Digital Electronics: Combinational Logic

Multilevel Logic and Hazards

Multilevel Logic

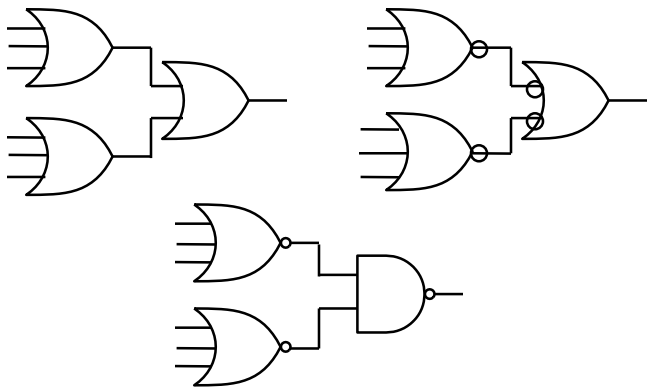
- We have seen previously how we can minimise Boolean expressions to yield so called ‘2-level’ logic implementations, i.e., SOP (ANDed terms ORed together) or POS (ORed terms ANDed together)
- Note also we have also seen an example of ‘multilevel’ logic, i.e., full adders cascaded to form a ripple carry adder – see we have more than 2 gates in cascade in the carry chain

Multilevel Logic

- Why use multilevel logic?
  - Commercially available logic gates usually only available with a restricted number of inputs, typically, 2 or 3.
  - System composition from sub-systems reduces design complexity, e.g., a ripple adder made from full adders
  - Allows Boolean optimisation across multiple outputs, e.g., common sub-expression elimination

Building Larger Gates

- Building a 6-input OR gate



Common Expression Elimination

- Consider the following minimised SOP expression:

$$z = a.d.f + a.e.f + b.d.f + b.e.f + c.d.f + c.e.f + g$$

- Requires:
  - Six, 3 input AND gates, one 7-input OR gate – total 7 gates, 2-levels
  - 19 literals (the total number of times all variables appear)

Common Expression Elimination

- We can recursively factor out common literals

$$z = a.d.f + a.e.f + b.d.f + b.e.f + c.d.f + c.e.f + g$$

$$z = (a.d + a.e + b.d + b.e + c.d + c.e).f + g$$

$$z = ((a + b + c).d + (a + b + c).e).f + g$$

$$z = (a + b + c).(d + e).f + g$$

- Now express z as a number of equations in 2-level form:

$$x = a + b + c \qquad y = d + e \qquad z = x.y.f + g$$

- 4 gates, 9 literals, 3-levels

Gate Propagation Delay

- So, multilevel logic can produce reductions in implementation complexity. What is the downside?
- We need to remember that the logic gates are implemented using electronic components (essentially transistors) which have a finite switching speed.
- Consequently, there will be a finite delay before the output of a gate responds to a change in its inputs – *propagation delay*

Gate Propagation Delay

- The cumulative delay owing to a number of gates in cascade can increase the time before the output of a combinational logic circuit becomes valid
- For example, in the Ripple Carry Adder, the sum at its output will not be valid until any carry has ‘rippled’ through possibly every full adder in the chain – clearly the MSB will experience the greatest potential delay

Gate Propagation Delay

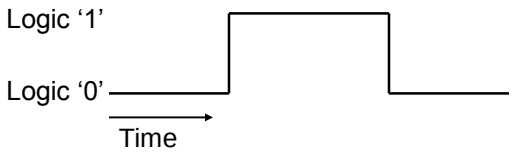
- As well as slowing down the operation of combinational logic circuits, gate delay can also give rise to so called '*Hazards*' at the output
- These *Hazards* manifest themselves as unwanted brief logic level changes (or *glitches*) at the output in response to changing inputs
- We will now describe how we can address these problems

Hazards

- Hazards are classified into two types, namely, static and dynamic
- Static Hazard – The output undergoes a momentary transition when *one input changes* when it is supposed to remain unchanged
- Dynamic Hazard – The output changes more than once when it is supposed to change just once

Timing Diagrams

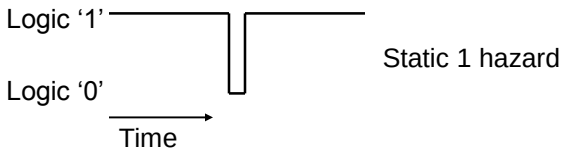
- To visually represent Hazards we will use the so called *'timing diagram'*
- This shows the logical value of a signal as a function of time, for example the following timing diagram shows a transition from 0 to 1 and then back again



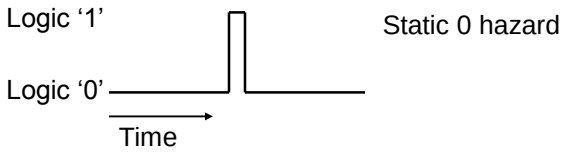
Timing Diagrams

- Note that the timing diagram makes a number simplifying assumptions (to aid clarity) compared with a diagram which accurately shows the actual voltage against time
  - The signal only has 2 levels. In reality the signal may well look more 'wobbly' owing to electrical noise pick-up etc.
  - The transitions between logic levels takes place instantaneously, in reality this will take a finite time.

Static Hazard

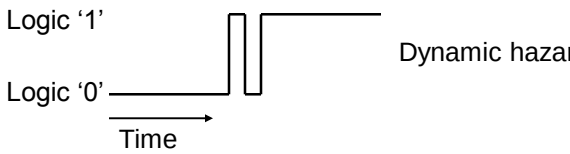


Static 1 hazard

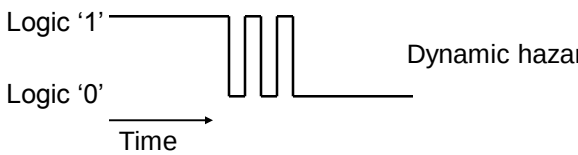


Static 0 hazard

Dynamic Hazard

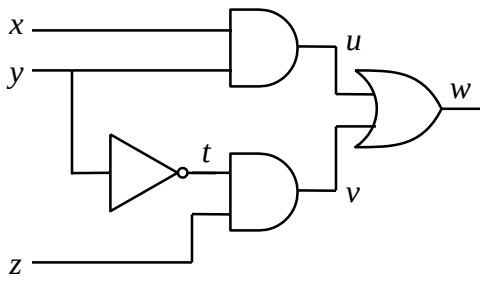


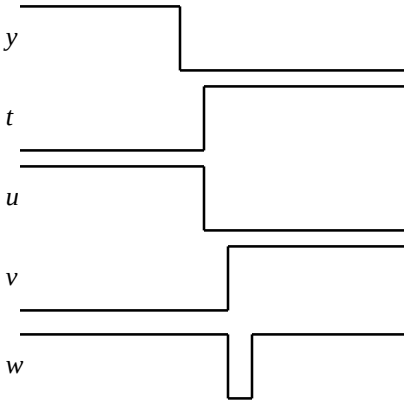
Dynamic hazard



Dynamic hazard

Static 1 Hazard





This circuit implements,

$$w = x.y + z.\bar{y}$$

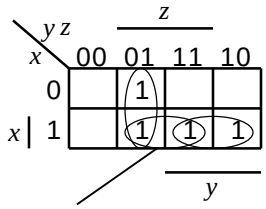
Consider the output when $z = x = 1$ and y changes from 1 to 0

Hazard Removal

- To remove a 1 hazard, draw the K-map of the output concerned. Add another term which overlaps the essential terms
- To remove a 0 hazard, draw the K-map of the complement of the output concerned. Add another term which overlaps the essential terms (representing the complement)
- To remove dynamic hazards – not covered in this course!

Removing the static 1 hazard

$w = x.y + z.\bar{y}$



Extra term added to remove hazard, consequently,

$w = x.y + z.\bar{y} + x.z$