Logic Optimization for Low Power

Combinational logic optimization

- Technology mapping

Sequential logic optimization

- Gated clocks
- Precomputation

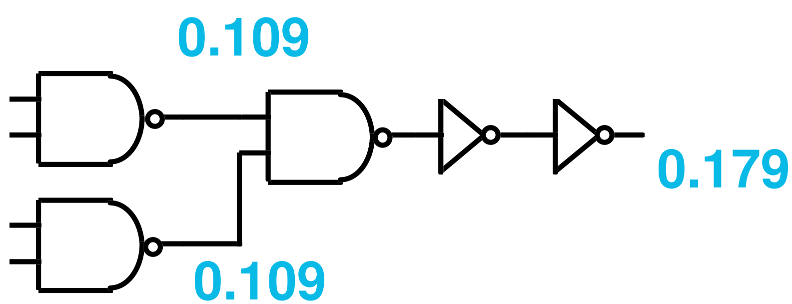
Technology Mapping

Outputs of nodes of combinational circuit mapped to library gates under the cost function of load capacitance multiplied by the switching activity

To minimize power dissipation, high switching activity points are either hidden within gates or driven by smaller gates.

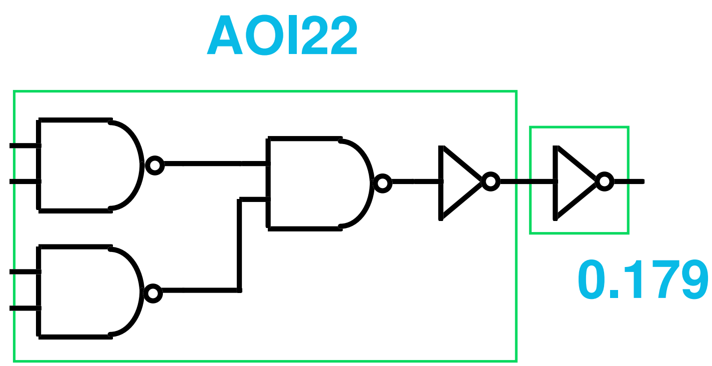
Minimum power realization under zero delay model can be obtained using dynamic programming.

Circuit and Gate Library



Gate	Area	Intrinsic Capacitance	Input Load Capacitance
INV	928	0.1029	0.0514
NAND2	1392	0.1421	0.0747
AOI22	2320	0.3410	0.1033

Minimum Area Mapping

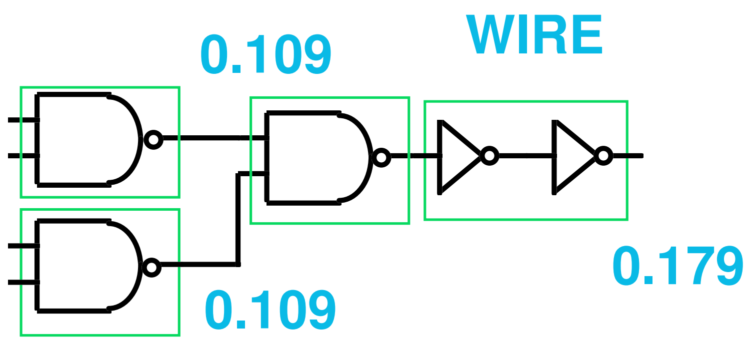


Area = 2320 + 928 = 3248

Power = 0.179 (0.3410 + 0.0514) + 0.179 (0.1029)
= 0.0887

Note: Ignoring capacitances at circuit inputs

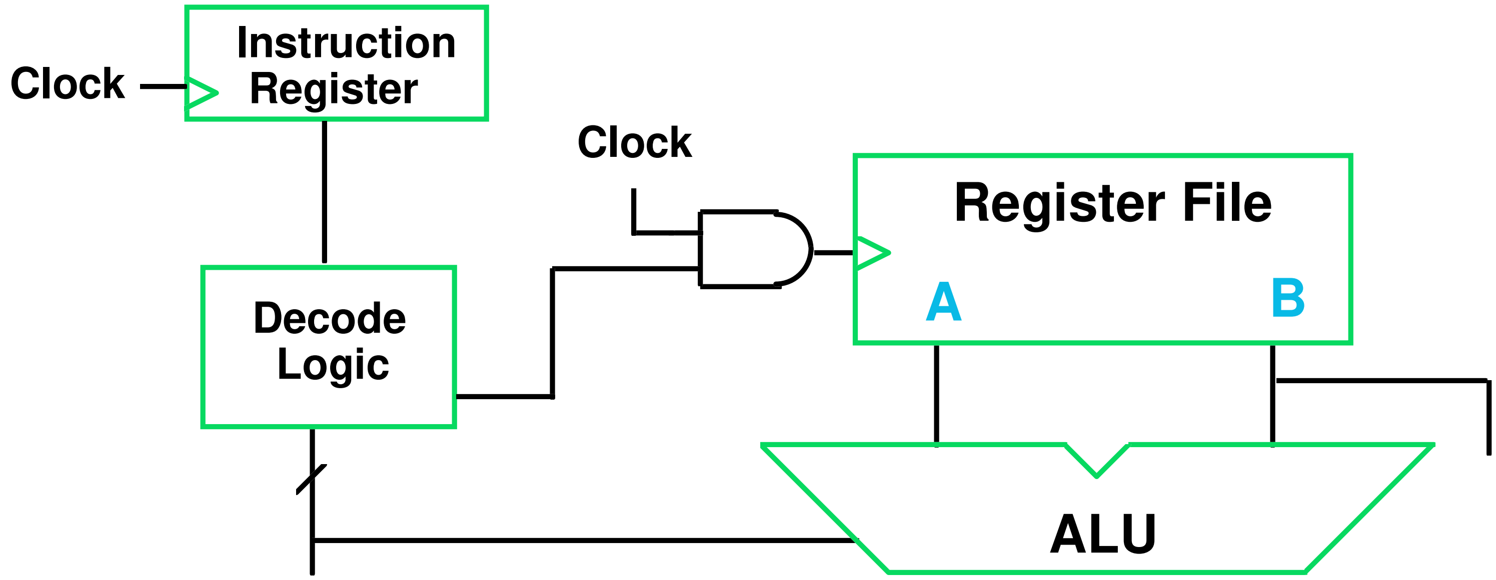
Minimum Power Mapping



Area = 1392 * 3 = 4176

Power = 0.109 (0.1421 + 0.0747) * 2 + 0.179 (0.1421)
= 0.0726

Gated Clocks

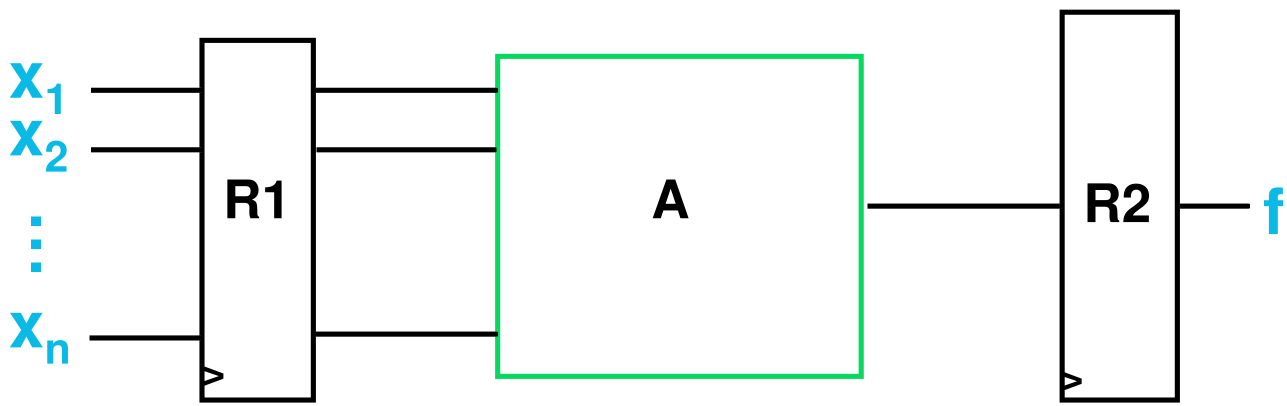


Disable clocking of register file if current instruction does not access it

Sequential Precomputation

Selectively precompute the outputs of the logic circuit one clock cycle before they are required and use the precomputed values to reduce switching activity in the next clock cycle.

Original Circuit

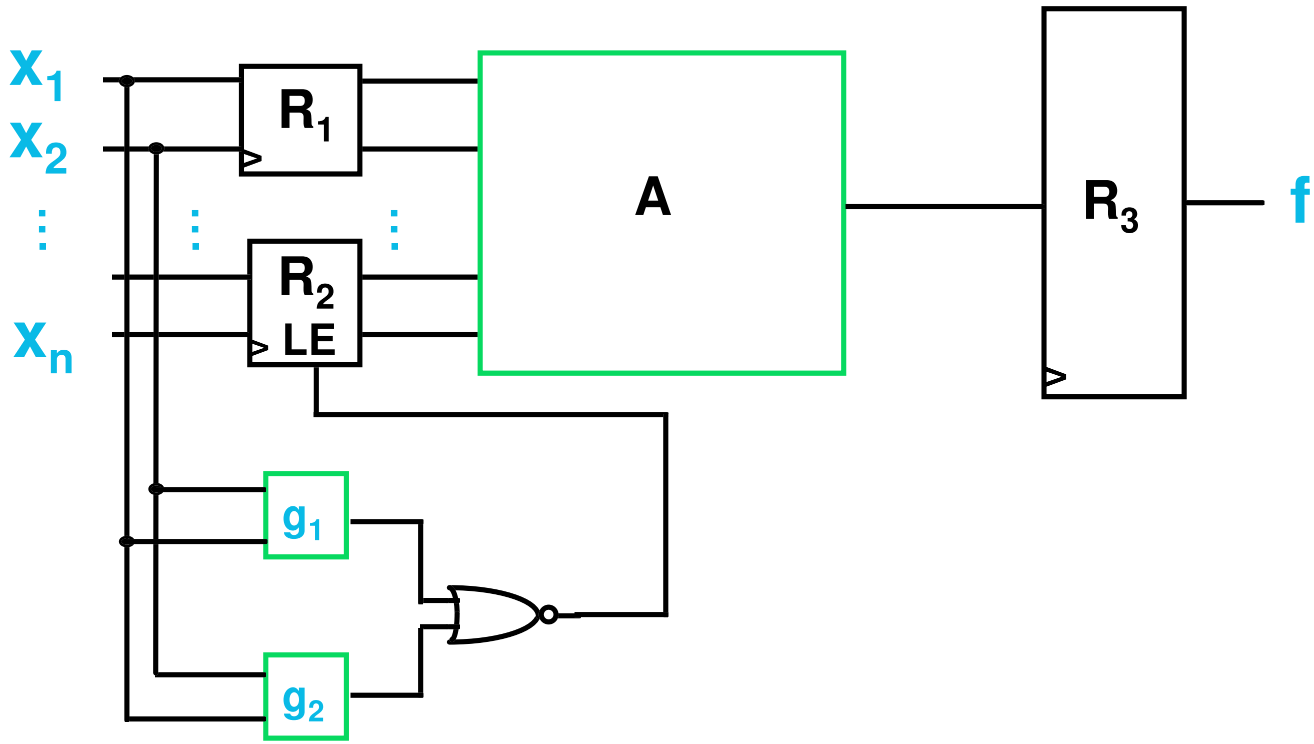


Predictor functions:

$$g_1 = 1 \Rightarrow f = 1$$

$$g_2 = 1 \Rightarrow f = 0$$

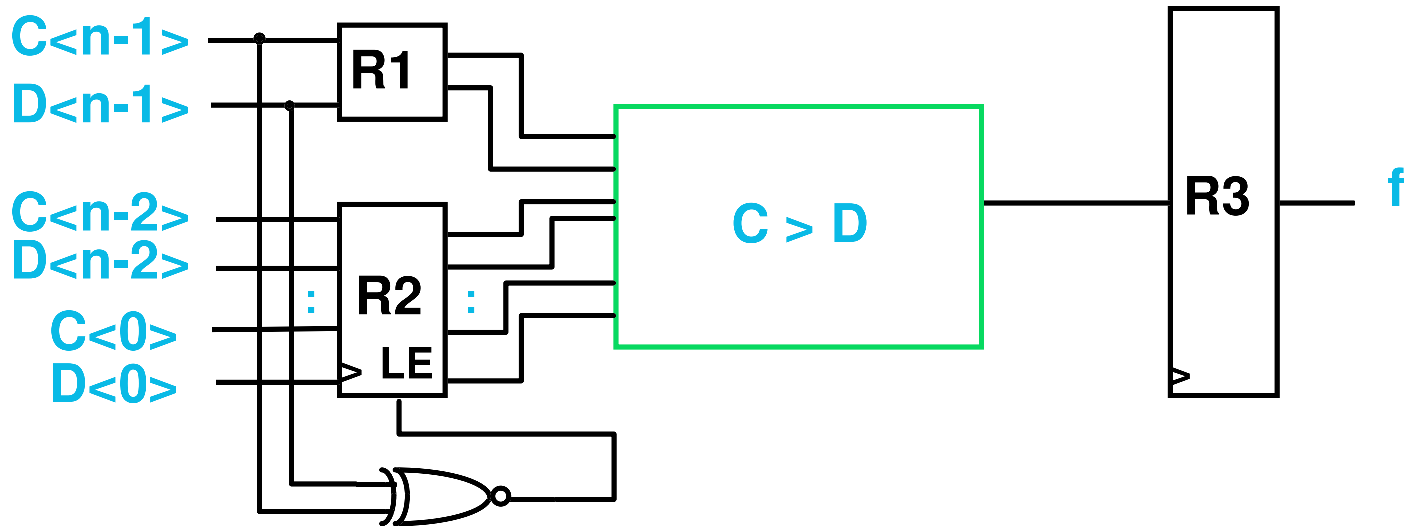
Subset Input Disabling Architecture



$g_1 = 1 \Rightarrow f = 1$

$g_2 = 1 \Rightarrow f = 0$

A Comparator Example



$$g_1 = C\langle n-1\rangle\ \overline{D\langle n-1\rangle}$$

$$g_2 = \overline{C\langle n-1\rangle}\ D\langle n-1\rangle$$

Precomputation Strategy

Given $f(x_1, \ldots, x_n)$ and

k, the number of inputs to the precomputation logic

Determine:

The inputs S to the precomputation logic

The precomputation logic functions g_1 and g_2

Find $S = \{ x_{j[1]}, \ldots, x_{j[k]} \}$ that maximizes

$$prob(g_1(x_{j[1]}, \ldots, x_{j[k]}) + g_2(x_{j[1]}, \ldots, x_{j[k]}) = 1)$$

Precomputation Logic

The Universal Quantification of a function f with respect to a variable x is defined as,

$$U_x f = f_x \, f_{\overline{x}}$$

where f_x ($f_{\overline{x}}$) is f with x set to 1 (0)

Given a set of inputs $X = \{x_1, \ldots, x_n\}$ choose a subset of inputs $S = \{ x_{j[1]}, \ldots, x_{j[k]} \}$ to the g_1 and g_2 functions.

Obtain the functions by:

$$g_1 = U_{x_{j[k+1]}, \ldots, x_{j[n]}} \, f$$

$$g_2 = U_{x_{j[k+1]}, \ldots, x_{j[n]}} \, \overline{f}$$

Example

$$f = ab + bc + ac$$

$$f_a = b + bc + c = b + c$$

$$f_{\overline{a}} = bc$$

$$U_a f = f_a f_{\overline{a}} = (b + c)\, bc = bc$$

$$U_{ab} f = U_a (U_b f) = U_b (U_a f)$$

$$= U_b (bc) = b\, 0 = 0$$

Input Selection Strategy

Initially set $g_1 = f$ and $g_2 = \overline{f}$

Traverse a binary tree where at each node we either select or discard an input x_i

Cost function: maximize $prob(g_1 + g_2 = 1)$

Pruning condition: $prob(g_1 + g_2 = 1)$ monotonically decreases as inputs are quantified

$$prob(U_x\, f) = prob(f_x\, f_{\overline{x}}) \leq prob(f) \quad \forall\, x, f$$

Input Selection Algorithm

```
SELECT_RECUR( g1, g2, D, Q )
{
    if ( |D| + |Q| < n - k ) return ;
    pr  =  prob(g1 = 1) + prob(g2 = 1) ;
    if ( pr  ≤  BEST_PROB )  return ;
    else if ( |D| == n - k )  {
        BEST_PROB  =  pr ;
        SELECTED_SET  =  X - D ;
        return;
    }
    choose xi ∈ Q such that  i  is minimum ;

    SELECT_RECUR( Uxi g1, Uxi g2, D ∪ xi, Q - xi ) ;
    SELECT_RECUR( g1, g2, D, Q - xi ) ;
    return ;
}
```

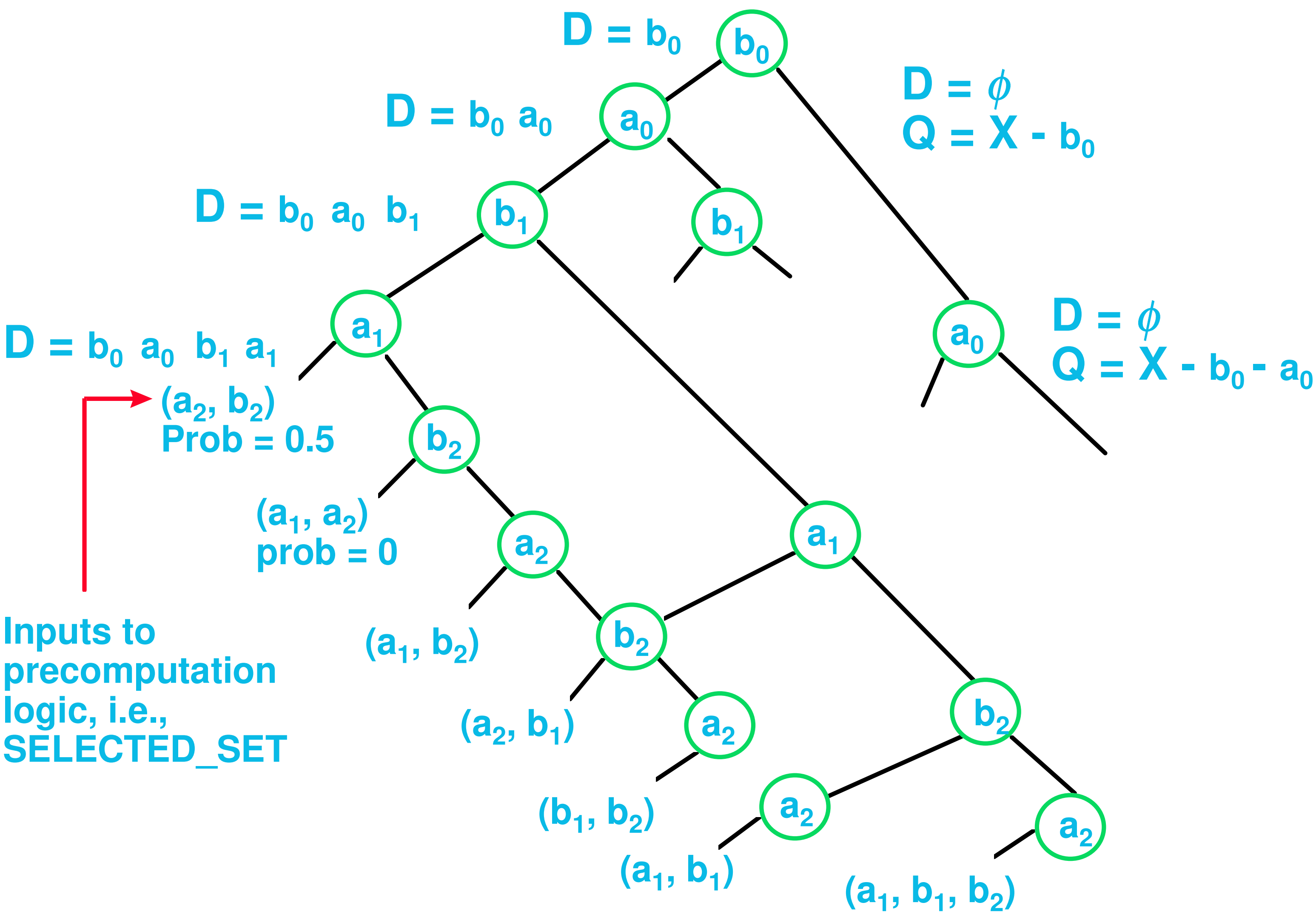
X: all n inputs

k: # inputs to precomputation logic

$D = \phi$ initially

$Q = X$ initially

A 3-Input Comparator Example



Multiple-Output Functions

Given a network $F = \{ f_1, \ldots, f_k \}$,
select a subset of outputs $G = \{ f_{j[1]}, \ldots, f_{j[m]} \}$

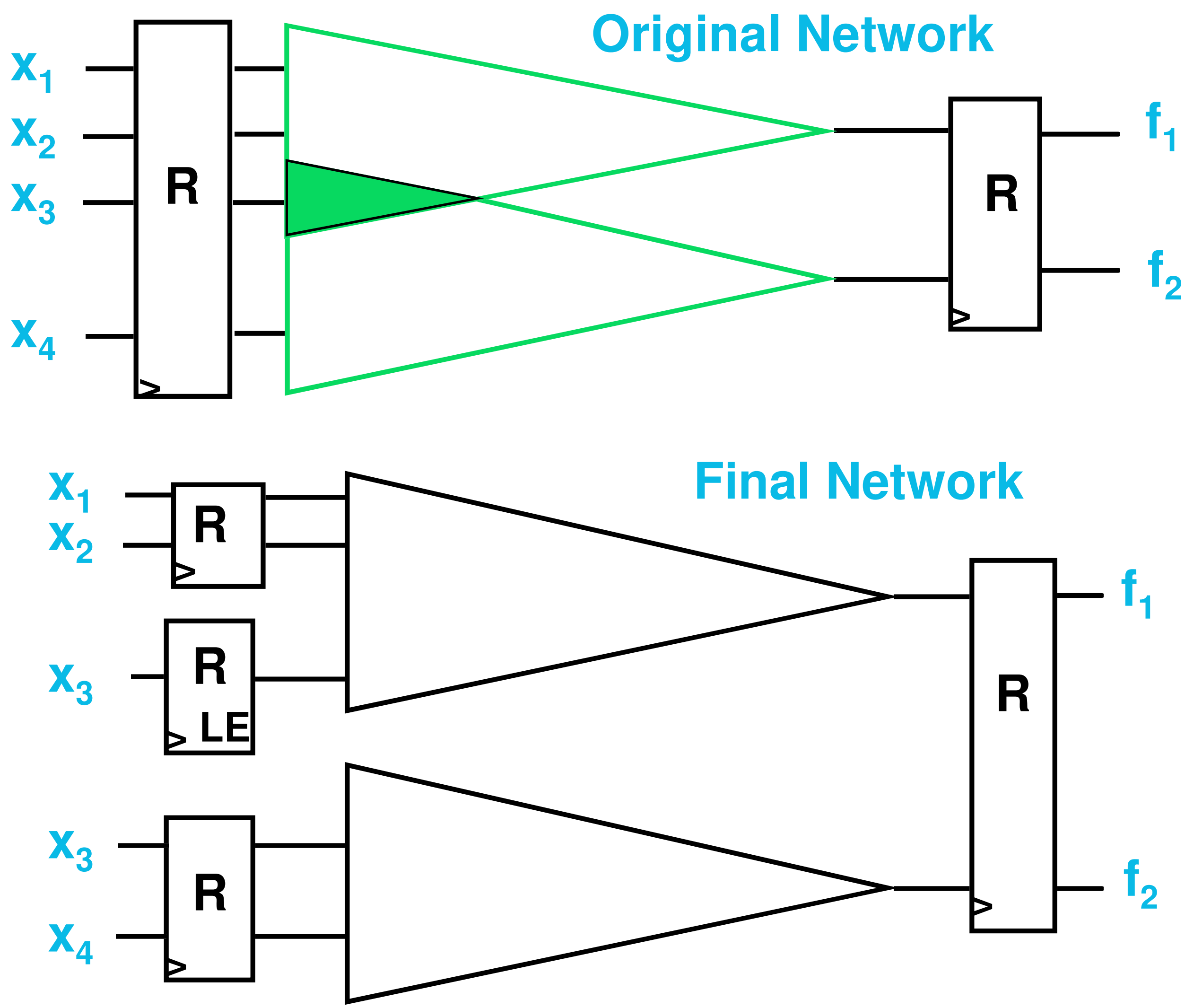
The load enable signal is given as,

$$g = \overline{\prod_{i=1}^{m} (g_{1i} + g_{2i})}$$

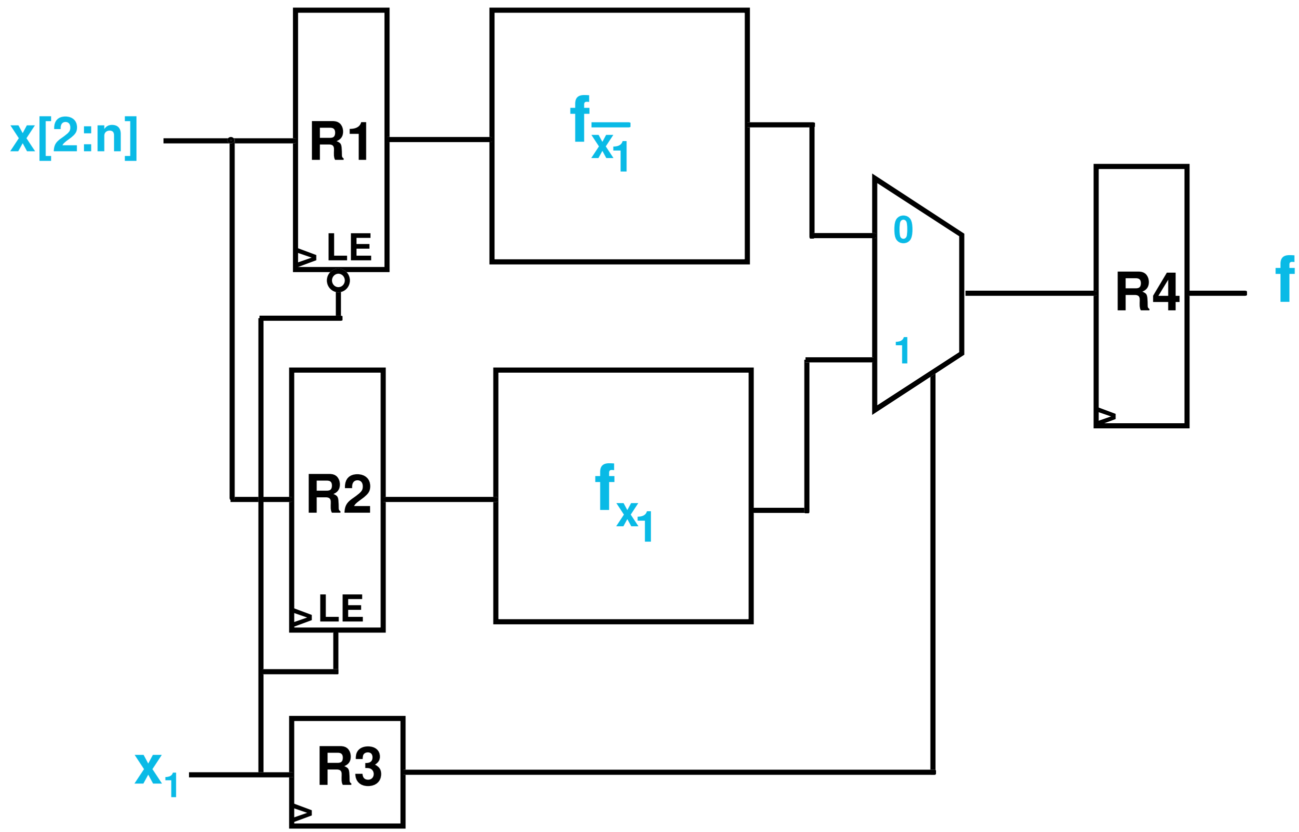
for predictor functions g_{1i} and g_{2i}

Duplicate some of the logic corresponding to
$H = \{ f_{j[m+1]}, \ldots, f_{j[n]} \}$

Logic Duplication

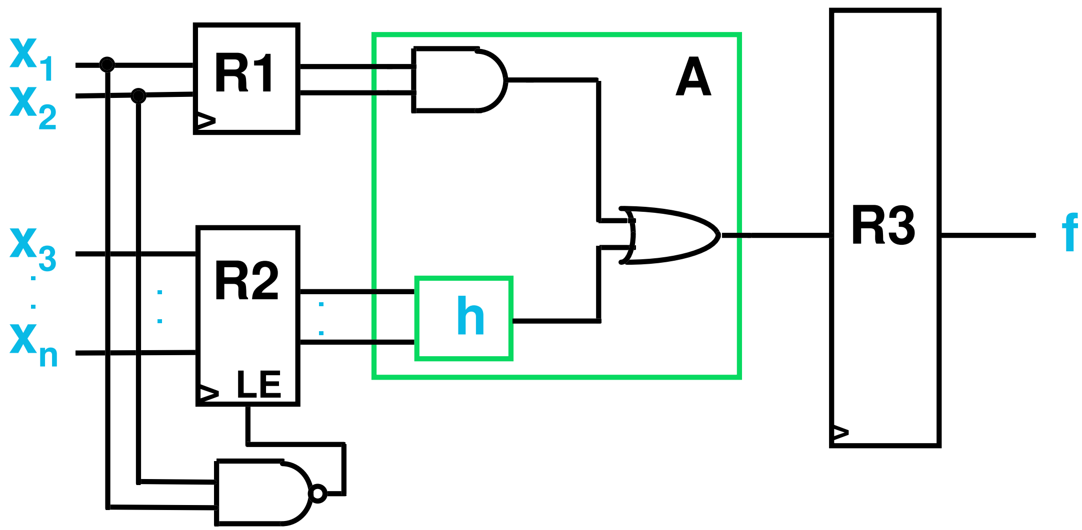


Multiplexor-based Precomputation



$$f = x_1 f_{x_1} + \overline{x_1} f_{\overline{x_1}}$$

Failing of SID Architecture

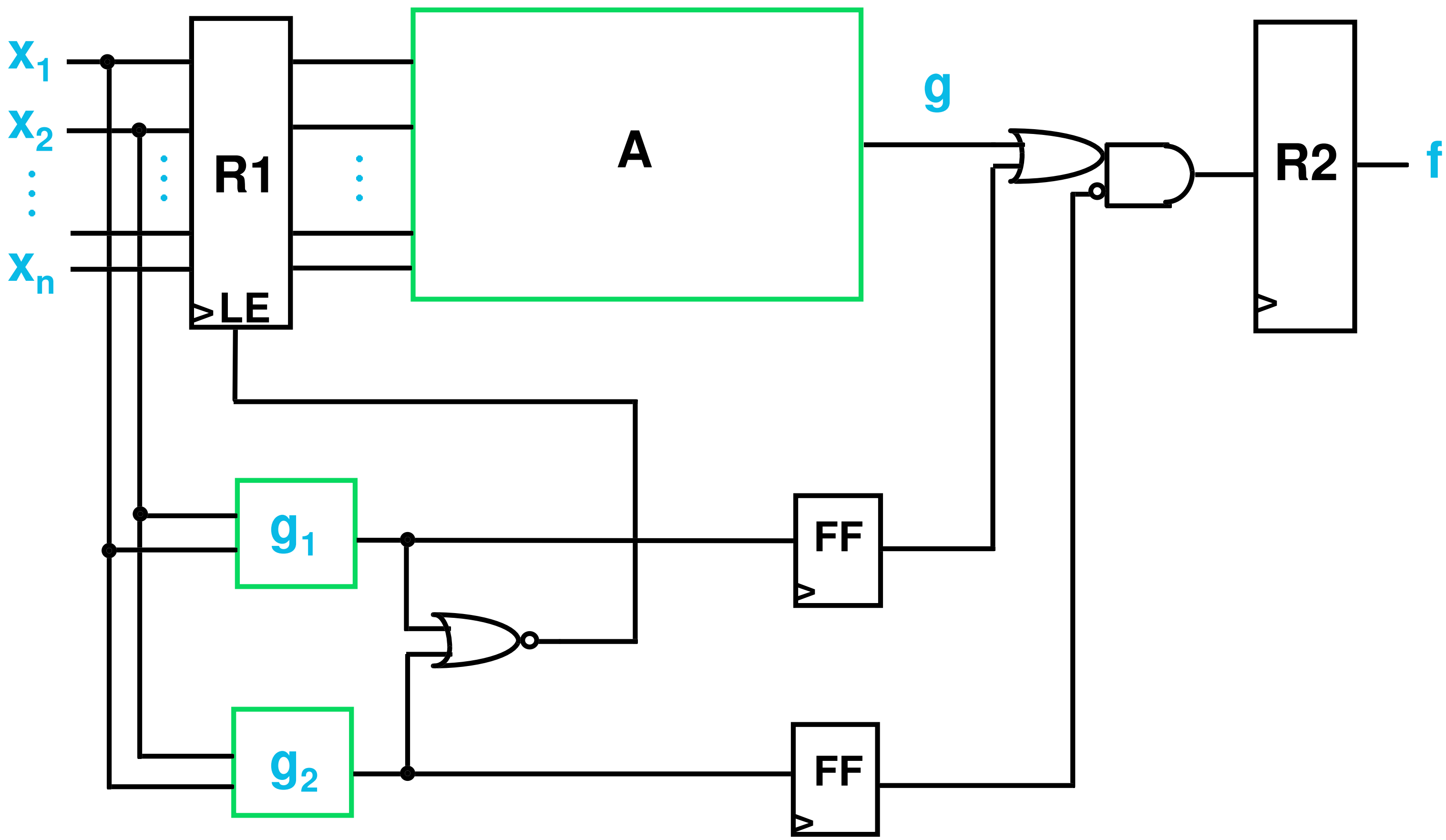


$f = x_1x_2 + h(x_3, \ldots, x_n) \Rightarrow$ **Power savings $\approx$ 25%**

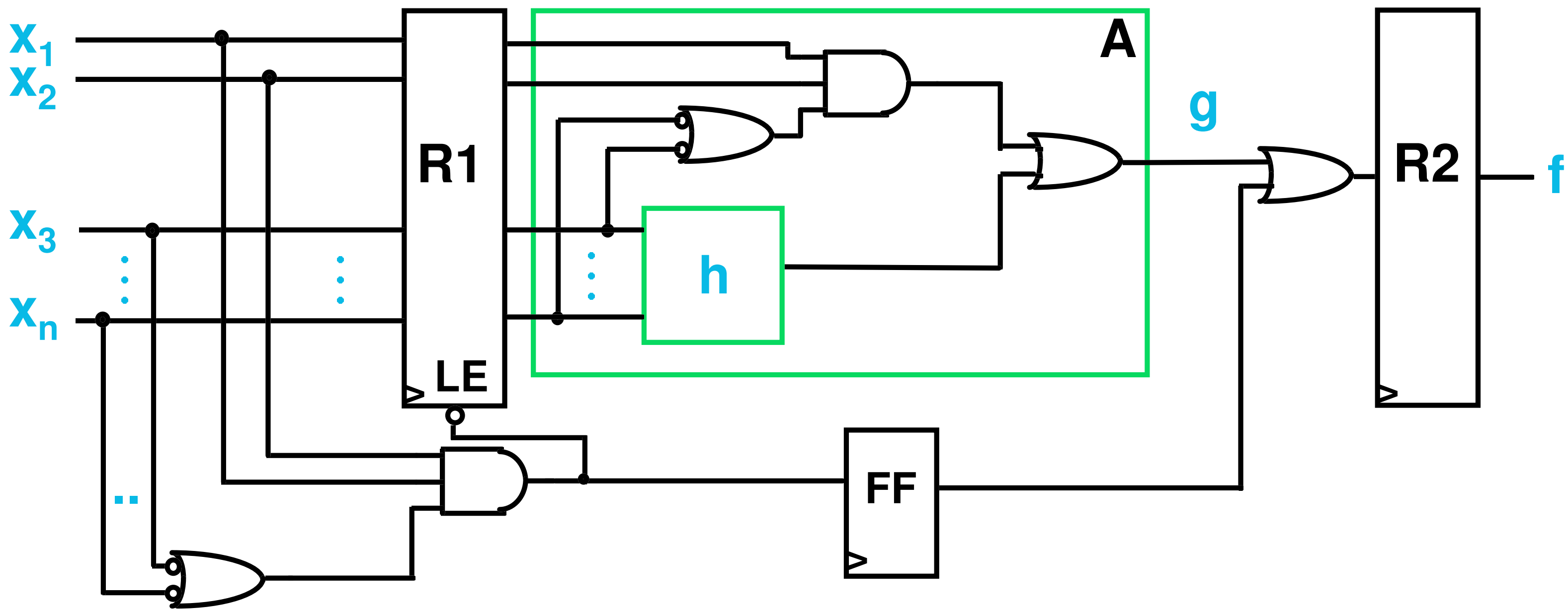
But we cannot precompute:

$$f = x_1x_2(\overline{x_3} + \overline{x_4} + \ldots + \overline{x_n}) + h(x_3, \ldots, x_n)$$

Complete Input Disabling Arch.



CID Architecture Example

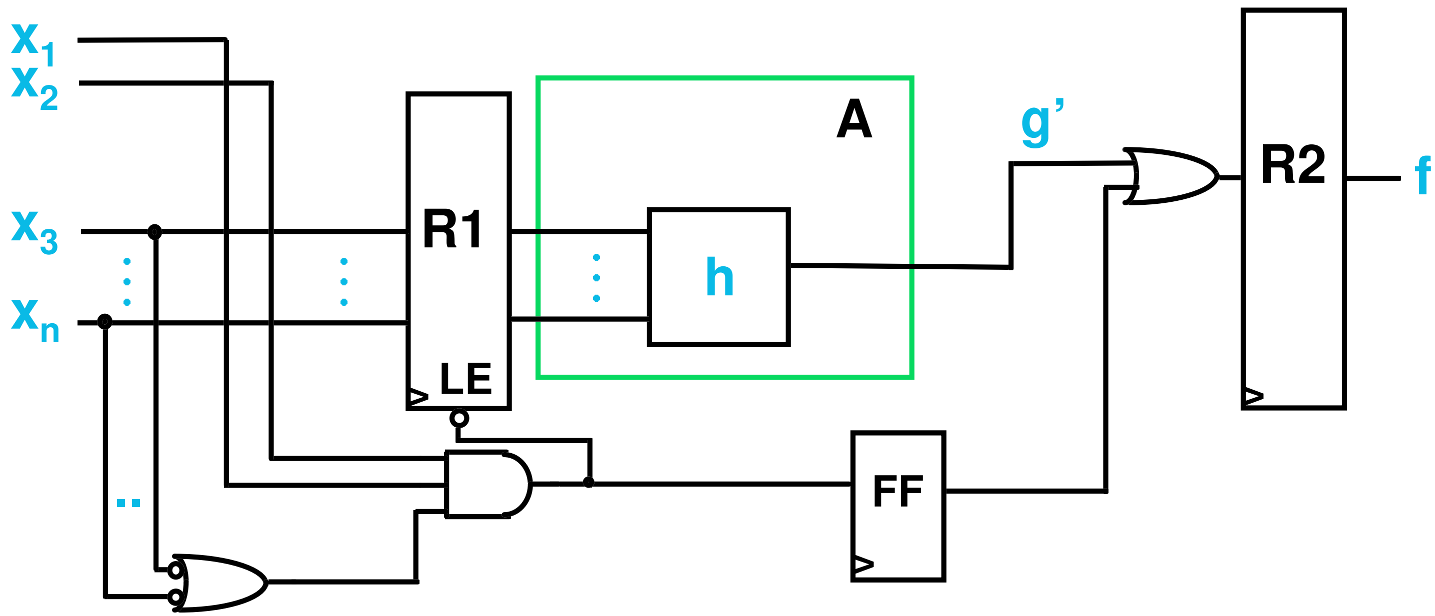


$$f = x_1x_2(\overline{x}_3 + \overline{x}_4 + \ldots + \overline{x}_n) + h(x_3, \ldots, x_n)$$

Power Savings ≈ 25%

$x_1x_2(\overline{x}_3 + \overline{x}_4 + \ldots + \overline{x}_n)$ are don't cares for g with respect to f

Don't Care Optimization



Similar to retiming original circuit, i.e., moving logic $x_1 x_2 (\overline{x}_3 + \overline{x}_4 + \ldots + \overline{x}_n)$ behind register to previous stage