ECE380 Digital Logic

Optimized Implementation of Logic Functions:

Multilevel Synthesis and Analysis

Multilevel synthesis

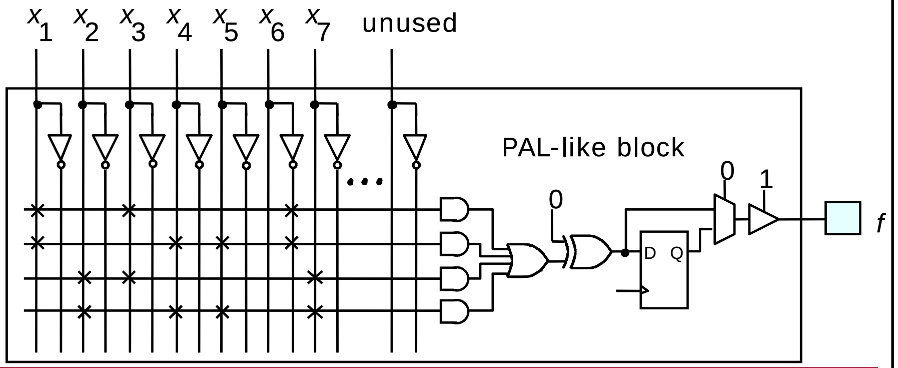
- For the previous minimization problems, the goal was to always find a minimum SOP or POS realization of a given logic function
- Circuits of this type have 2 levels (stages) of logic
  - For SOP form, the first level consists of only of AND gates that connect to a second level OR gate
  - For POS form, the first level consists of only of OR gates that connect to a second level AND gate
- We assume that both true and complement forms of the input variables are available.
- A two-level realization is usually efficient for functions of a few variables

Multilevel synthesis

- As the number of inputs increases, a two-level circuit may result in ***fan-in*** problems (depending on the technology used to implement the circuit)
- Fan-in: The number of inputs to a particular gate or circuit component
- Consider the following minimum cost SOP expression
  - f(x1,...,x7)=$X_1X_3X_6' + X_1X_4X_5X_6' + X_2X_3X_7 + X_2X_4X_5X_7$

Multilevel synthesis

- Consider implementing f in two types of PLDs:a CPLD and an FPGA
- This CPLD implementation works because we have enough inputs (at least 7), enough AND gates (one per product term), and enough OR gate inputs (one for each AND gate output)



Multilevel synthesis

- If we have an FPGA that has only 2-input LUTS, we cannot implement this function directly as written
  - Since the minimum SOP form had terms with three and four literals (requiring three- and four-input AND gates), and
  - There are four product terms needing to be ORed together (requiring a four-input OR gate)
- The ***fan-in*** required to implement this function is too high for an FPGA with only 2-input LUTs

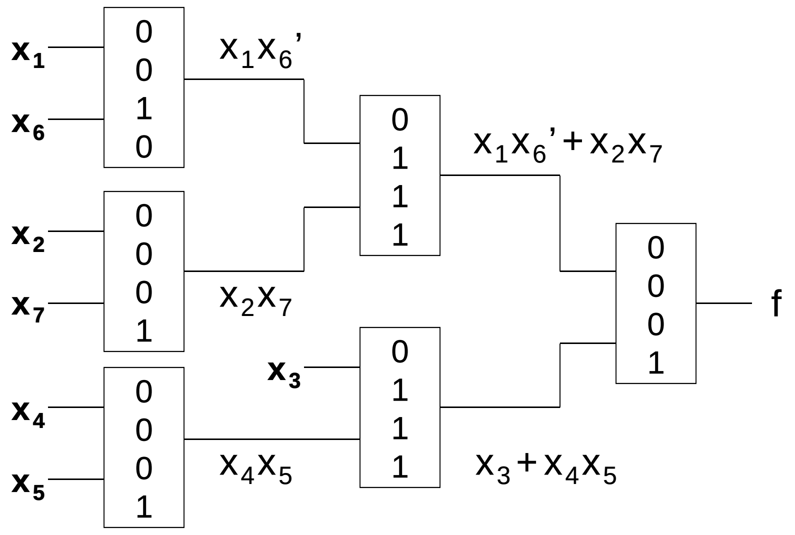
Multilevel synthesis

- To solve this problem, the function must be expressed in a form that has more than two levels of logic operations
  - Such a form is called a ***multilevel*** logic expression
- Two common techniques for synthesis of multilevel logic functions are:
  - ***Factoring***
  - ***Functional decomposition***

Factoring

- Factoring utilizes the distributive property to rewrite the expression in a form that generally has fewer literals per term
  - f(x1,…,x7) = $X_1X_3X_6' + X_1X_4X_5X_6' + X_2X_3X_7 + X_2X_4X_5X_7$
  - f(x1,…,x7) = $(X_1X_6' + X_2X_7)(X_3 + X_4X_5)$
- In this form, the function has no more than two literals comprising each ‘term’
- It can be implemented using only 2-input LUTs

Factoring



Fan-in problems

- Fan-in restrictions are not just a problem in PLDs as in the previous case
- Fan-in is also a problem for individual logic gates
- In general, as the number of inputs to a gate increases the ***propagation delay*** increases
- Propagation delay is the total amount of time needed for a change at a gate input to cause a change at the gate output
- Therefore, we may wish to limit the number of inputs to a given gate (5 is a typical maximum)

Fan-in problems

- Given the function
  - $f = X_1X_2'X_3X_4'X_5X_6 + X_1X_2X_3'X_4'X_5'X_6$
- The direct solution for this would require 2 six-input AND gates and 1 two-input OR gate (plus appropriate NOT gates)
- Factoring the function to the following form
  - $f = X_1X_4'X_6(X_2'X_3X_5 + X_2X_3'X_5')$
- Gives a solution requiring 2 three-input AND gates, 1 two-input OR gate, and 1 four-input AND gate

Fan-in problems

- Factor the following expression so that the solution requires only 2-input AND and OR gates
- Hint: The solution will require 4 AND and 2 OR gates (plus NOT gates)
  - $f(x_1,\ldots,x_7) = x_1x_2'x_4'x_5 + x_1x_2'x_6x_7' + x_3'x_4'x_5 + x_3'x_6x_7'$

Impact on wiring complexity

- Space on an integrated circuit is occupied by
  - The circuitry that implements logic gates, and
  - Wires needed to make connections between gates
- In a logic expression, each literal corresponds to a wire in the circuit that carries the desired logic signal
- Since factoring reduces the number of literals, it also aides in reducing the wiring complexity in a logic circuit
- During logic synthesis, CAD tools consider parameters such as: cost of the circuit (number of gates), fan-in, speed of the resulting logic and wiring complexity

Functional decomposition

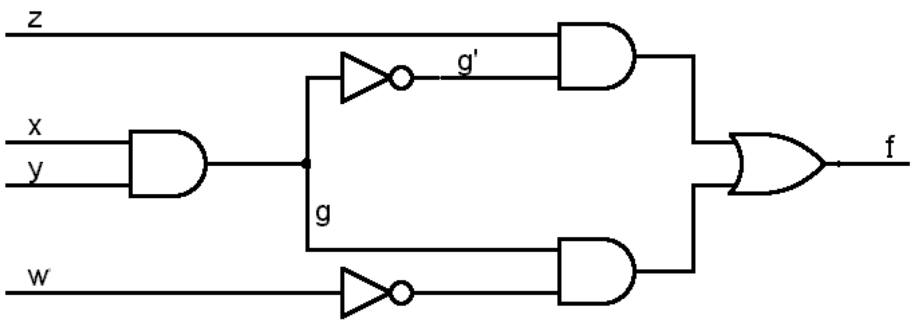
- Complexity of a logic circuit, in terms of wiring and logic gates, can often be reduced by ***decomposing*** a two-level circuit into subcircuits
  - One or more subcircuits implement functions that may be used in several places to construct the final circuit
- A single two-level logic expression is replaced by two or more new expressions
  - The new expressions are combined to define a multilevel circuit

Decomposition example

- Consider the following expression
  - $f(w,x,y,z)=xyw'+x'z+y'z$
- In this form, the function requires 1 three-input AND gate, 2 two-input AND gates, and 1 three-input OR gate
- COST=4 gates + 10 inputs = 13
- COST=19 if NOT gates (and their inputs) are included
- Rewrite f into the following form
  - $f(w,x,y,z)=(xy)w'+(x'+y')z$
- Let $g(x,y)=xy$ and note that $g'=x'+y'$

Decomposition example

- The function f becomes
  - f(g,w,z)=gw'+g'z
- The circuit would be the following with a cost of 16 (including NOT gates and their inputs)



Practical issues

- Functional decomposition is a powerful tool for reducing the complexity of logic functions
- It can also be used to implement general logic functions that have built-in constraints
  - For example, in PLDs, it is necessary to 'fit' a desired logic circuit into logic blocks that are available on these devices
  - Available logic blocks are a target for the decomposed subfunctions that are then used to form larger functions
- CAD tools make extensive use of the concept of functional decomposition