate hazards



Binary and Ternary XOR gate Truth Table

y \ x	0	1
0	0 [a]	1 [b]
1	1 [c]	0 [d]

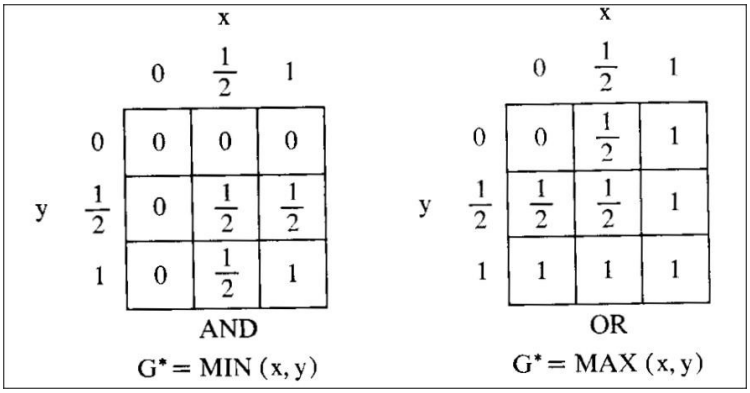
G

y \ x	0	1/2	1
0	0 [i]	1/2 [j]	1
1/2	1/2	1/2 [k]	1/2
1	1	1/2	0

G*

- Original truth table of logic function used to determine ternary truth table, whereas **½ = 0 OR 1**, e.g. for XOR:
  - 0 (+) ½ = 0 (+) (0 or 1) =1/2, as 0 (+) 0 and 0 (+) 1 produce different outputs
  - ½ (+) ½ = (0 or 1) (+) (0 or 1) = ½
- if p of n inputs are ½, output is 0 or 1 if all 2^p output entries agree

Ternary Truth tables for AND/OR



AND, $G^* = MIN(x, y)$	x = 0	x = 1/2	x = 1
y = 0	0	0	0
y = 1/2	0	1/2	1/2
y = 1	0	1/2	1

OR, $G^* = MAX(x, y)$	x = 0	x = 1/2	x = 1
y = 0	0	1/2	1
y = 1/2	1/2	1/2	1
y = 1	1	1	1

- AND, OR truth tables easier to derive due to their controlling values:
  - 0 . ½ = 0, 1 + ½ = 1

Ternary Gate Functions Properties

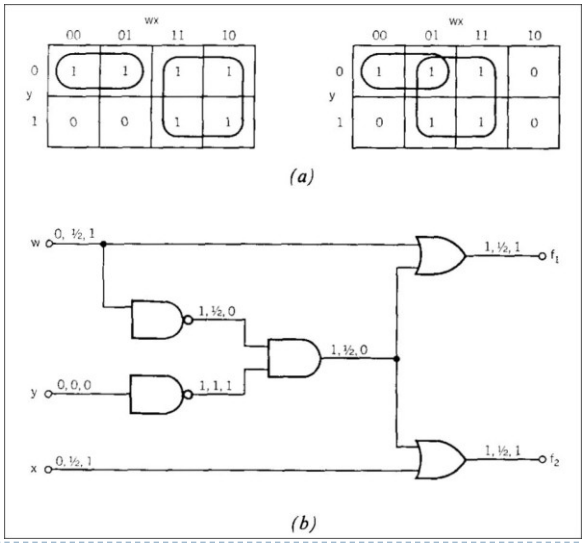
- Property 1:
  - If one or more ternary gate or logic network inputs are changed 1→1/2, or 0→1/2, the ternary outputs will either remain unchanged or change to 1/2
- Property 2:
  - If one or more ternary gate or logic network inputs are changed 1/2→1, or 1/2 → 0, the ternary outputs will either remain unchanged or change to 0 or 1
- Proof:
  - E.B. Eichelberger – *Hazard Detection in Combinational and Sequential Switching Circuits*, IBM Journal, 1965.

Hazard Detection using Ternary Algebra

- Theorem 1:
  - A combinational logic network contains a hazard for an input vector transition from A to B, where:
    - A = (a_1, …, a_p, a_{p+1}, …, a_n)
    - B = (a_1, …, a_v, a_{p+1}, …, a_n)
    - A/B = (1/2, …, 1/2, a_{p+1}, …, a_n)
  - iff (if and only if)
    1. **f(A) = f(B) ≠1/2**
    2. **f(A/B) = 1/2**
- Proof:
  - Based on previous properties

Hazard Detection using Ternary Algebra

- For functions f1, f2
- Consider input change:
  - w′x′y′ → wxy′
- is a hazard produced for f1, f2?
- Must determine:
  - f(w′, x′, y′), f(½ , ½, y′), f(w, x, y′)
- Outcome:
  - (1, ½, 1)
  - f1, f2 both contain a hazard



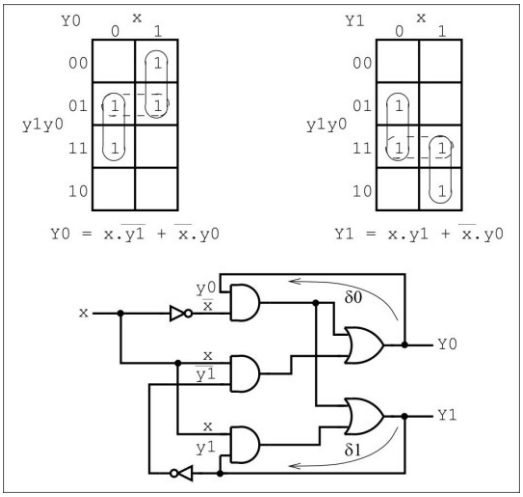
Additional Hazards in Sequential Circuits

- Critical Race
  - If order of changes in state variables affects final state, race is critical
  - 11 → 10 → 00, or, 11→ 01 → 00
- Essential Hazard
  - Critical race between input and feedback change – must add delay to fix
  - Property of FSM specification
- Essential Hazard Detection:
  - for input vector A→B,
  - If single change (A, B) produces different states and output to three change times (A→B, A, B) circuit contains an **Essential Hazard**

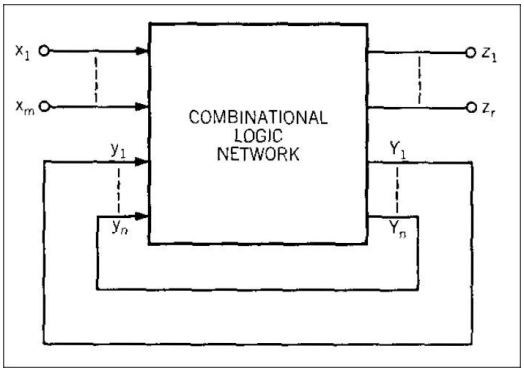
Essential Hazard Example

	x	
Y1Y0	**0**	**1**
1 (00)	1, 0	2, 0
2 (01)	3, 0	2, 0
3 (11)	3, 0	4,0
4 (10)	1,1	4,1

- δ0, δ1 are feedback delays for state signals `Y1, Y0`
  - `Y1, Y0` → `y1, y0`



Sequential Hazard Analysis



- Sequential Circuit Model:
  - Input vector ($x_1 \ldots x_m$) changes
  - Next State signals ($Y_1 \ldots Y_n$) change as a response to input change
  - Current State signals ($y_1 \ldots y_n$) change as a reponse to next state change

Sequential Hazard Analysis

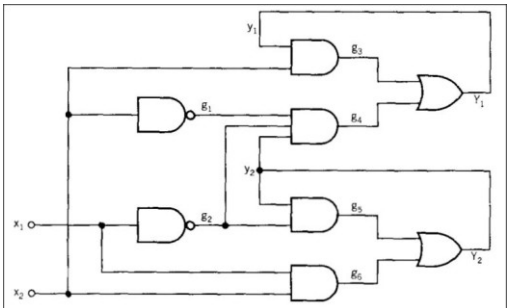
- Eichelberger's Two-Step Approach:
  - Procedure/Step 1 – Determine all ***changing*** Y signals
    1. Set changing input vector signals to intermediate 1/2 values, and all other x or y signals to their previous values
    2. Evaluate Y_i functions to determine changes from 1 or 0 to 1/2
    3. Propagate any 1/2 Y_i change to corresponding y_i change and repeat process until no further changes to Y_i occur
  - Procedure/Step 2 – Determine all ***stabilising*** Y signals
    - Set changing input vector signals to their final values, 1 or 0, and all other x or y signals to their previous values, as determined by Procedure 1
    - Evaluate Y_i functions to determine changes from 1/2 to 1 or 0
    - Propagate any 0 or 1 Y_i change to corresponding y_i change and repeat process until no further changes to Y_i occur

Sequential Hazard Analysis

- Theorem 2:
  - If Y_k = 1(0) after applying Procedures A and B to a sequential circuit for a given input change starting from a given internal state, then the Y_k signal must stabilise at 1(0) for this transition, regardless of the values of the finite delays of the logic gates
- Proof:
  - Based on previous Theorem (Theorem 1)

Sequential Hazard Analysis - Example

- Determine whether 00→11 change in x_1, x_2 inputs results in indeterminate final state



(a)

	x_1	x_2	y_1	y_2	g_1	g_2	g_3	g_4	g_5	g_6	Y_1^*	Y_2^*
1	0	0	0	0	1	1	0	0	0	0	0	0
2	½	½	0	0	½	½	0	0	0	½	0	½
3	½	½	0	½	½	½	0	½	½	½	½	½
4	1	1	½	½	0	0	½	0	½	1	½	1
5	1	1	½	1	0	0	½	0	1	1	½	1

(b)

Ternary Simulation Characteristics

- For n feedback lines, at most 2 x n evaluations are required
- Hazards and Races are detected automatically

- ***Optimisations***
  - During Procedure A, any gate with output at 1/2 need not be further considered, since output cannot change further
  - During Procedure B, any gate with output different from 1/2 need not be further considered, since again its output cannot change further
    - In both cases remove gate from simulation queue

From Ternary to 13-Value Logic for Hazard Detection

13-Value Logic

- **Goal:**
  - Separate hazards (1/2 value is X here) from transitions
  - Previous algorithms are often very pessimistic for many types of asynchronous circuits
- Each triplet represents transition from a signal state to another and the intermediate state, e.g. <1, v, 0>

Constant: $< 1, 1, 1 >, < 0, 0, 0 >$

Transition: $< 0, \uparrow, 1 >, < 1, \downarrow, 0 >$

Hazard: $< 0, X, 0 >, < 0, X, 1 >, < 1, X, 0 >, < 1, X, 1 >$

Stabilizing: $< X, X, 0 >, < X, X, 1 >$

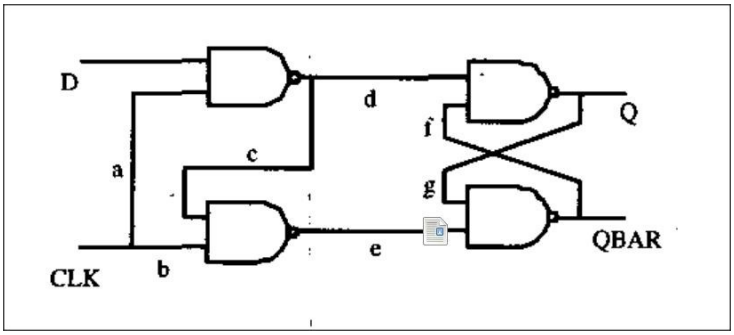
Destabilizing: $< 0, X, X >, < 1, X, X >$

Undefined: $< X, X, X >$

13-Value AND Gate Truth Table

	<0, X, 0>	<0, X, 1>	<0, X, X>	<1, X, 0>	<1, X, 1>	<1, X, X>	<X, X, 0>	<X, X, 1>	<X, X, X>	<0, 0, 0>	<0, ^, 1>	<1, 1, 1>	<1, v, 0>
<0, X, 0>	<0, X, 0>	<0, X, 0>	<0, X, 0>	<0, X, 0>	<0, X, 0>	<0, X, 0>	<0, X, 0>	<0,X, 0>	<0, X, 0>	<0, 0, 0>	<0, X, 0>	<0, X, 0>	<0, X, 0>
<0, X, 1>	<0, X, 0>	<0, X, 1>	<0, X, X>	<0, X, 0>	<0, X, 1>	<0, X, X>	<0, X, 0>	<0, X, 1>	<0, X, X>	<0, 0, 0>	<0, X, 1>	<0, X, 1>	<0, X, 0>
<0, X, X>	<0, X, 0>	<0, X, X>	<0, X, X>	<0, X, 0>	<0, X, X>	<0, X, X>	<0, X, 0>	<0, X, X>	<0, X, X>	<0, 0, 0>	<0, X, X>	<0, X, X>	<0, X, 0>
<1, X, 0>	<0, X, 0>	<0, X, 0>	<0,X, 0>	<1, X, 0>	<1, X, 0>	<1, X, 0>	<X, X, 0>	<X, X, 0>	<X, X, 0>	<0, 0, 0>	<0, X, 0>	<1, X, 0>	<1, X, 0>
<1, X, 1>	<0, X, 0>	<0, X, 1>	<0, X, X>	<1, X, 0>	<1, X, 1>	<1, X, X>	<X, X, 0>	<X, X, 1>	<X, X, X>	<0, 0, 0>	<0, X, 1>	<1, X, 1>	<1, X, 0>
<1, X, X>	<0, X, 0>	<0, X, X>	<0, X, X>	<1, X, 0>	<1, X, X>	<1, X, X>	<X, X, 0>	<X, X, X>	<X, X, X>	<0, 0, 0>	<0, X, X>	<1, X, X>	<1, X, 0>
<X, X, 0>	<0, X, 0>	<0, X, 0>	<0, X, 0>	<X, X, 0>	<X, X, 0>	<X, X, 0>	<X, X, 0>	<X, X, 0>	<X, X, 0>	<0, 0, 0>	<0, X, 0>	<X, X, 0>	<X, X, 0>
<X, X, 1>	<0, X, 0>	<0, X, 1>	<0, X, X>	<X, X, 0>	<X, X, 1>	<X, X, X>	<X, X, 0>	<X, X, 1>	<X, X, X>	<0, 0, 0>	<0, X, 1>	<X, X, 1>	<X, X, 0>
<X, X, X>	<0, X, 0>	<0, X, X>	<0, X, X>	<X, X, 0>	<X, X, X>	<X, X, X>	<X, X, 0>	<X, X, X>	<X, X, X>	<0, 0, 0>	<0, X, X>	<X, X, X>	<X, X, 0>
<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>
<0, ^, 1>	<0, X, 0>	<0, X, 1>	<0, X, X>	<0, X, 0>	<0, X, 1>	<0, X, X>	<0, X, 0>	<0, X, 1>	<0, X, X>	<0, 0, 0>	<0, ^, 1>	<0, ^, 1>	<0, X, 0>
<1, 1, 1>	<0, X, 0>	<0, X, 1>	<0, X, X>	<1, X, 0>	<1, X, 1>	<1, X, X>	<X, X, 0>	<X, X, 1>	<X, X, X>	<0, 0, 0>	<0, ^, 1>	<1, 1, 1>	<1, v, 0>
<1, v, 0>	<0, X, 0>	<0, X, 0>	<0, X, 0>	<1, X, 0>	<1, X, 0>	<1, X, 0>	<X, X, 0>	<X, X, 0>	<X, X, 0>	<0, 0, 0>	<0, X, 0>	<1, v, 0>	<1, v, 0>

13-Value Example

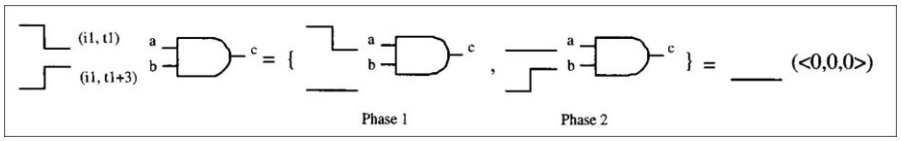


- CLK = <1, v, 0>, D = <1, 1, 1>
- Evaluate node e
  - a = b = <1, v, 0>, c = <0, ^, 1>, e = <1, X, 1>
- Very conservative as it is based on zero delay gate model
  - assumes no control over gate delays

Timestamps – Preserving Relative Transition Order

- maintaining the relative order of transitions, based on finite gate delays, can be achieved via ***timestamps***
- ***Timestamp*** is a pair (i, t)
  - i – signal group id – used to indicate causal transitions
  - t – time field – always increments
- Timestamps are only kept for signal transitions: <0, ^, 1>, <1, v, 0> - not necessary for other values
- Generating the i field for a multi-input gate (AND, OR)
  - If output result stems from input changes of same group (while other inputs remain stable) ➔ *group id remains the same*
  - Else if output result stems from input changes of multiple groups ➔ *generate new group id for the output signal and mark as successor*
- Successor group ids indicated via *group mask* of signal
  - bit per predecessor of current group id – 1 indicates true, 0 false

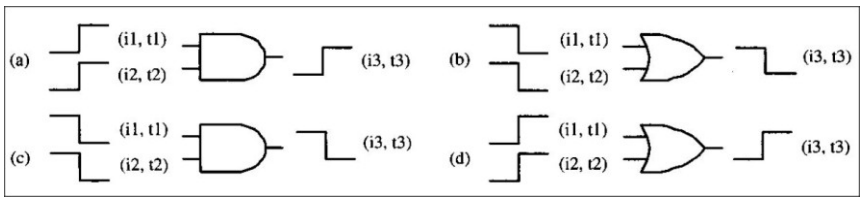
Gate Evaluation with Timestamps



- When inputs have same group id, they imply a relative order
  - Thus their evaluation is not performed via truth-table, but unfolds time into multiple phases
- Example above:
  - First time frame – inputs <1, v, 0>, <0, 0, 0>
  - Second time frame – inputs <0, 0, 0>, <0, ^, 1>
  - No hazard at output

Timestamps Example

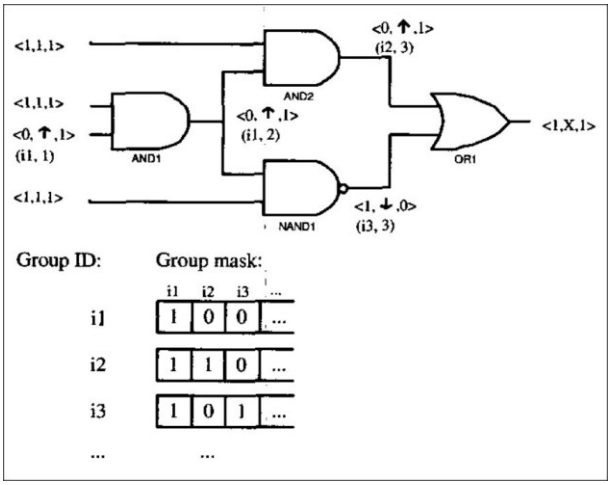
- In this example both inputs come from different groups: i1, i2
  - Thus a new group id, i3, is generated



  - cases (a), (b): output transition depends on both input transitions
  - cases (c), (d): output transition depends on first input transition
- Cases (a), (b): i3 is denoted as successor of i1, i2
  - Group masks:
  - i1i2i3(i1)= 100, i1i2i3(i2)= 010, i3i2i1(i3) = 011
- Cases (c), (d): i3 is not denoted as successor of i1, i2

13-Value coupled with Timing

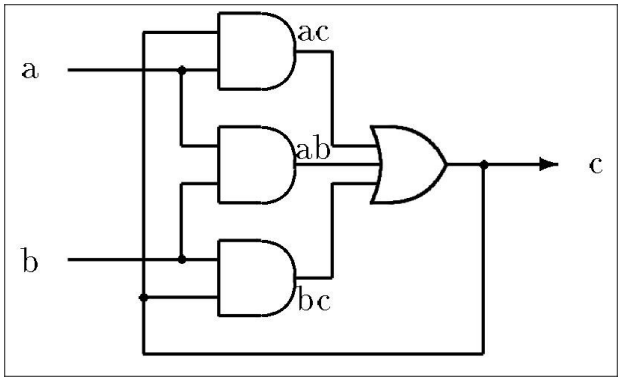
- At fanout points:
  - Different gate delays → different group ids for same input transition at fanout
- In RHS example:
  - Relative order between outputs of NAND, AND not guaranteed
  - Outputs assigned new group ids i2, i3
- Timing is incremented by 1
  - Not necessarily by actual gate delay



Sequential Hazard Analysis

- **Can be extended with 13-value logic and timeframes:**
  - Eichelberger's Two-Step Approach:
    - Procedure/Step 1 – Determine all ***changing*** Y signals
      1. Set changing input vector signals to intermediate 1/2 values, and all other x or y signals to their previous values
      2. Evaluate Y_i functions to determine changes from 1 or 0 to 1/2
      3. Propagate any 1/2 Y_i change to corresponding y_i change and repeat process until no further changes to Y_i occur
    - Procedure/Step 2 – Determine all ***stabilising*** Y signals
      - Set changing input vector signals to their final values, 1 or 0, and all other x or y signals to their previous values, as determined by Procedure 1
      - Evaluate Y_i functions to determine changes from 1/2 to 1 or 0
      - Propagate any 0 or 1 Y_i change to corresponding y_i change and repeat process until no further changes to Y_i occur

C-Element Hazard Example



- How do we analyse this circuit?
  - Ternary Sequential Hazard Analysis Approach?
  - 13-value Logic Approach?
  - Differences?

C-Element Hazard Example

- Ternary Analysis for 00→11→10 transitions

Time Frame	a	b	ab	bc	ac	C	c
1	0	0	0	0	0	0	0
2	1/2	1/2	1/2	0	0	1/2	0
3	1/2	1/2	1/2	0	0	1/2	1/2
4	1/2	1/2	1/2	1/2	1/2	1/2	1/2
5	1	1	1	1/2	1/2	1	1/2
6	1	1	1	1/2	1/2	1	1
7	1	1	1	1	1	1	1
8	1	1/2	1/2	1/2	1	1	1
9	1	0	0	0	1	1	1

- Does not detect sequential hazard – why?
  - No assumption in timing between 11→10 transitions (Fundamental Mode Operation)

C-Element Hazard Example

Circuit has not settled!

- 13-value Logic equivalent for 00→11→01 transitions

Time Frame	a	b	ab	bc	ac	C	c
1	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>
2	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, 0, 0>	<0, 0, 0>	<0, ^, 1>	<0, 0, 0>
3	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, 0, 0>	<0, 0, 0>	<0, ^, 1>	<0, ^, 1>
4	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>
5	<1, v, 0>	<0, ^, 1>	<0, X, 1>	<0, ^, 1>	<0, ^, 1>	**<0, X, 1>**	<0, ^, 1>
6	<1, v, 0>	<0, ^, 1>	<0, X, 1>	<0, X, 1>	<0, X, 1>	**<0, X, 1>**	<0, X, 1>

- By not letting **inputs** settle, 13-value logic detects hazard!

C-Element Hazard Example - 2

- 13-value Logic equivalent for 00→11→10 transitions

Time Frame	a	b	ab	bc	ac	C	c
1	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>	<0, 0, 0>
2	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, 0, 0>	<0, 0, 0>	<0, ^, 1>	<0, 0, 0>
3	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, 0, 0>	<0, 0, 0>	<0, ^, 1>	<0, ^, 1>
4	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>	<0, ^, 1>
5	<1, 1, 1>	<1, v, 0>	<1, v, 1>	<1, ^, 1>	<0, ^, 1>	**<1, X, 1>**	<0, ^, 1>
6	<1, 1, 1>	<1, v, 0>	<1, v, 1>	<1, ^, 1>	<0, ^, 1>	**<1, X, 1>**	<1, X, 1>

- By not letting **feedback** settle, 13-value logic detects hazard!

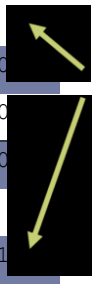
Another Approach to Hazards – Semi-modular Analysis

Semi-Modularity and Computation Interference

- Semi-modularity is a key property for designing hazard-free asynchronous systems
- Definition [*Semi-Modularity]*
  - a circuit is *semi-modular*, iff an output transition which has been enabled to fire (by an input transition) CANNOT subsequently be disabled (by a subsequent input transition)
- Semi-modularity forces output acknowledgement
  - Input→Output→next Input→ …
  - Thus leading to hazard-free SI systems
- Computation Interference
  - Violation of Semi-modularity
  - Example: AND2 gate with 11 inputs should present ^ rise transition before v in inputs

Semi-Modular FSM Specifications of Combinational Gates

Total State a b out (a.b)	00	01	10	11
000	000	010	100	110
001	000	X	X	X
010	X	010	X	110
011	X	010	X	110
100	X	X	100	110
101	X	X	100	X
110	X	X	X	111
111	001	011	101	111



- Arrows: states where the gate output will fire
  - Not accepting new inputs