2011 IC/CAD Contest

Introduction to EDA

National Taiwan University

Spring 2011

積體電路電腦輔助設計(CAD)軟體製作競賽

- 九十九學年度大學校院積體電路電腦輔助設計(CAD)軟體製作競賽
  - http://cad_contest.cs.nctu.edu.tw/cad11/

Outline

- A1: Structural checking of low-power logic design with voltage island & power gating
- A2: Formal property qualification
- A3: Multi-core static timing analysis
- B1: Numerical optimization on photo-mask – Model-based optical proximity correction
- B2: 3D IC design partitioning with power consideration
- B3: Gated clock cloning for timing fixing

Outline

- A1
- A2
- A3
- B1
- B2
- B3

Problem A1

- Structural checking of low-power logic design with voltage island & power gating
  - Low power techniques:
    - Multi-supply voltage (MSV)
    - Power gating (PSO)
  - Low power design validation:
    - Simulation (ineffective)
    - Structural checking (to be implemented)

Problem A1

- Problem description & I/O format
  - Input:
    - a design in gate-level Verilog netlist
    - specification of power intent
    - low power cell library
  - Output:
    - All design errors according to 10 pre-defined low-power design rules
    - Diagnosis information (location, type, occurrence of design error)

Problem A1

- Example

Input - Design file

```
module Top (iso, in, out1, out2);
  input iso, in;
  output out1, out2;
    Block1 u1 (.in(in), .out(n1));
    Block2 u2 (.in(n1), .iso(iso), .out(out1));
    Block3 u3 (.in(n1), .out(out2));
endmodule

module Block1 (in, out);
  input in;
  output out;
    SC_INV u0 (.A(in), .Y(out));
endmodule

module Block2 (iso, in, out);
  input iso, in;
  output out;
    IsoAND_TL_1P u_iso (.A(in), .Ib(iso), .Y(n1));
    SC_INV u1 (.A(n1), .Y(out));
endmodule

module Block3 (in, out);
  input in;
  output out;
    SC_INV u1 (.A(in), .Y(out));
endmodule
```

Problem A1

- Example (cont'd)

Input – Cell library

```
module SC_INV  (A, Y);
  input  A;
  output Y;
     not (Y, A);
endmodule

module IsoAND_TL_1P (A, Ib, Y);
  input  A, Ib;
  output Y;
     and (Y, Ib, A);
endmodule
```

Problem A1

- Example (cont'd)

Input – Power intent specification

```
define_isolation_cell -cells { IsoAND_TL_1P }  -enable Ib \
                      -valid_location to
create_power_domain -name TDD -boundary_ports { in } \
                              -instances { u1 }
create_power_domain -name TDR -boundary_ports { out1 out2 iso } \
                              -instances { u2 u3 } \
                              -default
create_nominal_condition -name onL -voltage 1.0
create_nominal_condition -name off -voltage 0
create_power_mode -name M1 -domain_conditions {TDD@onL TDR@onL} \
                  -default
create_power_mode -name M2 -domain_conditions {TDD@off TDR@onL}
create_isolation_rule -name iso1 -from TDD -to TDR \
                      -isolation_condition !iso \
                      -isolation_output low
```

Problem A1

- Example (cont'd)

Expected Ouput - Report ISO_4

[ISO_4] [Power domain crossing does not have user-defined isolation cell] [Occurrence:1]
- [#1] [Pin 'u3/in' in rule 'iso1' does not have isolation cell]

Problem A1

- Example (cont'd)

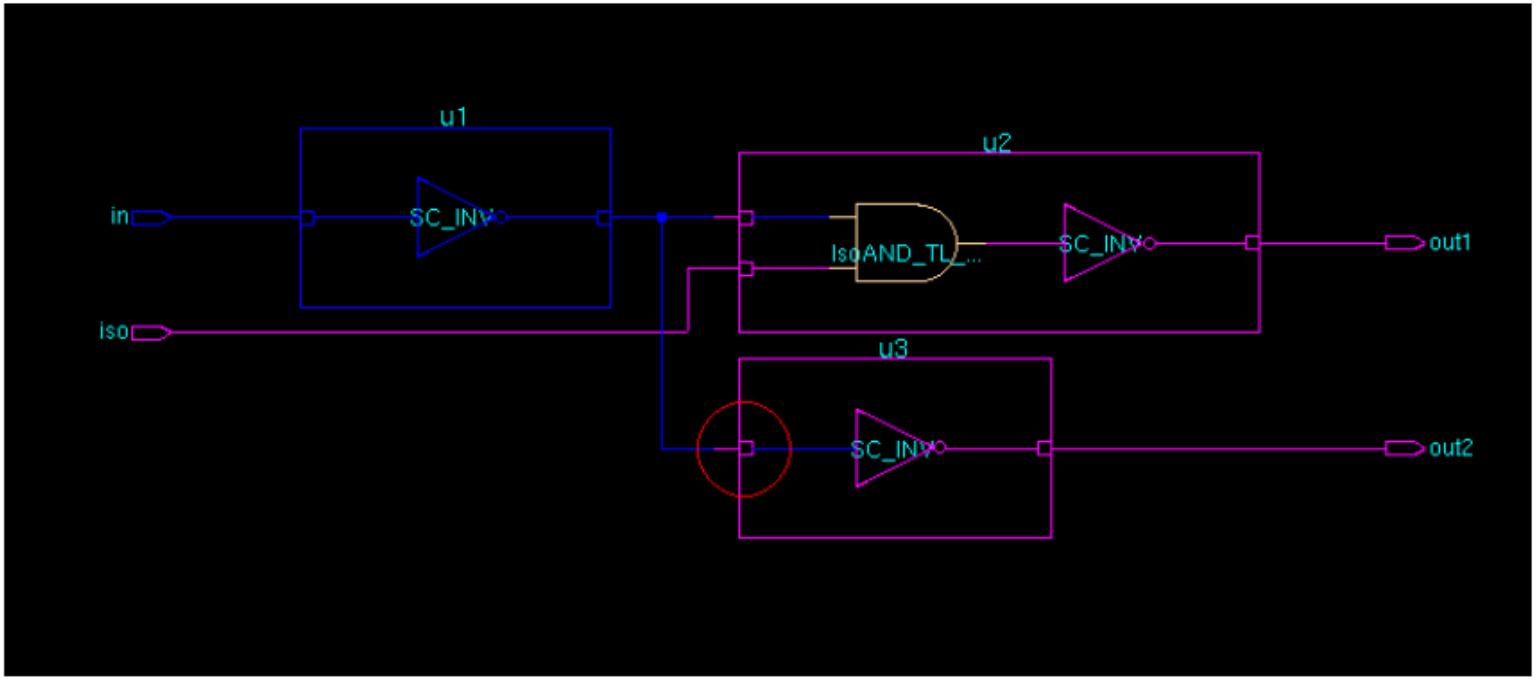


Figure 1. Missing isolation cell in the OFF/ON domain crossing (ISO_4) in the example 3.1

Problem A1

- Languate/platform
  - Language: C or C++
  - Platform: Linux OS
  - Implementation according to specified syntax
- Evaluation
  - Correctness
  - CPU time and memory

Outline

- A1
- A2
- A3
- B1
- B2
- B3

Problem A2

- Formal property qualification

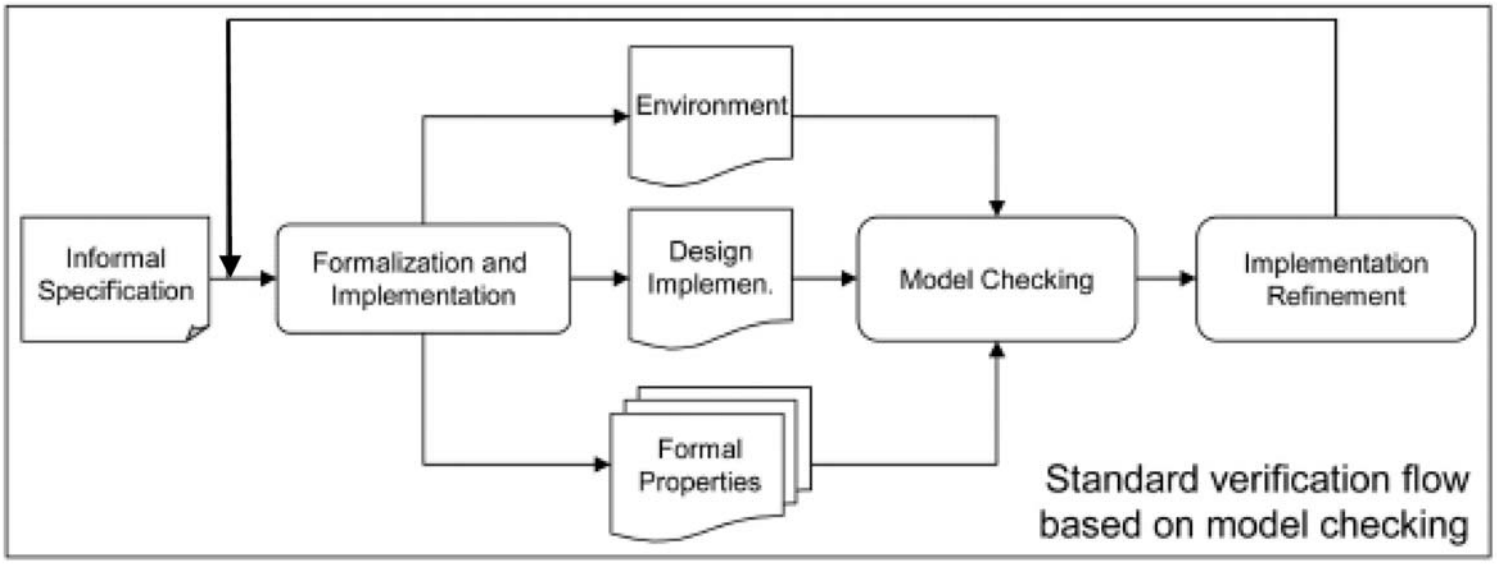


Figure 1: The formal verification process.

Problem A2

- Model checking
  - Verify whether a design satisfies some (un)desirable properties
  - Open problem: How many properties are enough for complete verification?
- Mutation based analysis (to be implemented)
  - A design is changed (mutated) into a set of mutants so as to introduce behavioral errors
  - A set of properties/tests is analyzed for each mutant to see which of the properties detect the difference between the design and the mutant

Problem A2

- I/O format
  - Input:
    - Design M and properties P[i], for i = 1,…,n, are given in DIMACS CNF format
    - Mutation types
      - A literal always positive
      - A literal always negative
      - A literal is negated
    - Mutants of M are denoted M[i], for i = 1, …, m
  - Output:
    - The set of “live” mutants
      - A mutant M[k] is live if it pass all P[i], that is, SAT(M[k], P[i]) = true for all i = 1, …, n

Problem A2

- Example

```
c Design: M
c
p cnf 50 80
16 23 42 0
-16 23 42 0
26 41 -42 0
-26 41 -42 0
32 -41 -42 0
6 15 -41 0
-6 15 -32 0
1 -32 46 0
-1 -32 46 0
-15 -41 -46 0
-15 -21 -46 0
.....
.....
```

Figure 2: Design *M*: written in DIMACS CNF format.

Problem A2

- Example (cont'd)

```
c Property: P[1]
c
p cnf 3 2
1 -3 0
2 3 -1 0
```

Figure 3: Property *P*[1]: written in DIMACS CNF format.

Problem A2

- Example (cont'd)

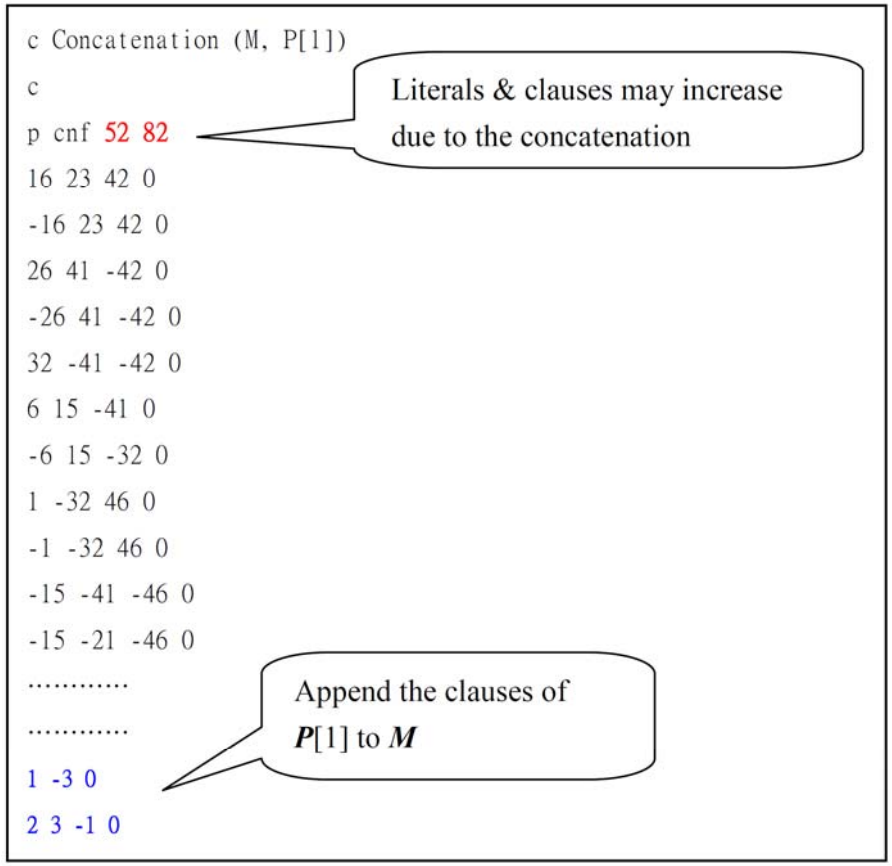


Figure 4: The concatenation of M and P[1]: (M, P[1]).

Problem A2

- Example (cont'd)

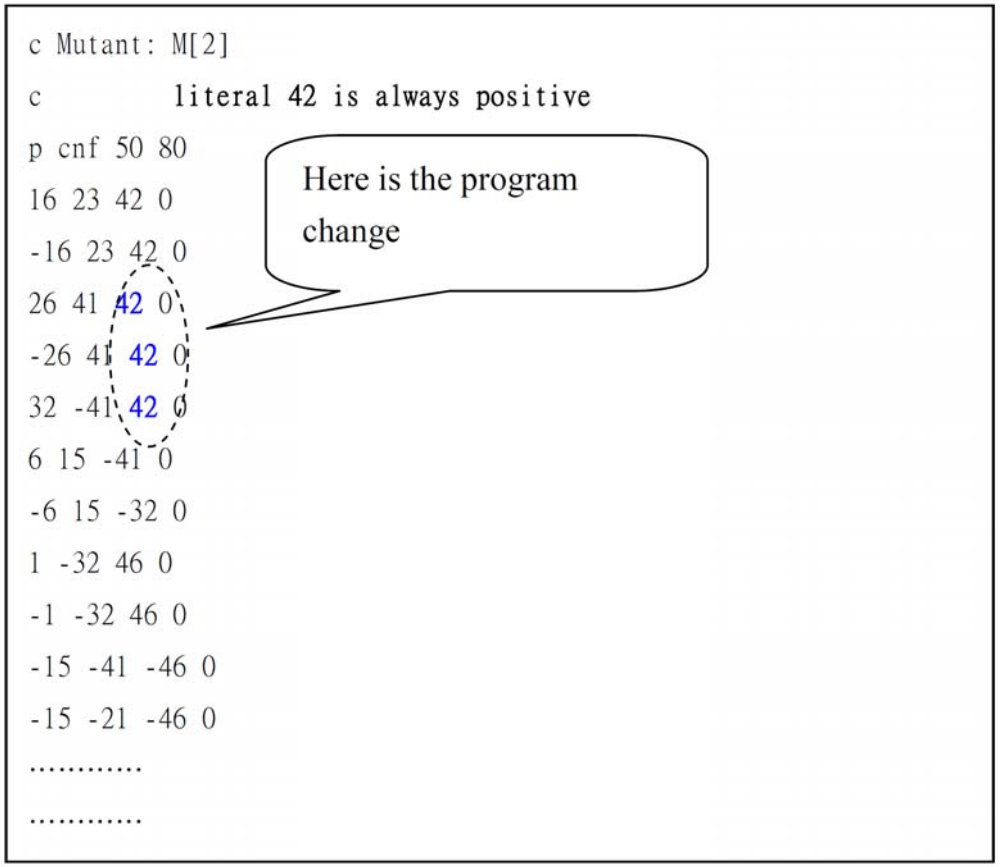


Figure 5: A Mutant *M*[2]: written in the DIMACS CNF format.

Problem A2

- Example (cont'd)

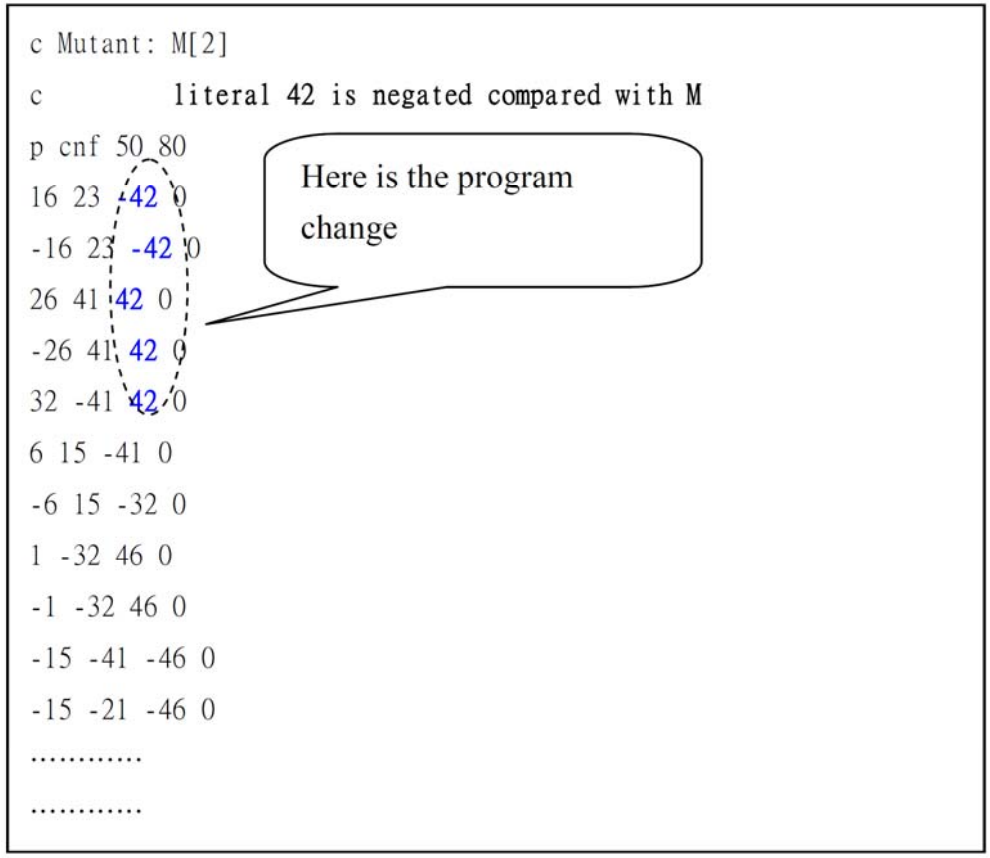


Figure 6: A Mutant *M*[4]: written in the DIMACS CNF format.

Problem A2

- Example (cont'd)

```
10
M_1
M_2
M_3
M_4
M_5
M_6
M_7
M_8
M_9
M_10
```

Figure 7: mutation_file

Problem A2

- Example (cont'd)

```
5
P_1
P_2
P_3
P_4
P_5
```

Figure 8: property_file

Problem A2

- Example (cont'd)

Figure 9: Output file format

Problem A2

- Popular state-of-the-art SAT solvers
  - MiniSAT
  - PrecoSAT
  - See SAT solver competitions
    - http://baldur.iti.uka.de/sat-race-2010/
    - http://www.satcompetition.org/

Outline

- A1
- A2
- A3
- B1
- B2
- B3

Problem A3

- Multi-core static timing analysis (STA)
  - STA validates the timing performance of a (combinational) design by indentifying timing “true” and “false” paths
  - Multi-core computer can be exploited to accelerate STA

Problem A3

- Terminology
  - True path
    - A path is true if there exists an input vector that **sensitizes** the path
    - We consider *sensitization* under the **floating mode delay** model
  - Floating mode delay model
    - All signals of a gate are initially of X (unknown) value
    - 3-valued simulation is performed for value update from inputs to outputs

Problem A3

- Example
  - True path

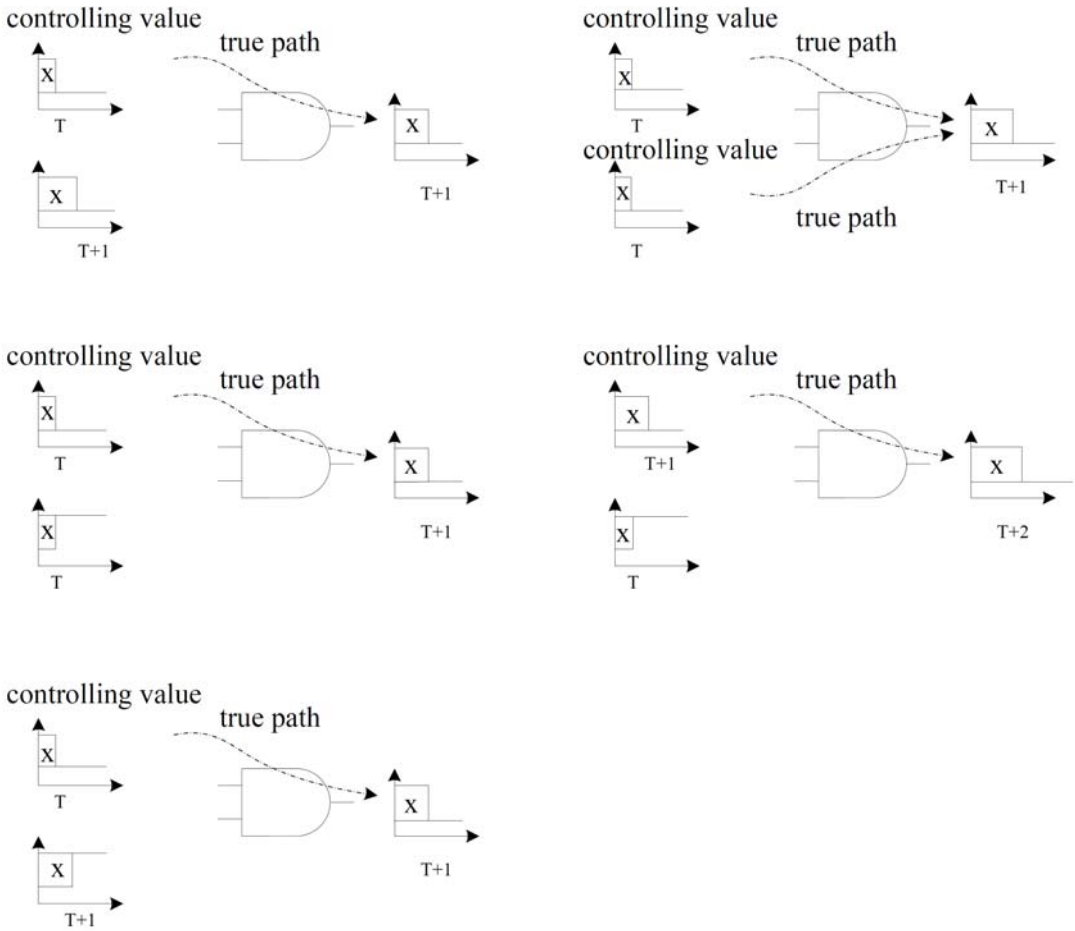


Figure3: An Example of AND gate Controlling value

Problem A3

- Example (cont'd)
  - True path

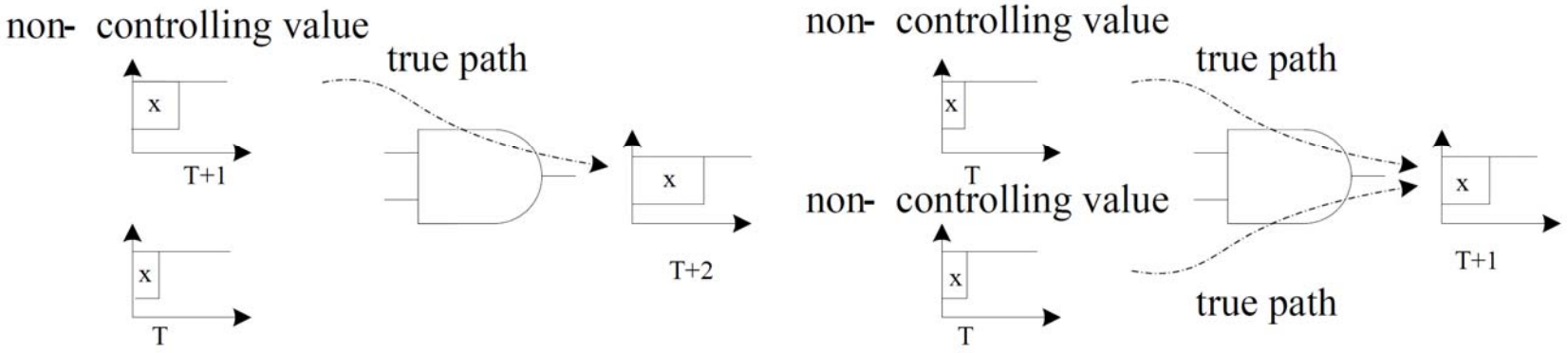


Figure4: An Example of AND gate non-controlling value

Problem A3

- Input example
  - Library cell

```
module NAND2(Y, A, B);
  input   A, B;
  output  Y;
  nand    (Y, A, B);
endmodule

module OR2(Y, A, B);
  input   A, B;
  output  Y;
  or      (Y, A, B);
endmodule
```

Figure 5: An example of Verilog model

Problem A3

- Input example
  - Verilog design

```
module multiplier2 ( A, B, M );
  input [1:0] A;
  input [1:0] B;
  output [2:0] M;
  wire   n1, n2, n3, n4, n5, n6, n7, n8;

  AND2 U1
( .A(n1), .B(B[1]), .Y(M[2]) );
  AND2 U2 ( .A(A[1]), .B(n2), .Y(n1) );
  NAND2 U3 ( .A(n3), .B(n4), .Y(n2) );
  NAND2 U4
( .A(n5), .B(n6), .Y(M[1]) );
  NAND2 U5 ( .A(n3), .B(n7), .Y(n6) );
  NOT1 U6 ( .A(n8), .Y(n3) );
  NAND2 U7 ( .A(n4), .B(n8), .Y(n5) );
  NAND2 U8
( .A(B[0]), .B(A[1]), .Y(n8) );
  NOT1 U9 ( .A(n7), .Y(n4) );
  NAND2 U10
( .A(A[0]), .B(B[1]), .Y(n7) );
  AND2 U11
( .A(A[0]), .B(B[0]), .Y(M[0]) );
endmodule
```

Figure 6: An example of Verilog gate-level netlist - 2 bits multipiler

Problem A3

- Output example

```
Header  {  A True Path Set  }

Benchmark  {  case1  }

Path  {  1  }

A True Path List
{
  ---------------------------------------------------------------------------
  Pin       type       Incr    Path delay
  ---------------------------------------------------------------------------
  A[0]      (in)       0       0        r
  U10/A     (NAND2)    0       0        r
  U10/Y     (NAND2)    1       1        r
  U9/A      (NOT1)     0       1        f
  U9/Y      (NOT1)     1       2        f
  U3/B      (NAND2)    0       2        r
  U3/Y      (NAND2)    1       3        r
  U2/B      (AND2)     0       3        f
  U2/Y      (AND2)     1       4        f
  U1/A      (AND2)     0       4        f
  U1/Y      (AND2)     1       5        f
  M[2]      (out)      0       5        f
  ---------------------------------------------------------------------------
  Data Required Time               10
  Data Arrival Time                5
  ---------------------------------------------------------------------------
  Slack                            5
}

Input Vector
{
  A[0]  =  r
  A[1]  =  1
  B[0]  =  1
  B[1]  =  1
}

Path  {  2  }

A True Path List
{
  ---------------------------------------------------------------------------
  Pin       type       Incr    Path delay
  ---------------------------------------------------------------------------
  A[0]      (in)       0       0        f
  U10/A     (NAND2)    0       0        f
  U10/Y     (NAND2)    1       1        f
  U9/A      (NOT1)     0       1        r
  U9/Y      (NOT1)     1       2        r
  U3/B      (NAND2)    0       2        f
  U3/Y      (NAND2)    1       3        f
  U2/B      (AND2)     0       3        r
  U2/Y      (AND2)     1       4        r
  U1/A      (AND2)     0       4        r
  U1/Y      (AND2)     1       5        r
  M[2]      (out)      0       5        r
```

Problem A3

- Example
  - Floating mode simulation

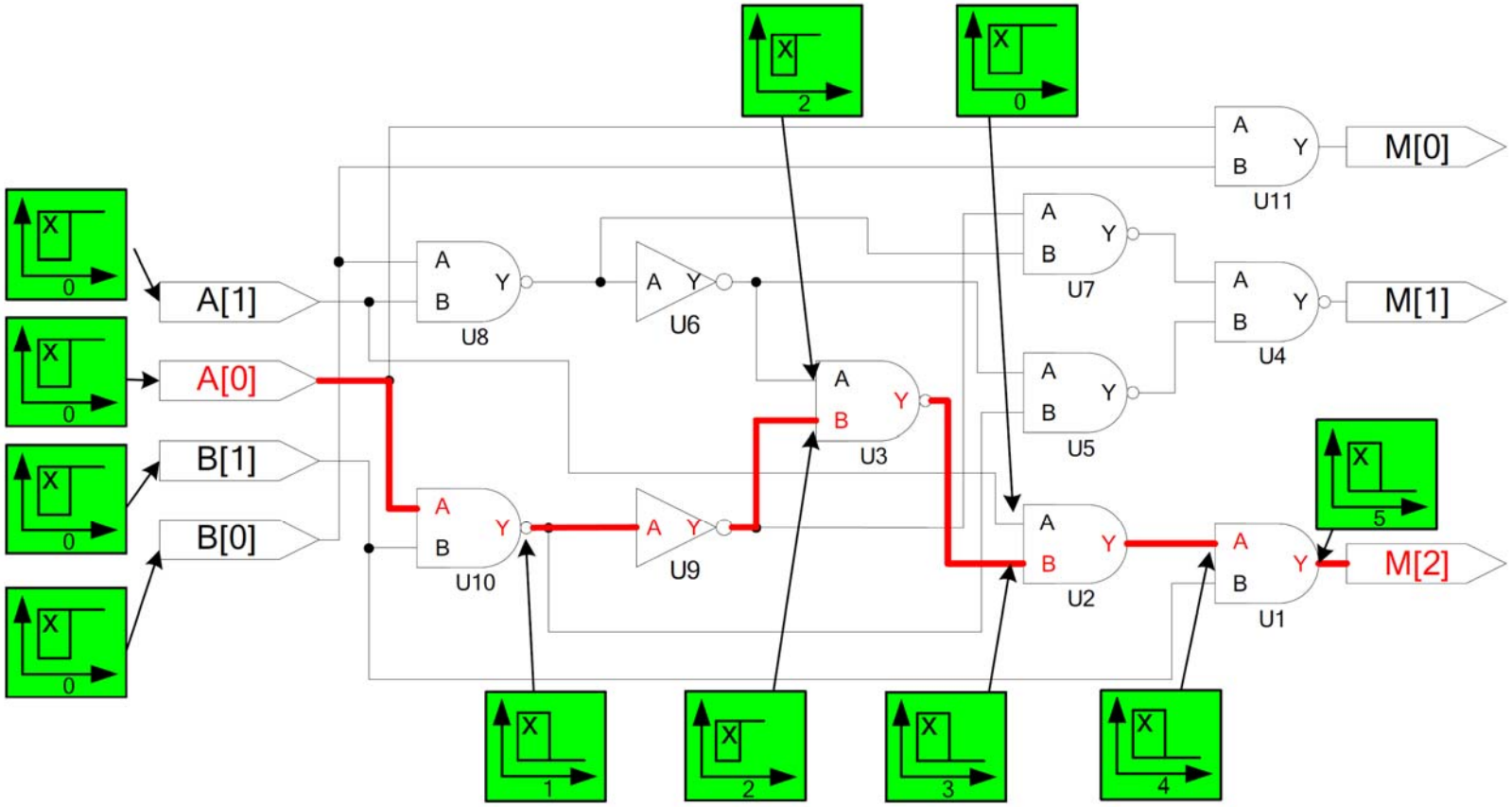


Figure 9: An example for floating mode simulation

Problem A3

- Example
  - Paths and slacks

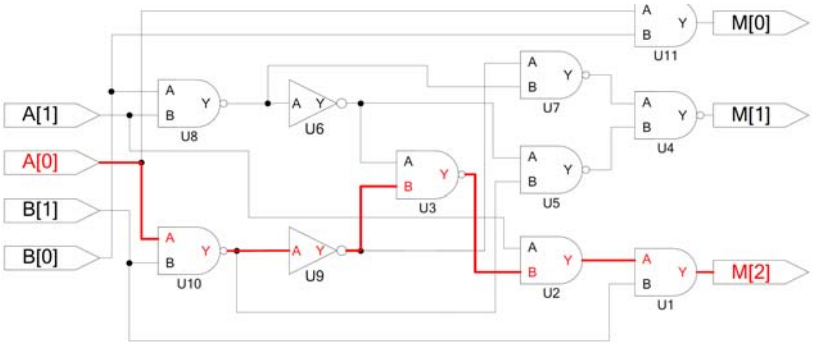


Figure 8: Schematic of example design

Table 2: All paths list of example design

Path #	Slack	Path Type	Path List
1	5	r	A[1]→U8/B→U8/Y→U6/A→U6/Y→U3/A→U3/Y→U2/B→U2/Y→U1/A→U1/Y→M[2]
2	5	f	A[1]→U8/B→U8/Y→U6/A→U6/Y→U3/A→U3/Y→U2/B→U2/Y→U1/A→U1/Y→M[2]
3	5	r	A[0]→U10/A→U10/Y→U9/A→U9/Y→U3/B→U3/Y→U2/B→U2/Y→U1/A→U1/Y→M[2]
4	5	f	A[0]→U10/A→U10/Y→U9/A→U9/Y→U3/B→U3/Y→U2/B→U2/Y→U1/A→U1/Y→M[2]
5	5	r	B[1]→U10/B→U10/Y→U9/A→U9/Y→U3/B→U3/Y→U2/B→U2/Y→U1/A→U1/Y→M[2]
6	5	f	B[1]→U10/B→U10/Y→U9/A→U9/Y→U3/B→U3/Y→U2/B→U2/Y→U1/A→U1/Y→M[2]
7	5	r	B[0]→U8/A→U8/Y→U6/A→U6/Y→U3/A→U3/Y→U2/B→U2/Y→U1/A→U1/Y→M[2]
8	5	f	B[0]→U8/A→U8/Y→U6/A→U6/Y→U3/A→U3/Y→U2/B→U2/Y→U1/A→U1/Y→M[2]
9	6	r	A[0]→U10/A→U10/Y→U9/A→U9/Y→U7/A→U7/Y→U4/A→U4/Y→M[1]
10	6	f	A[0]→U10/A→U10/Y→U9/A→U9/Y→U7/A→U7/Y→U4/A→U4/Y→M[1]
11	6	r	A[1]→U8/B→U8/Y→U6/A→U6/Y→U5/A→U5/Y→U4/B→U4/Y→M[1]
12	6	f	A[1]→U8/B→U8/Y→U6/A→U6/Y→U5/A→U5/Y→U4/B→U4/Y→M[1]
13	6	r	B[0]→U8/A→U8/Y→U6/A→U6/Y→U5/A→U5/Y→U4/B→U4/Y→M[1]
14	6	f	B[0]→U8/A→U8/Y→U6/A→U6/Y→U5/A→U5/Y→U4/B→U4/Y→M[1]
15	6	r	B[1]→U10/B→U10/Y→U9/A→U9/Y→U7/A→U7/Y→U4/A→U4/Y→M[1]
16	6	f	B[1]→U10/B→U10/Y→U9/A→U9/Y→U7/A→U7/Y→U4/A→U4/Y→M[1]
17	7	r	A[0]→U10/A→U10/Y→U5/B→U5/Y→U4/B→U4/Y→M[1]
18	7	f	A[0]→U10/A→U10/Y→U5/B→U5/Y→U4/B→U4/Y→M[1]
19	7	r	A[1]→U8/B→U8/Y→U7/B→U7/Y→U4/A→U4/Y→M[1]
20	7	f	A[1]→U8/B→U8/Y→U7/B→U7/Y→U4/A→U4/Y→M[1]
21	7	r	B[1]→U10/B→U10/Y→U5/B→U5/Y→U4/B→U4/Y→M[1]
22	7	f	B[1]→U10/B→U10/Y→U5/B→U5/Y→U4/B→U4/Y→M[1]
23	7	r	B[0]→U8/A→U8/Y→U7/B→U7/Y→U4/A→U4/Y→M[1]
24	7	f	B[0]→U8/A→U8/Y→U7/B→U7/Y→U4/A→U4/Y→M[1]
25	8	r	A[1]→U2/A→U2/Y→U1/A→U1/Y→M[2]
26	8	f	A[1]→U2/A→U2/Y→U1/A→U1/Y→M[2]
27	9	r	A[0]→U11/A→U11/Y→M[0]
28	9	f	A[0]→U11/A→U11/Y→M[0]
29	9	r	B[0]→U11/B→U11/Y→M[0]
30	9	f	B[0]→U11/B→U11/Y→M[0]
31	9	r	B[1] →U1/B→U1/Y→M[2]
32	9	f	B[1] →U1/B→U1/Y→M[2]

Problem A3

- Multi-core computing environment
  - CPU: Intel Xeon X7350 2.93G x 16
  - Memory: 128GB

Outline

- A1
- A2
- A3
- B1
- B2
- B3

Problem B1

- Numerical optimization on photo-mask – Model-based optical proximity correction
- Optical Proximity Correction (OPC) plays an important role in modern IC manufacturing
- Model-Based OPC (MBOPC) optimizes the mask pattern numerically based on a given lithography model
- MBOPC algorithm (to be implemented)
  - To achieve better printing fidelity for a given set of design patterns using a simplified lithography model

Problem B1

- Motivation

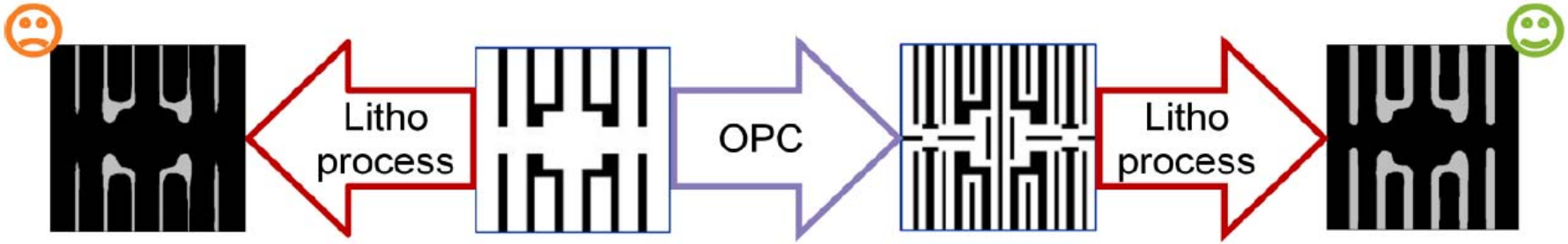


Figure: Challenge of modern lithography process ("OPC and image optimization using localized frequency analysis“, SPIE2002).

Problem B1

- OPC methods
  - Rule-based (fast)
  - Model-based (achieve better optimization)

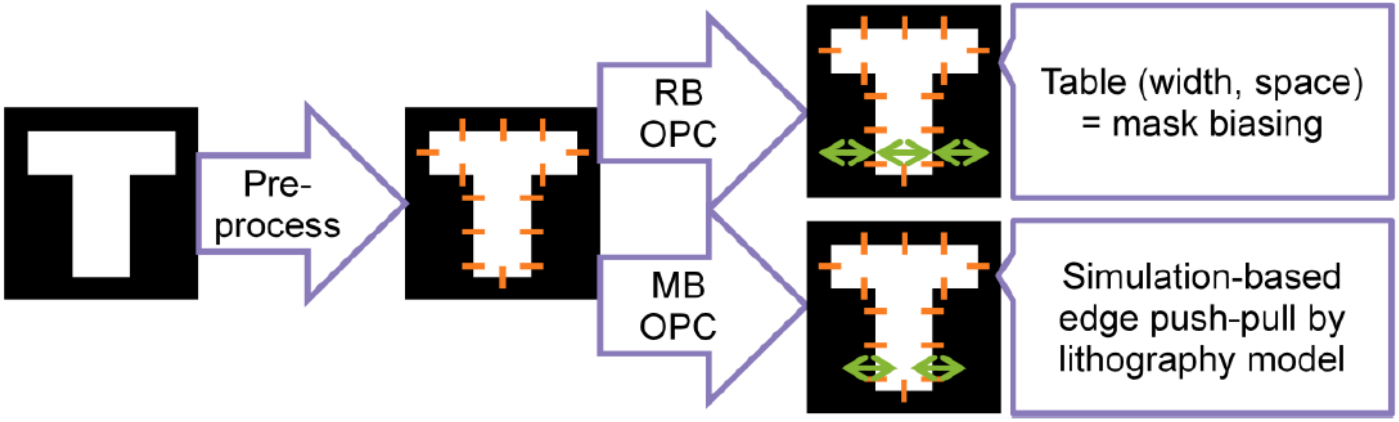


Figure: Illustration of data pre-processing and two traditional methods of OPC.

Problem B1

- Input format
  - Target design patterns given in CIF mask format
  - Information script file
    - Parameters of lithography process model (sum of coherent system, SOCS)
    - Manufacturability constraints of mask patterns
    - Information about interested evaluation box
- Output format
  - Optimized mask pattern in CIF format
  - Image result within interested evaluation box
  - Information about optimization process

Problem B1

- Example
  - Information script file

```
# Contents after the pound sign are comments. Format of information script file <.isf>, please load and parse it.
# The format is <token name> <token values>. The <token name> is always a string shorter than 31 characters.
# Parameters of lithography process model, it should NOT be modified.
PROCESS_NAME          FAKE    # Name of process model.
SOCS_KER_NUM          3       # Number of kernels used in SOCS.
SOCS_KER_AMBIT        500     # Effective proximity range of SOCS kernel (in nm).
SOCS_KER_PAR 1 0 0 0.01 2 1   # Parameters of SOCS kernel (see section 4 for detailed definition).
SOCS_KER_PAR 1 -1 0 0.01 2 1  # The format is <weight> <f> <g> <rho> <alpha> <beta> for each kernel.
SOCS_KER_PAR 1 1 0 0.01 2 1   # Note that kernels stored in matrix form can be complex-valued.
RESIST_STEP_THR       0.3     # Contour image threshold for constant resist step function.
# Mask manufacturability constraints, it should NOT be modified.
MASK_VAL_TYPE         BIN     # Type of mask value, the only option is binary-valued (0/1) mask.
                              # Continuous distribution or multi-valued masks are not allowed in this work.
MASK_SHAPE_TYPE       RECT    # Type of mask shape, the only option is Cartesian rectangular.
                              # No 45 degree nor curves are allowed in this work.
MASK_ADDRESSING       1       # DBU (in nm) of input mask data to some dedicate mask writer.
MASK_MIN_AREA         100     # Minimum area (in nm^2) of an isolated mask feature can be written out.
MASK_MIN_GRID         10      # Minimum size (in nm) of a manufacturable square grid for free form mask.
MASK_MIN_LENGTH       5       # Minimum length (in nm) of a mask segment for traditional edge biasing OPC.
# Information of interested evaluation box for full image result comparison, it should NOT be modified.
EVAL_BOX 512 512 0 0 511 511  # Matrix size (dimensionless) and corner coordinates (in nm).
                              # The format is <width> <height> <lower x> <lower y> <upper x> <upper y>.
# Optimization recipe and user-defined arguments that can be modified.
OPT_SCALE_FACT        0.5     # (required) User-defined optimization factor.
MAX_ITER_NUM          100     # (optional) Maximum iteration limit, an example of user-defined argument.
```

Problem B1

- Example
  - Manufacturability constraints

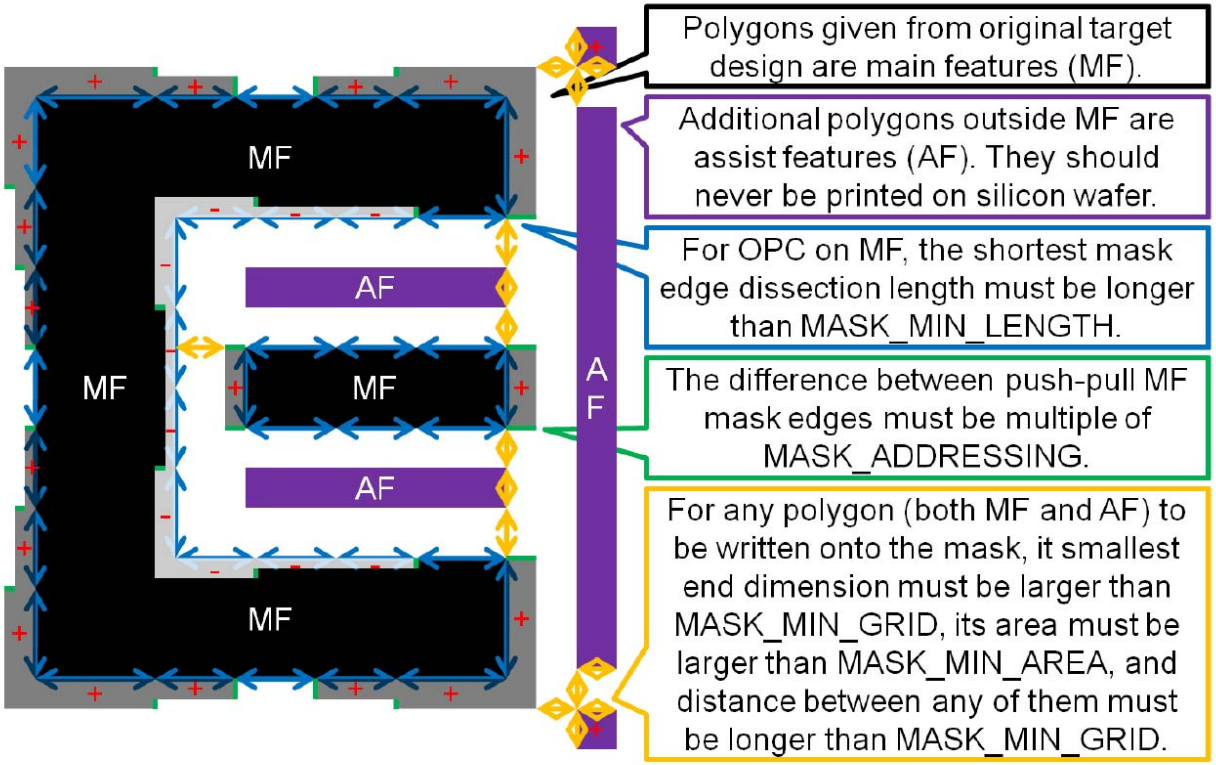


Figure: Illustration of mask manufacturability constraints.

Problem B1

- Lithography process model

$$image_{sim}(x,y) = \frac{\sum_{k}^{N_k} weight_k \left| kernel_k(x,y) * mask(x,y) \right|^2}{\sum_{k}^{N_k} weight_k \left| kernel_k(x,y) * 1 \right|^2} = \frac{\sum_{k}^{N_k} weight_k \left| F^{-1}\{F\{kernel_k(x,y)\} \cdot F\{mask(x,y)\}\} \right|^2}{\sum_{k}^{N_k} weight_k \left| F^{-1}\{F\{kernel_k(x,y)\} \cdot F\{1\}\} \right|^2}$$

$$kernel_k(x,y) = \frac{\cos(\rho_k^2(x^2+y^2))}{\beta_k + \alpha_k \rho_k^2 (x^2+y^2)} \exp\{-i2\pi(f_k x + g_k y)\rho_k\}$$

$weight_k, f_k, g_k, \rho_k, \alpha_k, \beta_k$ are given in ISF file for each kernel k

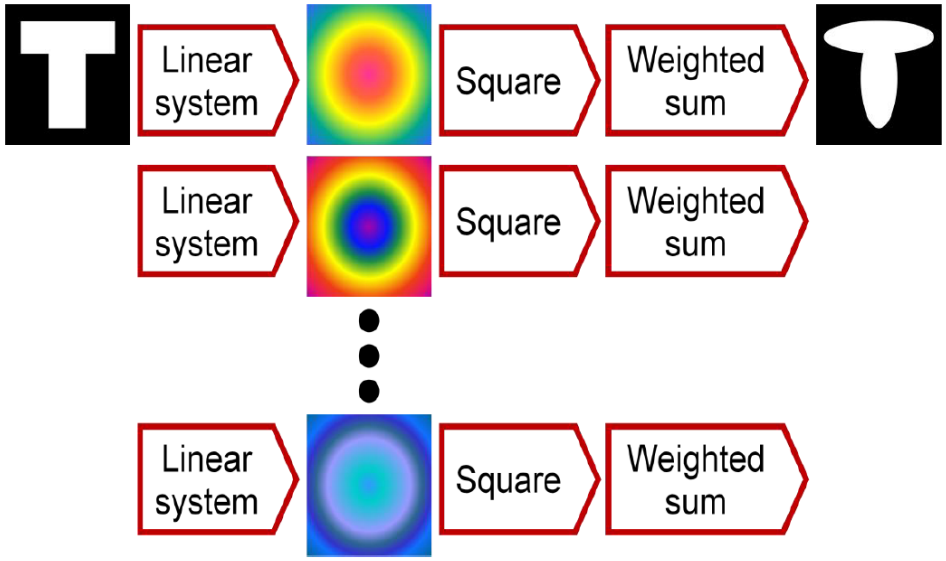


Figure: Illustration of a Sum of Coherent System (SOCS).

Problem B1

- Bitmap error as the measuring metric for both simulation accuracy and litho process fidelity
  - Bitmap error is defined in a certain interested evaluation box, specified in the script file

$$error_{BMP} = \frac{1}{N_x N_y} \sum_{x}^{N_x} \sum_{y}^{N_y} \left| step(image_{sim}(x,y)) - design(x,y) \right|$$

$$step(image) = \begin{cases} 1, image \geq threshold \\ 0, otherwise \end{cases}$$

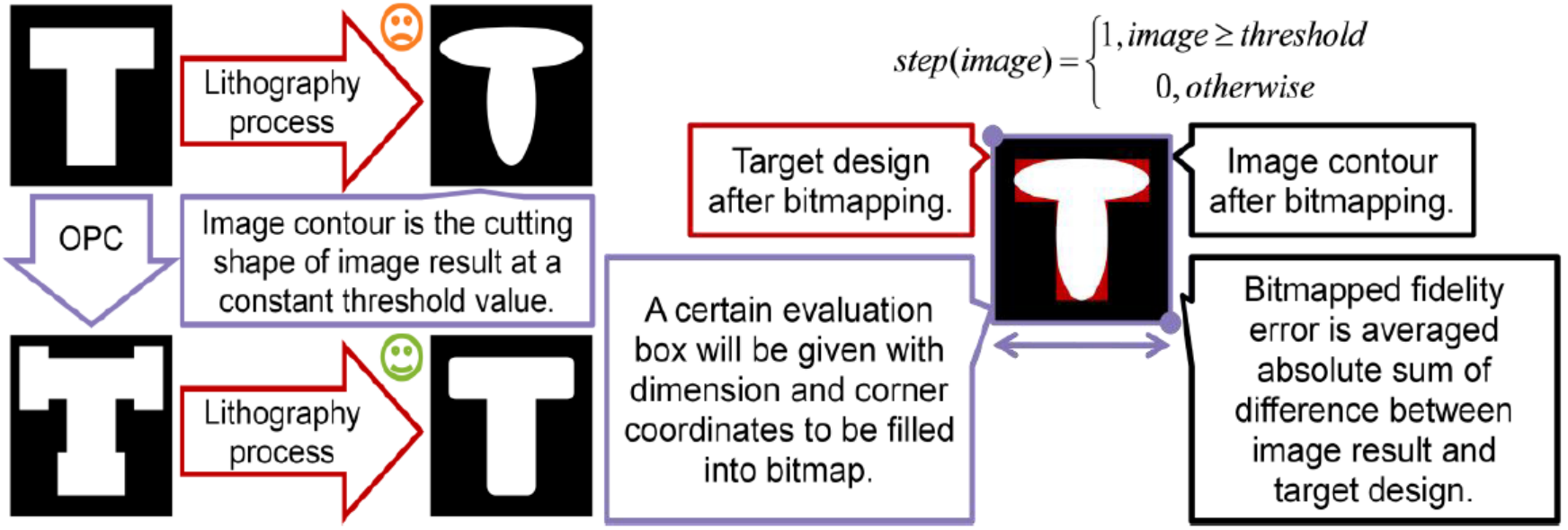


Figure: Illustration of an interested evaluation box and measurement of bitmap error.

Problem B1

- Caltech Intermediate Format (CIF) for pattern definition
  - Describe patterns with polygons

```
DS 1;
L L1;
P x1 y1 x2 y1 x2 y2 x1 y2;
DF 1;
```

  - E.g.,

```
P 0 0 50 0 50 10 10 10 10 100 0 100;
```

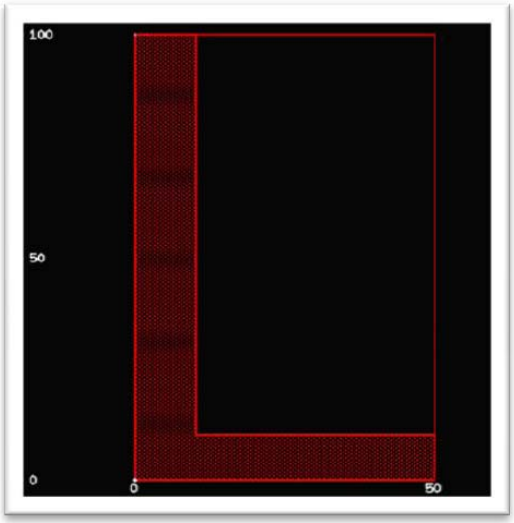


Figure: An example of L-shape polygon

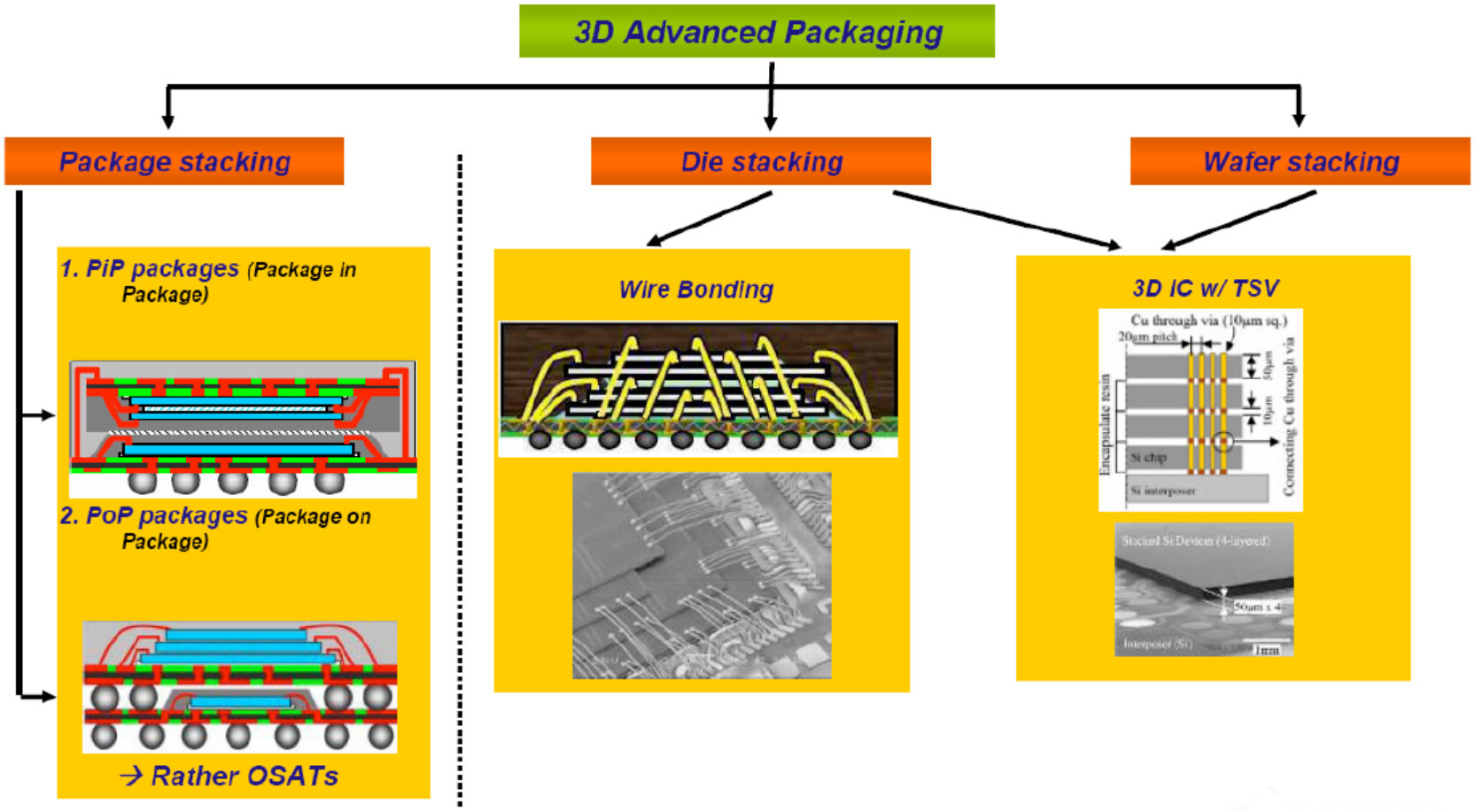
Outline

- A1
- A2
- A3
- B1
- B2
- B3

Problem B2

- 3D IC design partitioning with power consideration

Problem B2


3D Advanced Packaging
Package stacking
Die stacking
Wafer stacking
1. PiP packages (Package in Package)
2. PoP packages (Package on Package)
→ Rather OSATs
Wire Bonding
3D IC w/ TSV
Cu through via (10μm sq.)
20μm pitch
50μm
10μm
Encapsulate resin
Si chip
Si interposer
Connecting Cu through via
Stacked Si Devices (4-layered)
50μm x 4
Interposer (Si)
1mm

Problem B2

- Thermal and power issues in 3D IC

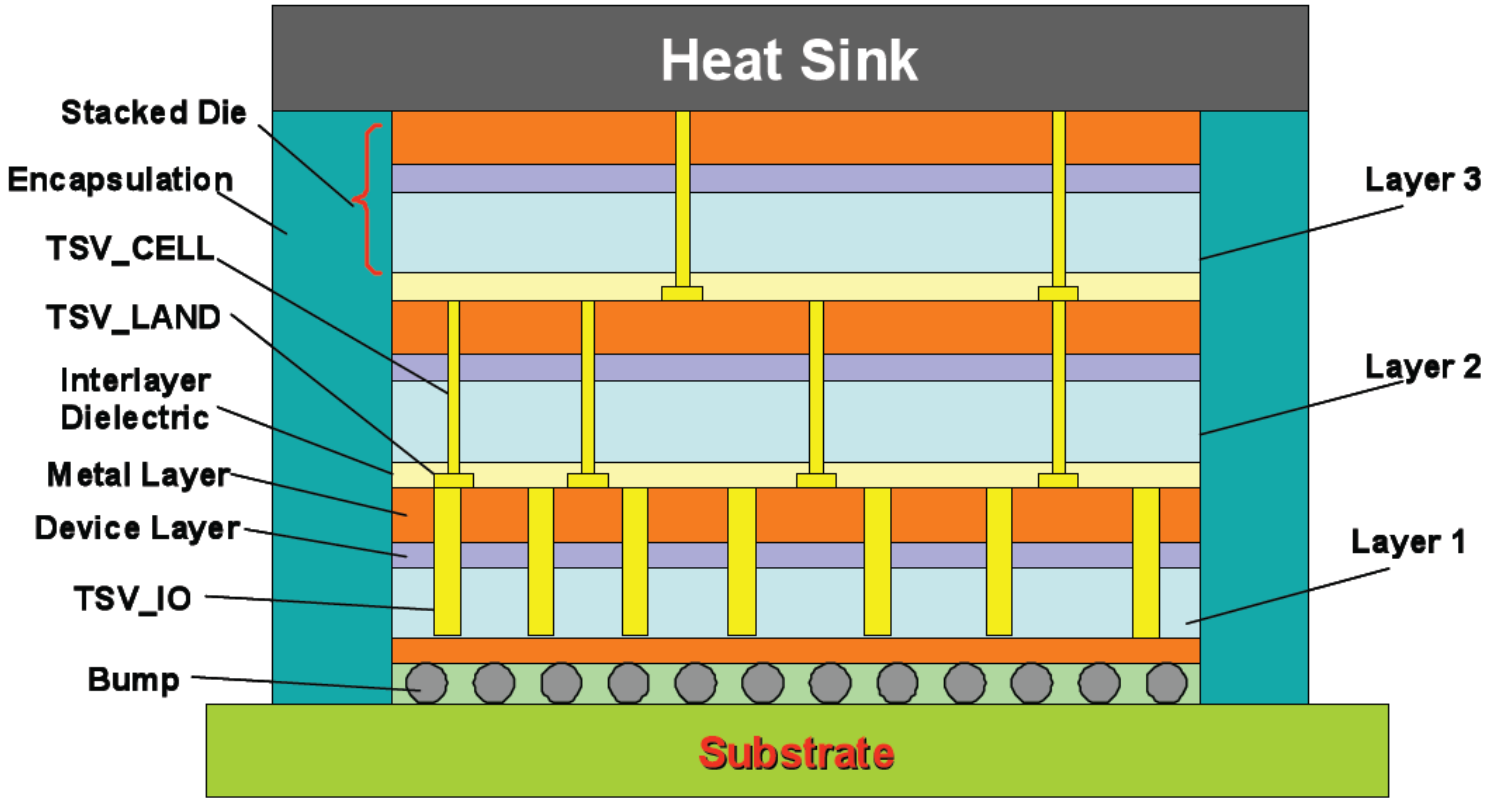


Figure 3 TSVs for layer interconnection

Problem B2

- Input format
  - A gate-level netlist (in Verilog)
  - Information file about the standard cells, standard I/O cells, TSV cells (for power and area calculation)
  - Design constraint file
- Output format
  - The gate-level netlist (in Verilog) for each layer in 3D IC
  - A report of basic statistics

Problem B2

- Power model

$$Power_{avg} = \sum_{i=1}^{L} P_{I,i} f_{I,i} + \sum_{i=1}^{L} \frac{1}{2} V_{dd\_core}^2 f_{I,i} C_{I,i} + \sum_{i=1}^{M} P_{DFF,i} f_{DFF,i} + \sum_{i=1}^{X} P_{I\_PAD,i} f_{I\_PAD,i} + \sum_{i=1}^{Y} P_{O\_PAD,i} f_{O\_PAD,i}$$

$$Power_{core} = \sum_{i=1}^{L} P_{I,i} f_{I,i} + \sum_{i=1}^{L} \frac{1}{2} V_{dd\_core}^2 f_{I,i} C_{I,i} + \sum_{i=1}^{M} P_{DFF,i} f_{DFF,i}$$

$$Power_{PAD} = \sum_{i=1}^{X} P_{I\_PAD,i} f_{I\_PAD,i} + \sum_{i=1}^{Y} P_{O\_PAD,i} f_{O\_PAD,i}$$

Problem B2

- Goal
  - Basic requirement
    - The output netlist must be equivalent to the original desing
    - Total power before and after partition must be the same
    - The power of each layer must satisfy the power constraints
  - Optimization consideration
    - Area overhead
    - TSV cell number
    - Program performance (in term of run time and memory usage)

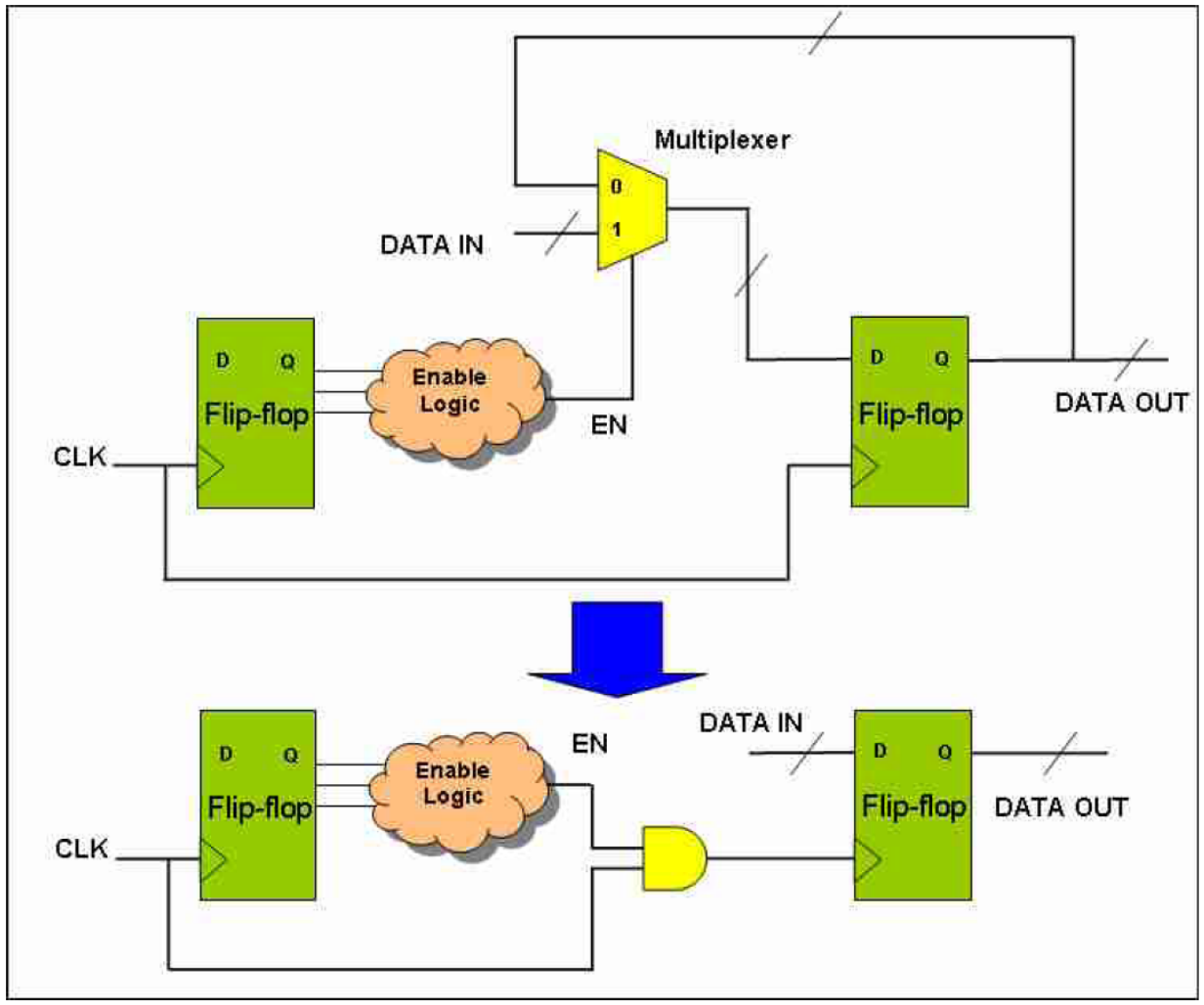
Outline

- A1
- A2
- A3
- B1
- B2
- B3

Problem B3

- Gated clock cloning for timing fixing
- Clock gating is an important technique in low power design
  - Prevent unnecessary signal switching in FFs
  - May cause setup time violation
- Timing fix by logic duplication (to be implemented)

Problem B3


Multiplexer
0
1
DATA IN
D
Q
Flip-flop
Enable
Logic
EN
CLK
D
Q
Flip-flop
DATA OUT
EN
DATA IN
D
Q
Flip-flop
Enable
Logic
CLK
D
Q
Flip-flop
DATA OUT

Problem B3

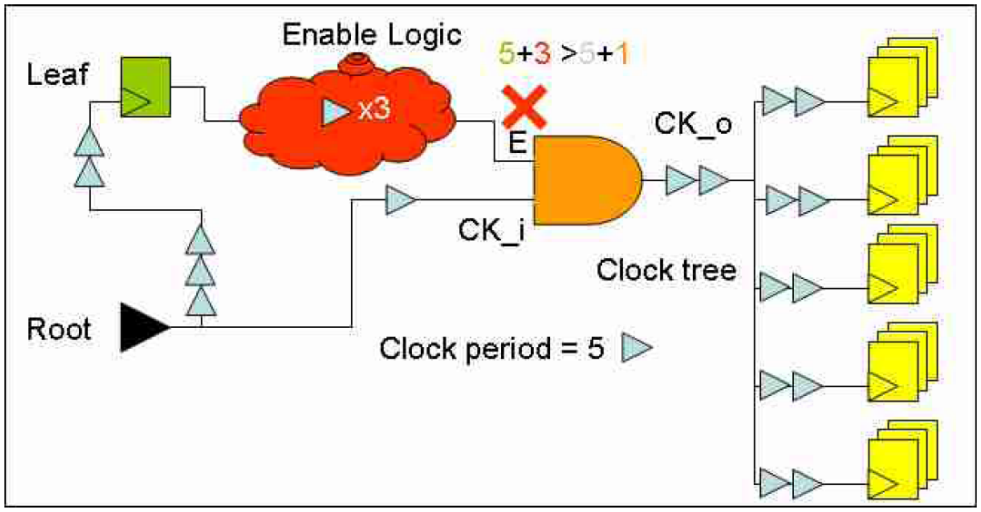


Figure 3: An example of a design with clock gates which violate setup time

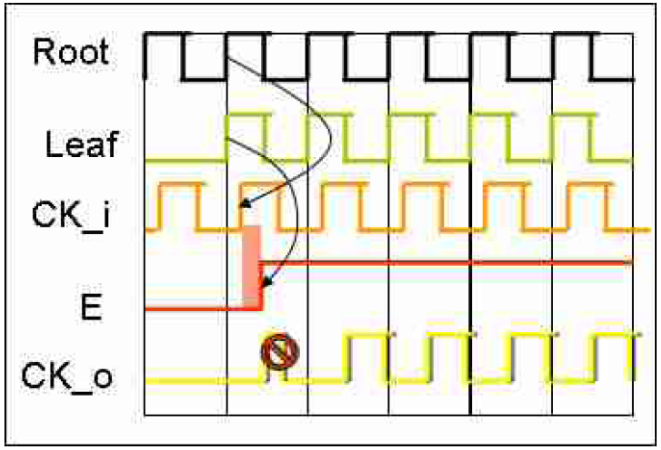


Figure 4: The waveform of the example in Fig 3.

Problem B3

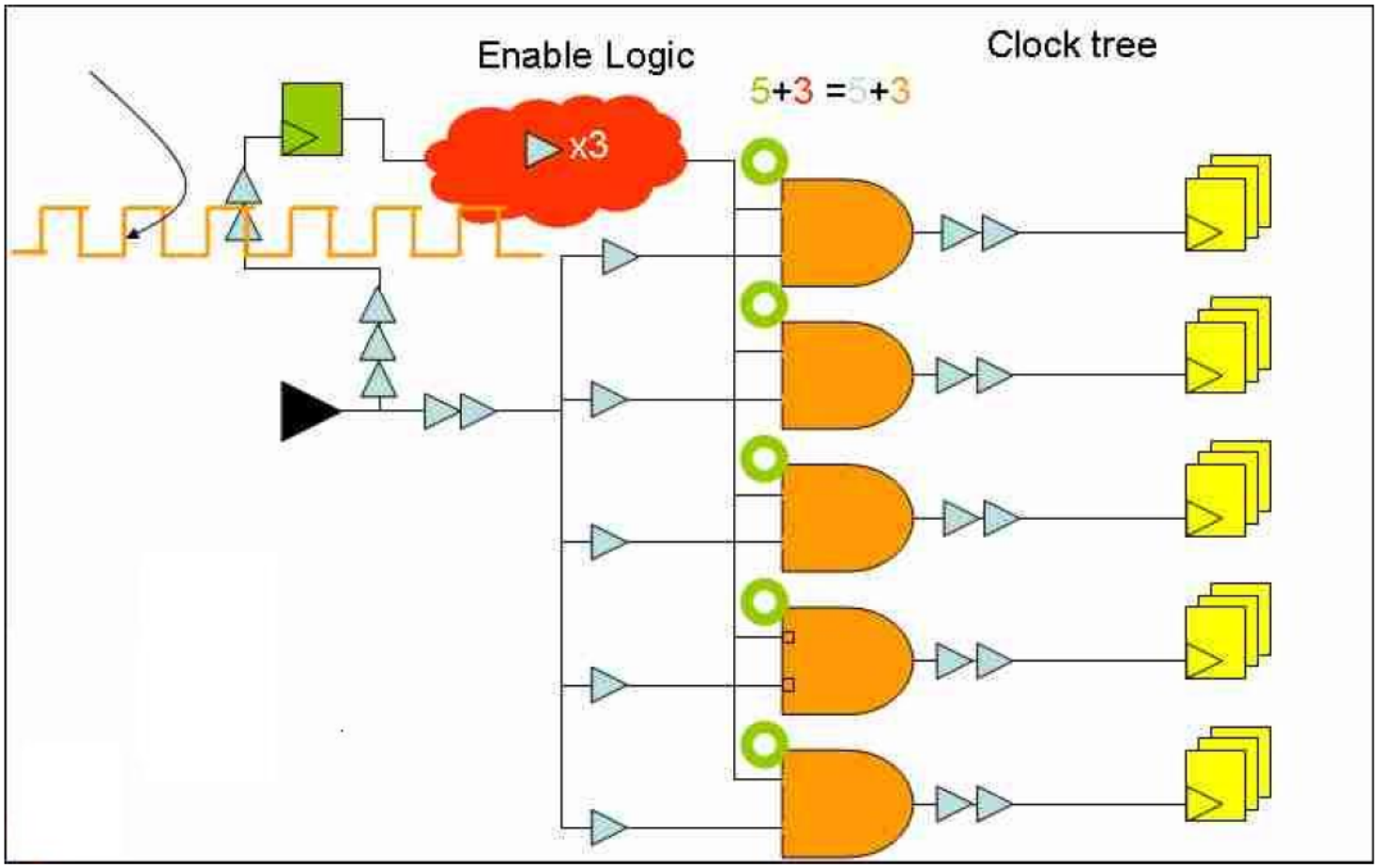


Figure 5: Clone the clock gate to satisfy the setup time

Problem B3

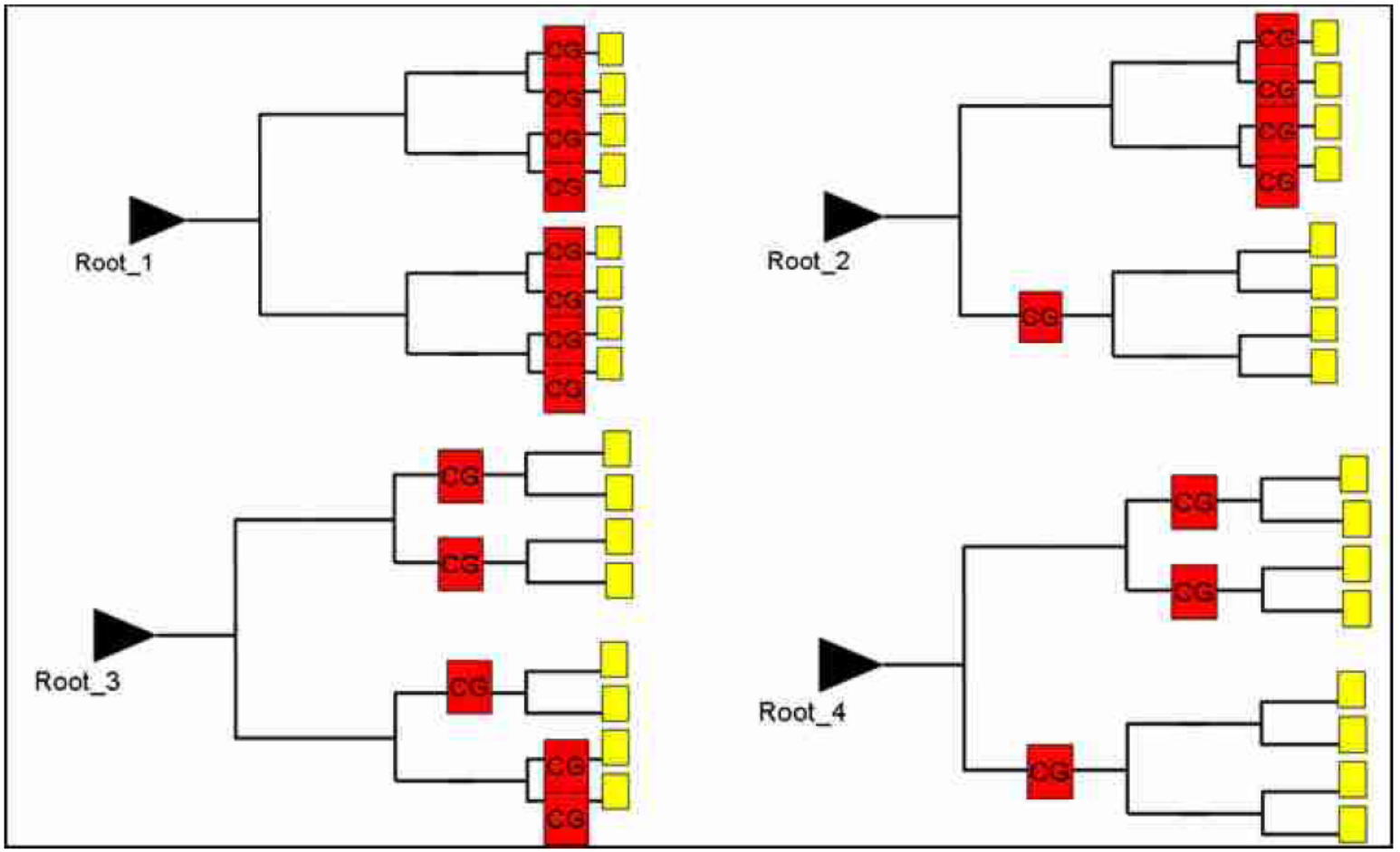


Figure 6: Possible clock gate location for clock tree.

Problem B3

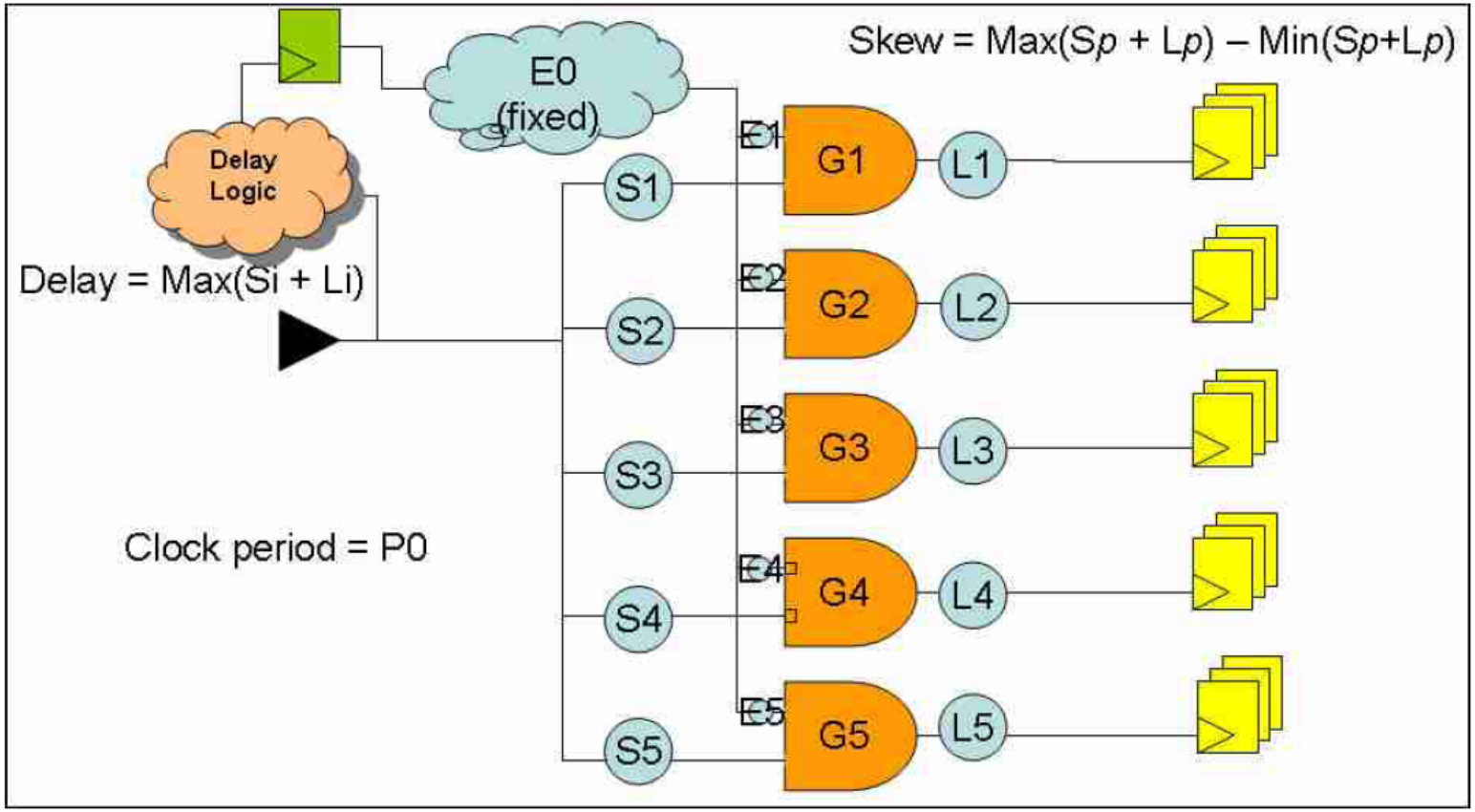


Figure 7: Example of constraint function and skew.

Problem B3

Setup Time Constraint: Max(*Si*+*Li*) + Enable Logic(*E0*+*Ej*) <= Clock Period (P0) + Delay (S*j*)

***Skew* = *Max*(*Sp* + *Lp*) – *Min*(*Sq*+*Lq*)**

Constraint Function: Max(*Si*+*Li*) + Enable Logic(E0+Ej) + *Skew* <= Clock Period (P0) + Delay (S*j*)

Problem B3

Length \ Buffer number	3	10	14	20
50	174	146	122	165
100	258	199	182	170
200	382	322	301	288
400	455	412	389	372
800	952	878	833	798
1600	1952	1825	1766	1612

Figure 8: Example of delay lookup table for logic ***Si*** and ***Ei***

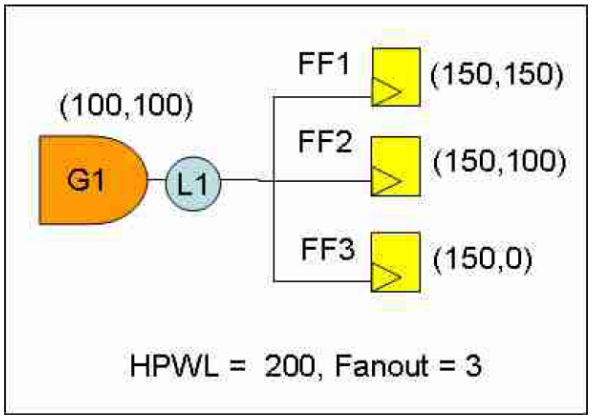


Figure 10: An example of logic L1.

HPWL \ Fan-out	3	10	14	20
50	222	285	357	411
100	432	511	698	802
200	712	985	1255	1658
400	1385	1985	2358	3125
800	2587	3945	4587	6012
1600	4989	7845	9121	12543

Figure 9: Example of delay lookup table for logic ***Li***

HPWL \ Fan-out	3	10	14	20
50	1	2	2	2
100	2	2	2	2
200	2	2	2	2
400	2	3	3	3
800	3	4	4	4
1600	4	5	5	6

Figure 11: Example of buffer number lookup table for logic Li

Problem B3

- Input format
  - Design section
    - Locations and names of clock source, enable logic, FFs, clock gating cells and nets)
  - Parameter section
    - Die size, clock period, enable logic delay, buffer weight and clock gating cell weight
- Output format
  - Design section
  - Value of objective function and execution time