

Consensus

- If two implicants has distance larger or equal to 2
 - Consensus return 0
- If two implicants have distance equal to 1
 - Consensus return a single implicant
 - Parts that is “really close”

$$\text{CONSENSUS}(\alpha, \beta) = \left\{ \begin{array}{cccc} a_1 + b_1 & a_2 \cdot b_2 & \dots & a_n \cdot b_n \\ a_1 \cdot b_1 & a_2 + b_2 & \dots & a_n \cdot b_n \\ \dots & \dots & \dots & \dots \\ a_1 \cdot b_1 & a_2 \cdot b_2 & \dots & a_n + b_n \end{array} \right.$$



Cofactor

- First test to see if there is intersection
 - Then perform $\alpha_1 + \beta_1'$ $\alpha_2 + \beta_2'$... $\alpha_n + \beta_n'$



Operations on logic covers

- Recursive Paradigm
 - Base on the cofactor expansion theorem
 - E.g. $f+g=x(f_x+g_x)+x'(f_x'+g_x')$
 - Expand about a variable (possibly multi-valued)
 - Apply operation to cofactors
 - Merge Results
- Unate Heuristics
 - Operations on unate functions are simpler
 - Select variables so that cofactors become unate functions

$$f \odot g = \sum_{k=0}^{p-1} x^{(k)} \cdot (f_{x^{(k)}} \odot g_{x^{(k)}})$$



3

Unate mvi-functions

- Weak Unateness
 - A function f is weakly unate in variable x iff there exists a value j such that, for all other values k , $f_x^{(k)} \supseteq f_x^{(j)}$
 - A cover F is weakly unate in x iff subset of all implicants depend on variable x has a column of 0's in the x field



4

Compute all prime

- Given any two orthonormal basis x^A and x^B , A prime of f is
 - A prime of $x^A f_x^A$
 - A prime of $x^B f_x^B$
 - A prime of consensus of $x^A f_x^A$ and $x^B f_x^B$
 - $x^A f_x^A$ and $x^B f_x^B$ don't intersect
 - Its consensus is the corresponding part of the minterms that is distance 1
 - May be redundant in a cover (contained in union of multiple prime)



An algorithm for division

- $A = \{C_j^A, j = 1, 2, \dots, l\}$ set of cubes (monomials) of the dividend.
- $B = \{C_i^B, i = 1, 2, \dots, n\}$ set of cubes (monomials) of the divisor.
- Quotient Q and remainder R are sum of cubes (monomials).

```

ALGEBRAIC_DIVISION(A, B) {
  for (i = 1 to n) {
    D = {C_j^A such that C_j^A ⊇ C_i^B};
    if ( D == ∅ ) return(∅, A);
    D_i = D with var. in sup(C_i^B) dropped ;
    if i = 1
      Q = D_i;
    else
      Q = Q ∩ D_i;
  }
  R = A - Q × B;
  return(Q, R);
}
    
```

Kernel set computation

```

R_KERNELS(f){
  K = ∅;
  foreach variable x ∈ sup(f) {
    if ( |CUBES(f, x)| ≥ 2 ){
      /* skip cases where f/x yields one cube */
      C = largest cube containing x s.t. CUBES(f, C) = CUBES(f, x);
      K = K ∪ R_KERNELS(f/fC);
      /* recur on f/fC */
    }
  }
  K = K ∪ f;
  /* add f to kernel set */
  return(K);
}

```



Characterizing Observability Don't Care

- Given a logic network $G_n(V, E)$ and a vertex v_x , the **perturbed network** at v_x is the one obtained by replacing the local function f_x by $f_x \oplus \delta$, where δ is an additional input called perturbation
 - If $\delta = 0$, f_x is not perturbed
 - If $\delta = 1$, f_x is flipped
- ODC is characterized by $\overline{f \oplus f^x(1)}$,
 - where $f^x(1)$ is the perturbed network



Calculating Retiming

- A relaxation based method
- Calculate $|V|$ time
- Guarantee to find a feasible solution
 - If one exists

```

FEAS( $G_{sn}(V, E, W)$ ,  $\phi$ ){
  Set  $r = 0$ ;
  Set  $\tilde{G}_{sn}(V, E, \tilde{W}) = G_{sn}(V, E, W)$ ;
  for ( $k = 1$  to  $|V|$ ) {
    Compute the set of data-ready times;
    if ( $\max_{v_i \in V} t_i \leq \phi$ ) /* All path delays are bounded by  $\phi$  */
      return ( $\tilde{G}_{sn}(V, E, \tilde{W})$ );
    else /* Some path delays exceed  $\phi$  */
      foreach vertex  $v_i$  such that  $t_i > \phi$ 
         $r_i = r_i + 1$ ; /* Retime vertices with excessive delay */
         $\tilde{G}_{sn}(V, E, \tilde{W}) = G_{sn}(V, E, W)$  retimed by  $r$ ;
      }
  }
  return ( FALSE );

```

CS220: ↓

9

Tree matching

```

MATCH( $u, v$ ) {
  if ( $u$  is a leaf) return (TRUE); /* Leaf of the pattern graph reached */
  else {
    if ( $v$  is a leaf) return (FALSE); /* Leaf of the subject graph reached */
    if ( $degree(v) \neq degree(u)$ ) return(FALSE); /* Degree mismatch */
    if ( $degree(v) == 1$ ) { /* One child each: visit subtree recursively */
       $u_c = \text{child of } u$ ;  $v_c = \text{child of } v$ ;
      return (match( $u_c, v_c$ ))
    }
    else { /* Two children each: visit subtrees recursively */
       $u_l = \text{left-child of } u$ ;  $u_r = \text{right-child of } u$ ;
       $v_l = \text{left-child of } v$ ;  $v_r = \text{right-child of } v$ ;
      return (MATCH( $u_l, v_l$ ) · MATCH( $u_r, v_r$ ) + MATCH( $u_r, v_l$ ) · MATCH( $u_l, v_r$ ));
    }
  }
}

```

CS220: Synthesis of Digital System, Fall 06

10

Dynamic Programming

- Tree_Cover($T(V,E)$)
 - Set the cost of the internal vertices to -1 ;
 - Set the cost of the leaf vertices to 0 ;
 - While (some vertex has negative weight) do {
 - Select a Vertex $v \in V$ whose children have all nonnegative cost;
 - $M =$ set of all matching pattern trees at vertex v ;
 - $\text{Cost}(v) = \min_{m \in M(v)} (\text{cost}(m) + \sum_{u \in L(m)} \text{cost}(u))$;



N-Knapsacks Dynamic Programming

Algorithm

NKDP (S, n, d, A)

1. $A \leftarrow$ Sort the accelerators in CP in the decreasing order of their frequencies
2. $\text{min_t} \leftarrow \sum A_i.\text{mem_accesses} * d + A_i.\text{clock_cycles} / A_i.\text{clock}$
3. $\text{opt_sol} \leftarrow \{\emptyset\}$
4. for $i \leftarrow 1$ to n
 - 4.1. $\text{freq} = A_i.\text{fq}$
 - 4.2. for $j \leftarrow 1$ to $(i-1)$

$$V_j \leftarrow ((A_j.\text{mem_accesses} * f + A_j.\text{comp_cycles}) / A_j.\text{clk_freq}) - ((A_j.\text{mem_accesses} + A_j.\text{comp_cycles}) / \text{freq});$$

$$W_j \leftarrow A_j.\text{size};$$
 - 4.3. $S' = S - A_i.\text{size}$
 - 4.4. $\text{tmp_sol}, \text{tmp_t} \leftarrow \text{Knapsack01}(V, W, S')$
 - 4.5. $\text{tmp_sol} \leftarrow \text{tmp_sol} \cup A_i$;
 $\text{tmp_t} \leftarrow \text{tmp_t} + (A_i.\text{mem_accesses} + A_i.\text{comp_cycles}) / \text{freq}$
 - 4.6. if $\text{tmp_t} < \text{min_t}$
 $\text{min_t} \leftarrow \text{tmp_t}, \text{opt_sol} \leftarrow \text{tmp_sol}$
5. return opt_sol



Problem Definition

- Given:
 - a set of accelerators $A = \{a_1, a_2, \dots, a_n\}$ each with the following attributes:
 - Clock Cycles, Max Clock Frequency, Clock Frequency
 - A maximum number of unique clock frequencies F
- Find a positive integer value for every a_i .freq, s.t each a_i .freq is less than a_i .max_freq for every I , the number of distinct a_i .freq values is less than or equal to F , and the execution time E is minimized



13

CS220: Synthesis of Digital System, Fall 06

Problem Definition

Dynamic Programming Formulation

- Let $X(A,C)$ = total execution time of the first A accelerators using the first C clock frequencies.

```

If(A = 0)
  then X(A,C) = 0
Else if (C=0)
  then X(A,C) = ∞
Else
  X(A,C) = MIN1->A((∑aj.cycles)/aA.max_freq) + X(i-1, C-1)

```



14

CS220: Synthesis of Digital System, Fall 06

Model Checking Problem

- Model specified as Kripke structure $M=(S,R,L)$
 - A set of states S
 - Transition relation $R(S,S')$
 - Predicates over states specified by function L
- Given a Kripke structure $M = (S,R,L)$ that represents a finite-state transition graph and a temporal logic formula f
 - Find all states in S that satisfy f :
 - $\{s \in S \mid M,s \models f\}$
 - and check that initial states are among these.



15

CS220: Synthesis of Digital System, Fall 06

CTL properties

Path operators

- A - holds over all path.
- E - holds over one path.

Temporal operators

- Fp - p holds sometime in the *future*.
- Gp - p holds *globally* in the future.
- Xp - p holds *next* time.
- pUq - p holds *until* q holds.



Check AF, AG, AX, AU, EX, EF, EG, EU

- Can be reduced to check EX, EF, EU, EG



16

CS220: Synthesis of Digital System, Fall 06