Design Verification

Lecture 04 - Multi-Level Logic Verification I

1. Technique #1: flatten the multi-level circuits into 2-level and use tautology checks.
   $\rightarrow$ problem: worst case: flat 2-level representation can have 2^{n-1} terms!

2. Technique #2: Enumeration-simulation ($A \equiv B$?)

   - Enumerate the ON-set cubes of A
     cube simulate these cubes on B, if answer is not 1, then $A \not\equiv B$
   - Enumerate also the OFF-sets of A
     cube simulate on B, if answer is not 0, then $A \not\equiv B$

   This technique is similar to containment checks, except no explicit storing of covers; instead, we need to enumerate them!

3. Technique #3: Satisfiability (SAT) on the miter circuit

   - SAT: given a formula f, derive a value assignment that satisfies f
   - need: express the miter circuit in a formula, and satisfy the output of the miter
   - want: the formula for the miter circuit to be reasonably short
   - CNF: conjunctive normal form

4. **Example 1a:** SAT formula for a two-input AND gate

   note: Satisfying the formula means $Z = XY$ is satisfied.

5. **Example 1b:** SAT formula for a three-input AND gate

6. **Example 1c:** SAT formula for an OR gate

   Example 2

7. Satisfying the SAT formula: simple algorithm (Davis-Putnam)

   (a) pick a variable v_i (v_i may be necessary assignment)

   (b) set $v_i = 0$ or 1

   (c) unit propagate v_i to formula

   (d) if any clause evaluates to 0, backtrack

   (e) repeat

   This algorithm is a **search** procedure that implicitly traverses the space of 2^n possible binary assignments to the problem. (n=#variables)

   Example 3

Example 3 (cont.)

8. Complexity of SAT solver
   - worst case can be exponential in the number of variables
   - decision tree: assignments nodes in the search/decision process
   - decision level: denotes the level of decision node in decision tree (first decision is at level 1)
   - additional assignments can be derived by **deduction/implication** process (eg. if a clause has one unassigned var left, then that var must evaluate to 1)
   - deduction process may lead to identification of unsatisfied clauses (all literals in the clause evaluate to 0)
   - backtrack: reversing the current assignment - try another assignment

9. Efficiency of SAT solver depends on:
   - quick identification of necessary assignment (all but one variable is 0 in a clause)

- selection of variable: compute cost in selecting variable v_i. Pick best variable.
- earlier backtrack: add additional clauses that may evaluate to 0 if wrong variable is selected

10. Quick identification of necessary assignment (Boolean Constraint Propagation (BCP))

    - keep a counter on number of unassigned variables in each clause
    - keep track on which variable is still unassigned/free in clause
    - necessary assignment on **unit** clauses (unit clause = a clause with one unassigned var)

11. Cost of variable v_i:

    - simple: cost(v_i) = # clauses v_i appears in
    - balanced weight: cost$(v_i) = K \times w(\overline{v_i}) \times w(v_i)$, where
      $w(\overline{v_i})$ = # clauses reduced when $v_i = 0$
      $w(v_i)$ = # clauses reduced when $v_i = 1$
      Key: favor variables whose $w(v_i) \sim w(\overline{v_i})$
    - Variable State Independent Decaying Sum (Chaff):
      (a) Need: computing occurrences of v_i or $\overline{v_i}$
      (b) Each variable in each polarity has a counter, initialize it to 0
      (c) When adding a clause (reading in the clause), increment the counter associated with each literal in the clause
      (d) Update the counter whenever a variable is assigned/unassigned
      (e) Divide the counter for every variable from time to time to *low-pass filter*, allow new conflict clauses added to take heavier weight
      (f) Pick unassigned variable with the highest counter value

Example 4

12. Enable earlier backtrack

    - View conflict as opportunity to augment the problem description to increase deductive power
    - conflict assignment: conjunction of conflicting assignment
    - conflict-induced clause: negation of the conjunction. This clause does not exist in current formula
    - Add conflict clauses that may evaluate to empty early if wrong variable assignment is chosen
    - These new clauses can prevent occurrence of same conflict in future
    - Deriving conflict clause:

      (a) includes those literals that occurred at previous decision levels, in addition to the decision that causes the conflict at current level

    Example 5

Conflict-driven learning continued

Example 5b

13. Exploring symmetry

    - if one branch of x (say x=0) leads to no solution, then we can prune the space under x=1 further, by looking at the conflicts obtained under x=0.
    - concept of supercubing

14. Technique #4: ATPG: use ATPG to try to detect the miter-output stuck-at-0 fault (bulk of ATPG algorithm covered in Testing course).

15. Basic ATPG algorithm: objective is miter-output = 1

```
Podem()
    if (miter-output == 1) return SUCCESS;
    (PI_i, val) = backtrace(miter-output, 1);
    if (PI_i = ∅) return FAILURE;
    logic_simulate(PI_i, val);
    if (Podem() == SUCCESS)        /* recursion */
        return SUCCESS;
    /* reverse decision */
    logic_simulate(PI_i, not(val));
    if (Podem() == SUCCESS)        /* recursion */
        return SUCCESS;
    logic_simulate(PI_i, X);
    return FAILURE;

backtrace(g, v)
    while (g != primary input)
        select an input, i, of g whose value is not don't care (X)
        if (g has an inversion) /* NAND, NOR, NOT, etc. */
            v = v XOR 1;
        g = i;
    return (g, v);
```

Example 6:

Example 7: