Sungmin Cho
Solution Manual for VLSI CAD
October 8, 2006

Problem 1

(a) Encode the circuit as a SAT problem

As written in Larrabee's paper[2], one can transform a circuit element to a boolean equation(CNF form) using the idea that $P = Q$ is expressed as $(P \rightarrow Q) \cdot (Q \rightarrow P)$.

Logic	Input	Output	Expression
AND	X,Y	Z	$(\overline{Z} + X) \cdot (\overline{Z} + Y) \cdot (\overline{X} + \overline{Y} + Z)$
NAND	X,Y	Z	$(Z + X) \cdot (Z + Y) \cdot (\overline{X} + \overline{Y} + \overline{Z})$
OR	X,Y	Z	$(Z + \overline{X}) \cdot (Z + \overline{Y}) \cdot (\overline{Z} + X + Y)$
XOR	X,Y	Z	$(\overline{X} + Y + Z) \cdot (X + \overline{Y} + Z) \cdot (\overline{X} + \overline{Y} + \overline{Z}) \cdot (X + Y + \overline{Z})$
NOT	X	Z	$(X + Z) \cdot (\overline{X} + \overline{Z})$

Table 1: Translating Formulas into CNF

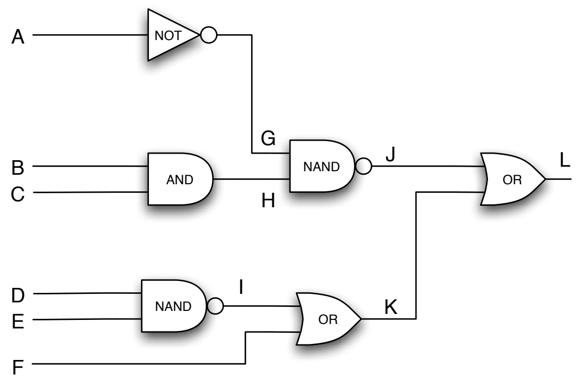


Figure 1: Original circuit

Using Table 1, the original circuit in Figure 1 is expressed as follows

$$\begin{aligned} res = &(L + \overline{J}) \cdot (L + \overline{K}) \cdot (\overline{L} + J + K) \cdot \\ &(J + G) \cdot (J + H) \cdot (\overline{J} + \overline{G} + \overline{H}) \cdot \\ &(K + \overline{I}) \cdot (K + \overline{F}) \cdot (\overline{K} + I + F) \cdot \\ &(G + A) \cdot (\overline{G} + \overline{A}) \cdot \\ &(\overline{H} + B) \cdot (\overline{H} \cdot C) \cdot (\overline{B} + \overline{C} + H) \\ &(I + D) \cdot (I + E) \cdot (\overline{I} + \overline{D} + \overline{E}) \cdot \end{aligned}$$

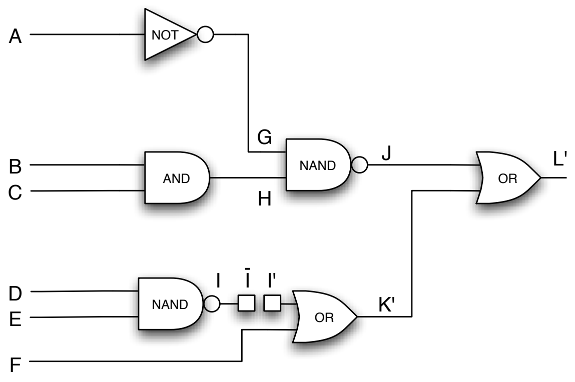


Figure 2: Fault circuit

(b) Fault circuit

$$
\begin{aligned}
res =& (L' + \bar{J}) \cdot (L + \overline{K'}) \cdot (\bar{L} + J + K') \cdot \\
& (K' + \overline{I'}) \cdot (K' + \bar{F}) \cdot (\overline{K'} + I' + F) \cdot \\
& (I') \cdot \\
& (J + G) \cdot (J + H) \cdot (\bar{J} + \bar{G} + \bar{H}) \cdot \\
& (G + A) \cdot (\bar{G} + \bar{A}) \cdot \\
& (\bar{H} + B) \cdot (\bar{H} \cdot C) \cdot (\bar{B} + \bar{C} + H) \\
& (I + D) \cdot (I + E) \cdot (\bar{I} + \bar{D} + \bar{E}) \cdot
\end{aligned}
$$

(c) Find a test input that exhibits the fault

Find a test vector is the same as finding a vector that satisfies a CNF formula of Figure 3.
And the formula is as follows.

$$
\begin{aligned}
res =& (L' + \bar{J}) \cdot (L + \overline{K'}) \cdot (\bar{L} + J + K') \cdot \\
& (K' + \overline{I'}) \cdot (K' + \bar{F}) \cdot (\overline{K'} + I' + F) \cdot \\
& (I') \cdot \\
& (J + G) \cdot (J + H) \cdot (\bar{J} + \bar{G} + \bar{H}) \cdot \\
& (K + \bar{I}) \cdot (K + \bar{F}) \cdot (\bar{K} + I + F) \cdot \\
& (J + G) \cdot (J + H) \cdot (\bar{J} + \bar{G} + \bar{H}) \cdot \\
& (G + A) \cdot (\bar{G} + \bar{A}) \cdot \\
& (\bar{H} + B) \cdot (\bar{H} \cdot C) \cdot (\bar{B} + \bar{C} + H) \\
& (I + D) \cdot (I + E) \cdot (\bar{I} + \bar{D} + \bar{E}) \cdot \\
& (\bar{L} + L' + Z) \cdot (L + \overline{L'} + Z) \cdot (L + L' + \bar{Z}) \cdot (\bar{L} + \overline{L'} + \bar{Z})
\end{aligned}
$$

- Activate - For the stuck 1 value, make an input that generates the reverse of that value. So D and E should be 1.

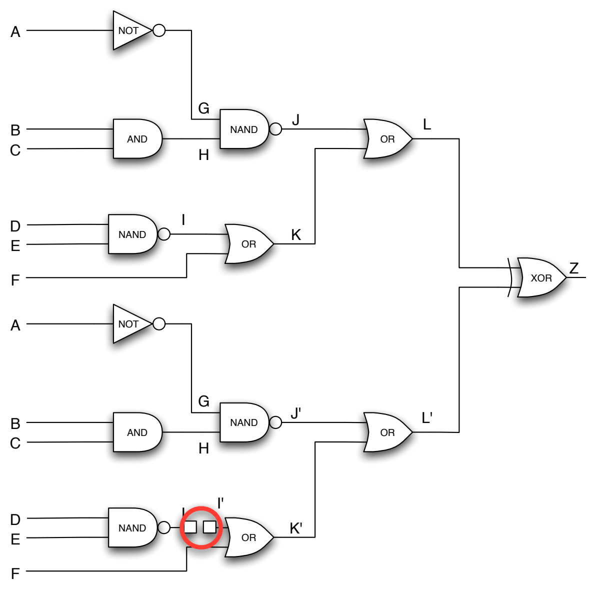


Figure 3: Fault circuit

- Propagate and Justify - For the error to propagate, give 1 input to the AND gate and 0 to the OR gate.

Using this method the test vector has three patterns. There are three patterns because there are 3 ways to make J equals 1.

A	B	C	D	E	F
1	X	X	1	1	1
X	0	X	1	1	1
X	X	0	1	1	1

Table 2: Test patterns

There are two main ways to solve the SAT problems([1]).

- Resolvent Algorithm.
- Search Based Algorithm.

Resolvent algorithm uses *unate variable rule* and *binate variable rule*. By applying those rules the SAT equation can be simplified to find values that evaluate the equation to 1.

Search Based Algorithm enumerates all input assignments to evaluate them to decide whether a formula is satisfiable.

Problem 2

(a) Draw retiming graph and compute W and D matrices

Figure 4 shows the graph of the circuit.

Using this graph D(delay) matrix is in Table 3.

And W(weight) matrix is in Table 4.

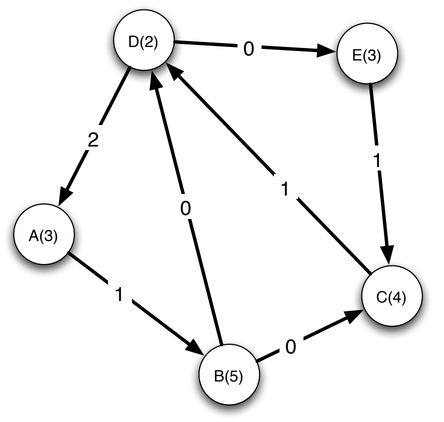


Figure 4: Retiming graph

	A	B	C	D	E
A	3	8	17	14	17
B	10	5	14	11	14
C	9	14	4	6	9
D	5	10	14	2	5
E	12	17	7	9	3

Table 3: D matrix

	A	B	C	D	E
A	0	1	1	1	1
B	2	0	0	1	1
C	3	4	0	1	1
D	2	3	3	0	0
E	4	5	1	2	0

Table 4: W matrix

(b) Minimum clock cycle

Before retiming path $B \rightarrow D \rightarrow E$ is the critical path with delay 10.

(c) Is retiming is possible with a clock period of 5?

In order to retiming one should build constraints out of delay matrix. Out of D matrix components, the value that exceeds 5 should be selected to make a contraint.

	A	B	C	D	E
A	3	**8**	**17**	**14**	**17**
B	**10**	5	**14**	**11**	**14**
C	**9**	**14**	4	**6**	**9**
D	5	**10**	**14**	2	5
E	**12**	**17**	**7**	**9**	3

Table 5: Components in D matrix that is more than 5 delays

And the constraint is expressed as $r(u) - r(v) \leq W(u,v) - 1$.
So, the constraint inequalities are as follows.

$$
\begin{gathered}
r(A) - r(B) \leq 0 \\
r(A) - r(C) \leq 0 \\
r(A) - r(D) \leq 0 \\
r(A) - r(E) \leq 0 \\
r(B) - r(A) \leq 1 \\
r(B) - r(C) \leq -1 \\
r(B) - r(D) \leq 0 \\
r(B) - r(E) \leq 0 \\
r(C) - r(A) \leq 2 \\
r(C) - r(B) \leq 3 \\
r(C) - r(D) \leq -1 \\
r(C) - r(E) \leq 0 \\
r(D) - r(B) \leq 2 \\
r(D) - r(C) \leq 2 \\
r(E) - r(A) \leq 3 \\
r(E) - r(B) \leq 4 \\
r(E) - r(C) \leq 0 \\
r(E) - r(D) \leq 1
\end{gathered}
$$

Out of these constraint sets, we can make another graph. For example, $r(A) - r(B) \leq 0$ can be expressed as Figure 5

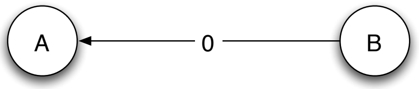


Figure 5: Constraint expressed in graph

By using this relation, the whole graph is Fig 6.

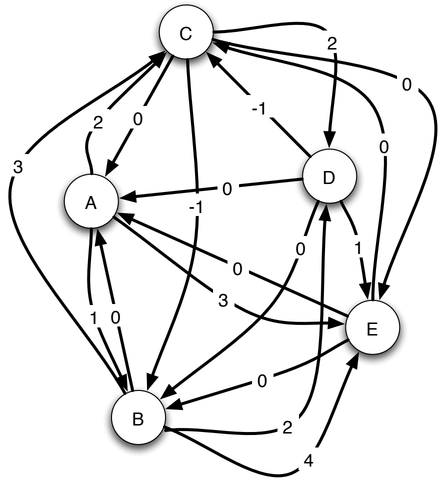


Figure 6: Constraint graph

Finally, using Bellman-Ford algorithm, one can solve this problem to find out how the registers are moved. I implemented a Bellman-Ford algorithm to solve this problem using Python programming language. The source code is listed in page 20.

Using the BF solver the result is in Table 11.

r(A)	r(B)	r(C)	r(D)	r(E)
0	1	2	3	2

Table 6: Result from BF solver

Problem 3

(a) Make BDD for 2 bit comparator

Table 7 shows the truth table for 2 bit comparator of $b > a$. Figure 7 shows BDD if ordered $a_1 < b_1 < a_0 < b_0$ and its simplification. The number of non-terminal is 5. Figure 8 shows BDD if ordered $a_1 < a_0 < b_1 < b_0$ and its simplification. The number of non-terminal is 9. In both cases, the dotted line implies 0 and solid line implies 1.

(b) BDD for n bit comparator

As one can see by rearranging the Table 7 into Table 8, If the sequence is ordered $a_{n-1} < b_{n-1} < \cdots < a_0 < b_0$. The 0's and 1's are collected in one parts so that the resulting BDD can be simplified. It is because the MSB is compared first, so the LSB's are always 0 if MSB of b is smaller than that of a and vice versa. So the circuit grows in **linear** fashion.

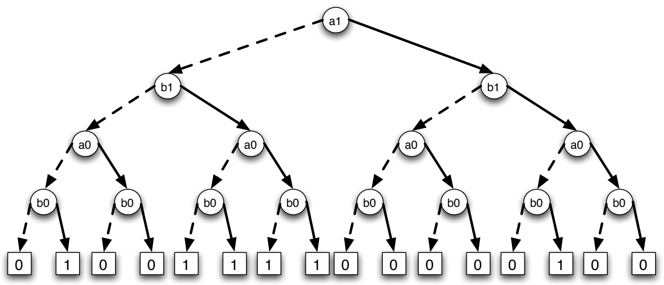
But if the sequence is ordered as $a_{n-1} < a_{n-2} < \cdots < b_1 < b_0$, the bits are shuffled so that the circuit cannot be simplified. So the circuit grows in **exponential** fashion.

(c) Implement XOR with BDD

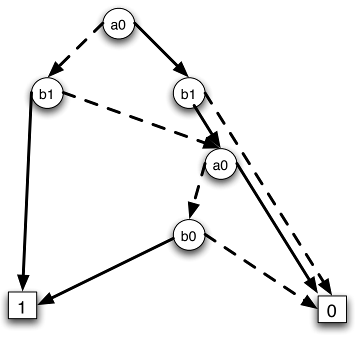
The equation is $x \oplus y = ITE(x, y', y)$ because of the relation $x \oplus y = x \cdot y' + x' \cdot y$ and $ITE(f, x, y) = x \cdot f + x' \cdot y$. So only 1 ITE operation is needed to construct XOR function.

a1	a0	b1	b0	out
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table 7: Truth Table for 2bit Comparator

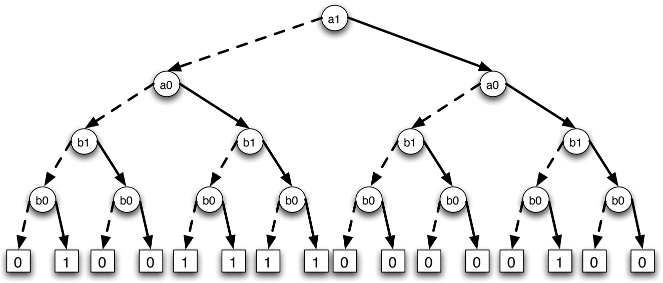


(a) BDD

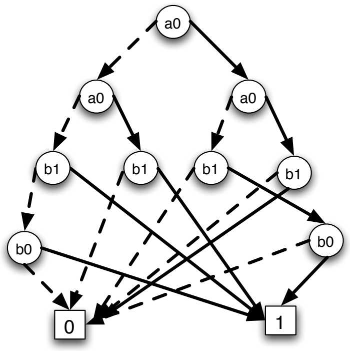


(b) simplified BDD

Figure 7: 2 bit comparator BDD and its simplification - case1



(a) BDD



(b) simplified BDD

Figure 8: 2 bit comparator BDD and its simplification - case2

a1	b1	a0	b0	out
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Table 8: Truth Table for 2bit Comparator when a and b are alternated

(d) Prove ITE decomposition

Let $Z = ITE(f, g, h)$ and let x be the top variable of function f, g and h.

$$\begin{aligned} Z &= x \cdot Z_x + x' \cdot Z_{x'} \\ &= x(f \cdot g + f' \cdot h)_x + x'(f \cdot g + f' \cdot h)_{x'} \\ &= x(f_x \cdot g_x + f'_x \cdot h_x) + x'(f_{x'} \cdot g_{x'} + f'_{x'} \cdot h_{x'}) \\ &= ITE(x, ITE(f_x, g_x, h_x), ITE(f_{x'}, g_{x'}, h_{x'})) \end{aligned}$$

So, $ITE(f, g, h) = x \cdot ITE(f_x, g_x, h_x) + x' \cdot ITE(f_{x'}, g_{x'}, h_{x'})$

Problem 4

(a) Write down δ for the state machine

The transition δ can be expressed as follows.

$$\begin{aligned} \delta(a,b,c,d,x,a^+,b^+,c^+,d^+) = {} & a\ \overline{b}\ \overline{c}\ \overline{d}\ \overline{x}\ \overline{a^+}\ b^+\ \overline{c^+}\ \overline{d^+} + \\ & \overline{a}\ b\ \overline{c}\ \overline{d}\ \overline{x}\ \overline{a^+}\ b^+\ \overline{c^+}\ \overline{d^+} + \\ & \overline{a}\ b\ \overline{c}\ \overline{d}\ x\ \overline{a^+}\ \overline{b^+}\ \overline{c^+}\ d^+ + \\ & \overline{a}\ \overline{b}\ \overline{c}\ d\ 1\ a^+\ \overline{b^+}\ \overline{c^+}\ \overline{d^+} + \\ & a\ \overline{b}\ \overline{c}\ \overline{d}\ x\ \overline{a^+}\ \overline{b^+}\ c^+\ \overline{d^+} + \\ & \overline{a}\ \overline{b}\ c\ \overline{d}\ 1\ \overline{a^+}\ \overline{b^+}\ c^+\ \overline{d^+} \end{aligned}$$

I tried to simplify the logic using Espresso, but this logic cannot be simplified further. And the transition can be expressed in *functional representation* as follows.

$$R_{k+1} = \delta(R_k, x) \tag{1}$$

The other way to express the transition is *relational representation* as follows.

$$T(R_{k+1}, R_k, x) = \begin{cases} 1, & R_{k+1} = \delta(R_k, x) \\ 0, & otherwise \end{cases} \tag{2}$$

(b) Deriving R_1, R_2 and so on

Using the expressions one can get the next state.

$$\begin{aligned} a^+ &= d \\ b^+ &= a\overline{x} + b\overline{x} \\ c^+ &= ax + c \\ d^+ &= bx \end{aligned}$$

So the T can be expressed as follows.

$$\begin{aligned} T = & (a,b,c,d,x,a^+,b^+,c^+,d^+) \\ & (\ \overline{a^+ \oplus d}\ )(\ \overline{b^+ \oplus (a\overline{x} + b\overline{x})}\ )(\ \overline{c^+ \oplus (ax + c)}\ )(\ \overline{d^+ \oplus bx}\ ) \end{aligned}$$

This equation checks, when there is a input x, if the result state is the same is expected. When the state is R_0 and the input is 0 the T is as follows.

$$R_1 = T(1,0,0,0,0,a^+,b^+,c^+,d^+) = (\overline{a^+ \oplus 0})(\overline{b^+ \oplus 1})(\overline{c^+ \oplus 0})(\overline{d^+ \oplus 0})$$

This equation means that $a^+b^+c^+d^+$ equals 0100. Again one can use the same method to get the R_2. In this case input x is 1 to go to the other state.

$$R_2 = T(0,1,0,0,1,a^+,b^+,c^+,d^+) = (\overline{a^+ \oplus 0})(\overline{b^+ \oplus 0})(\overline{c^+ \oplus 0})(\overline{d^+ \oplus 1})$$

(c) Working for VeriBackward

What the boss wants to express can be drawn by Figure 9 and Figure 10. As can be seen in the 9, one can progress from normal state to error state. It can be seen as desribed in Fig 10. That is, from the error state, one can go to the next step on and on. And if the state includes the initial state, the circuits are not equivalent.

With this method when we iterate the state, the state goes to the initial state which means stop. And I can check whether R_0 is contained in E_m by using the following 1 sentence.

With Initial state $I(s)$, and a fault state $F(s)$, if $(S \cdot F) \neq 0$, R_0 is contained in E_m.

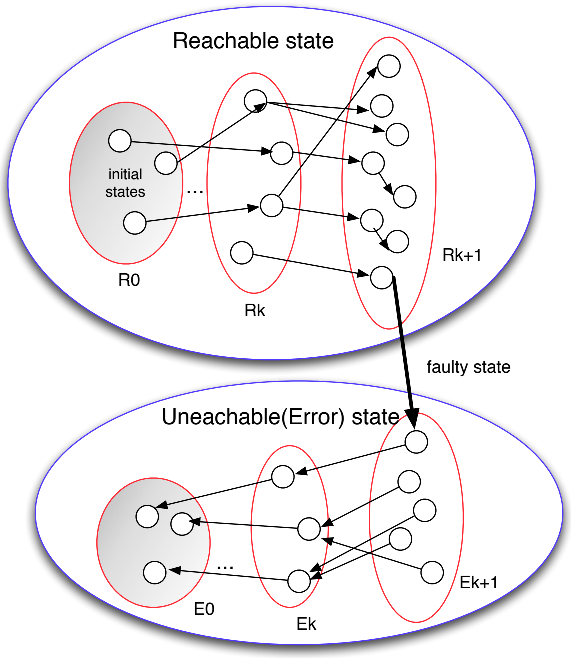


Figure 9: State Transition to Error State

Problem 5

(a) Find technology mapping

If one maps the logic gate to the circuit one by one, one can get the area of 15.0 as is seen in Table 9.

My algorithm for finding the best match is as follows.

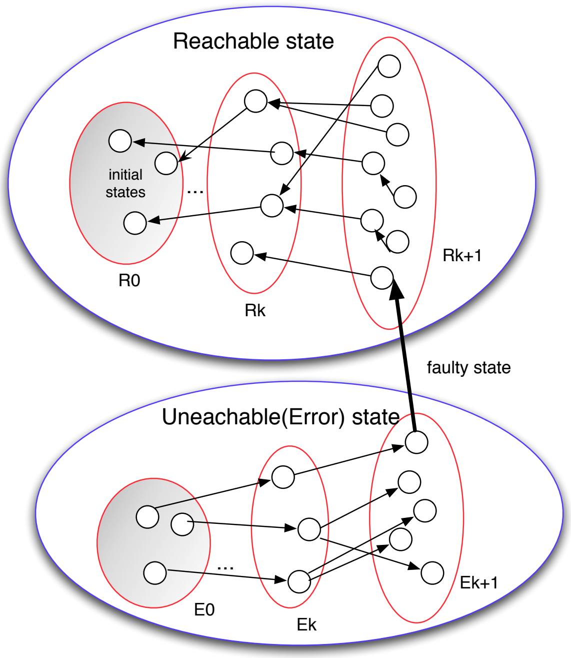


Figure 10: State Transition from Error State

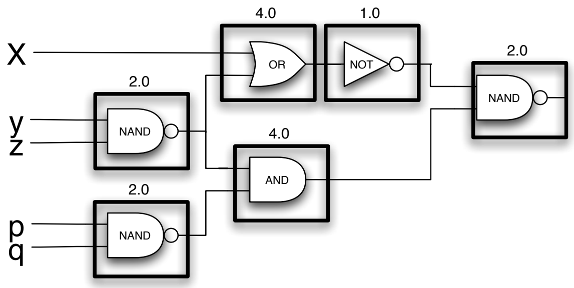


Figure 11: Result of the 1:1 mapping

Gate	Size	Number	Total Size
NAND2	2.0	3	6.0
AND2	4.0	1	4.0
OR2	4.0	1	4.0
INV	1.0	1	1.0
			15.0

Table 9: 1:1 mapping result

```
Step 1. Transform each library in NAND2 form.
Step 2. Transform each logic gate in NAND2 form and normalize.
Step 3. If there is a fan-out isolate the circuit
Step 4. Calculate the circuit
for each logic
  loop
    Mapping each circuit to minimize the area
    Calculate at each step
  end loop
end each circuit
Step 5. attach the isolated circuit to each circuit and go to for loop and recalculate
Step 6. If there is no not gate between the gate insert the 2 not gate and recalculate
```

Step1

For Step1 the library circuit should be transformed into orthogonal AND circuit. The transform rule is shown in Figure 12.

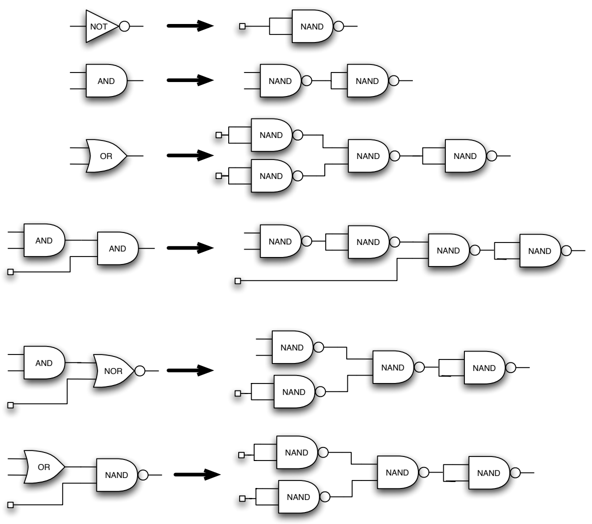


Figure 12: Transform the library

Step2

For Step2 the input circuit is transformed into orthogonal AND circuit. The transformed result is shown in Figure 13.

Step3

For Step3 If the circuit has fan-out separate the circuit. The transformed result is shown in Figure 14.

Step6

For Step6, if there is no circuit between the gates, insert 2 not gates to see if it is possible to transform into AOI or OAI circuit. The transformed result is shown in Figure 15.

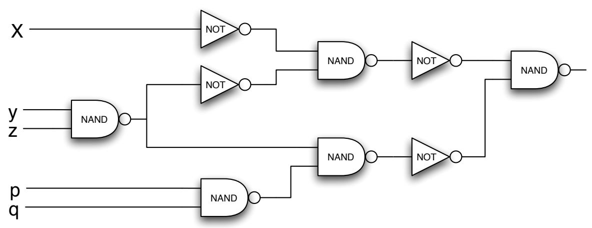


Figure 13: Transform the circuit
/Users/smcho/Desktop/2nd midterm exam/p5.tex

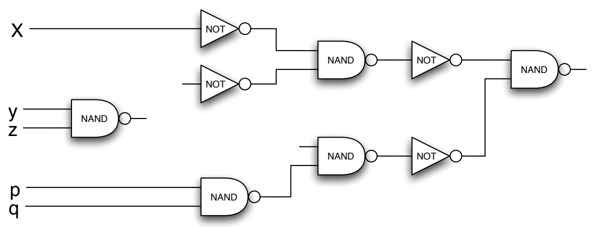


Figure 14: Remove the fan-out

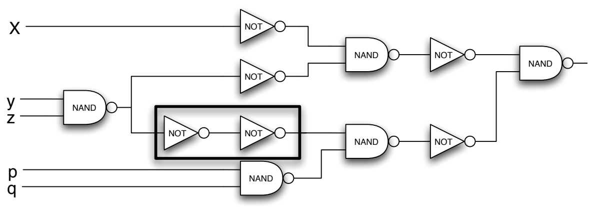


Figure 15: Insert 2 not gates

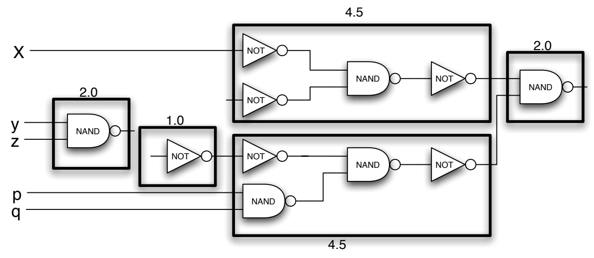


Figure 16: Final result

Gate	Size	Number	Total Size
AOI3	4.5	1	4.5
OAI3	4.5	1	4.5
NAND	2.0	2	4.0
NOT	1.0	1	1.0
			14.0

Table 10: Final result

(b) Result

With the example circuit, after going through all the algorithm and compare the result, one can get the following result. Mapped result is in Figure 16 and calculated size is in Table 10. In this case, by inserting 2 not gates and separating the circuit into 3 parts, I can get the best result.

Problem 6

(a) Survey

Before solving the problem, it is necessary to see when those four cases occur. I used the book [4] and [3] as a reference.

Blocked

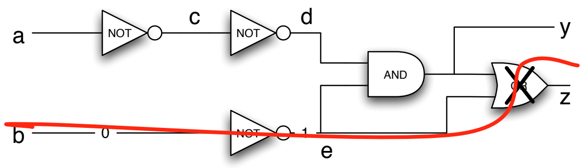


Figure 17: Blocked path

The path is blocked when one of the gate in the path sends only 1 or 0. For example, if AND gate has input 0, it always sends 0. And if OR gate has input 1, it sends 1 always. In both cases the path is blocked. Figure 17 shows the path that is blocked. If $b = 0$ path $b \rightarrow e \rightarrow z$ is blocked, where as path $a \rightarrow c \rightarrow d \rightarrow y$ becomes critical path.

Statically co-sensitizable

Consider a path $P = (v_{x0}, v_{x1}, \cdots, v_{x_m})$, A vector is *statically co-sensitizes* a path to 1 (or to 0) if input $x_m = 1$ (or 0) and if $v_{x_{m-1}}$ has a controlling value whenever v_{x_i} has a controlled value.

As an example with Figure 18, Path $a \rightarrow d \rightarrow g \rightarrow o$ is statically co-sensitizable to 0 by input $a = 0, b = 0, c = 0$. It is because the gates with controlled value(o, d) have a controlling input along the path ($g = 0, a = 0$).

Statically sensitizable

Figure 19 shows a statically sensitizable path with test vector $b = 1, c = 1, d = 1$. The algorithm for finding test vector is as follows. Actually, it is same as 'Fault propagation' algorithm for finding

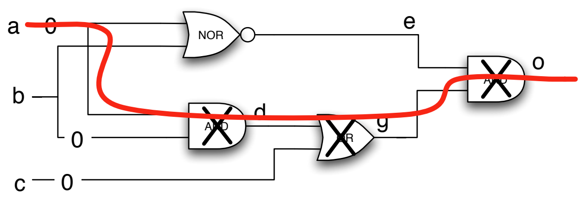


Figure 18: Statically co-sensitizable path

stuck-at error.

Step 1. Identify the path.
Step 2. Identify input vector.
Step 2-1. With AND gate, give 1 to make a path. Find vector to send 1 from the previous gate.
Step 2-2. With OR gate, give 0 to make a path. Find vector to send 0 from the previous gate.

At Figure 19, o is AND gate. So the input is 1. In order to send 1 from g, input vector should be 1.

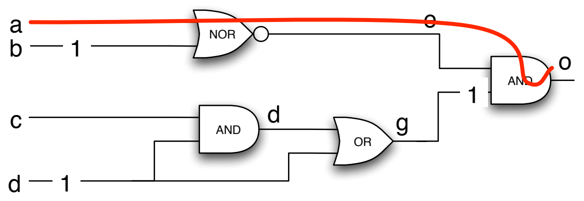


Figure 19: Statically sensitizable path

HFRPDFT

Statically sensitizable circuit can be simplified with the following rule.

- With AND gate, if the input is 1, the gate becomes buffer.
- With OR gate, if the input is 0, the gate becomes buffer.

After simplification, if the path to the input gate has the same delay. We can identify the path as HFRPDFT. Figure 20 shows 2 circuits after simplification. Circuit in upper part is not HFRPDFT but circuit in lower part is HFRPDFT. The HFRPDFT path has 2 test vectors, the first vector makes result 0 and the second vector makes the result 1.

(b) Identify each path

Path1,2,3,4,5

Figure 21 shows path 1,2,3,4,5 and its simplified circuit when input vector is $d = 0$ to propagate the signal through the path. The five paths are blocked.

Path 6,7

Figure 22 shows path 6,7 when input vector is $d = 0$ to propagate the signal through the path. The two paths are blocked.

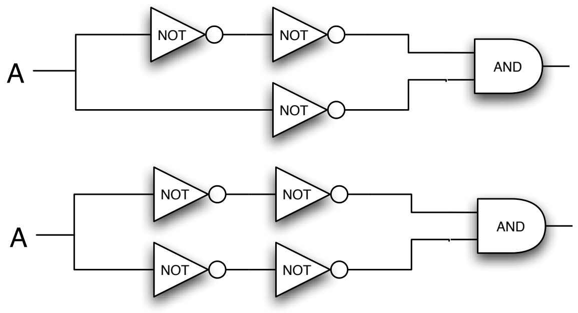


Figure 20: After simplification

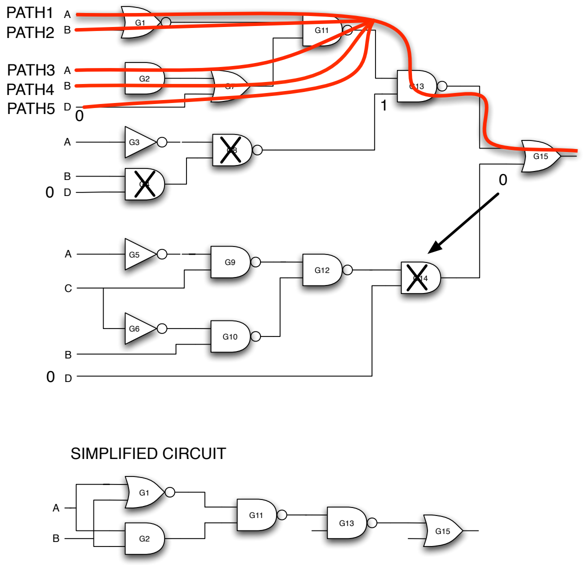


Figure 21: Blocked path

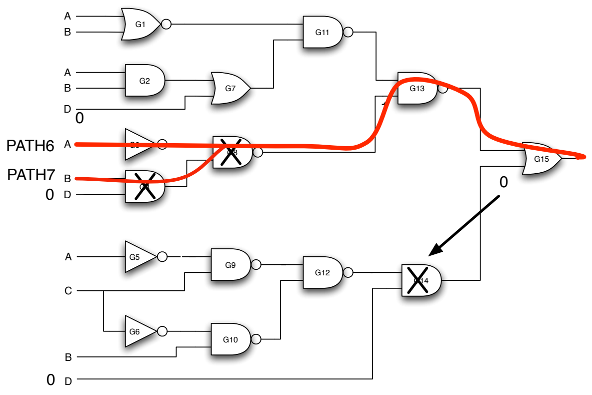


Figure 22: Path 6,7

Path 8

Figure 23 shows path 8 and its simplified circuit when input vector is $a = 0, b = 1, c = 0$ to propagate the signal through the path.

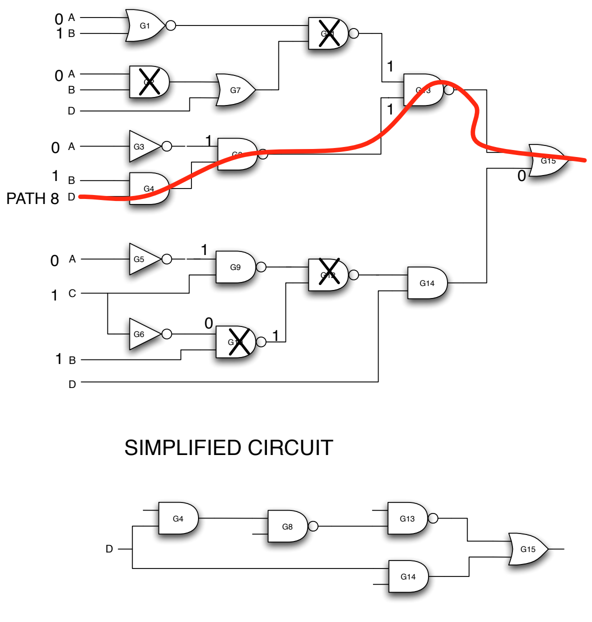


Figure 23: Path 8

Path 9

Figure 24 shows path 9 and its simplified circuit when input vector is $b = 1, c = 1$ to propagate the signal through the path.

Path 10

Figure 25 shows path 10 and its simplified circuit when input vector is $a = 0, b = 0, d = 1$ to propagate the signal through the path. This path is HFRPDFT with test vector $< (0,0,0,1),(0,0,1,1) >$.

Path 11

Figure 26 shows path 11 and its simplified circuit when input vector is $a = 0, b = 1, d = 1$ to propagate the signal through the path.

Path 12

Figure 27 shows path 12 and its simplified circuit when input vector is $a = 1, c = 0, d = 1$ to propagate the signal through the path. This path is HFRPDFT with test vector $< (1,0,0,1),(1,1,0,1) >$.

Path 13

Figure 28 shows path 13 and its simplified circuit when input vector is $a = 0, b = 1, d = 0$ to propagate the signal through the path.

(c) Result

Table 11 shows the identified path.

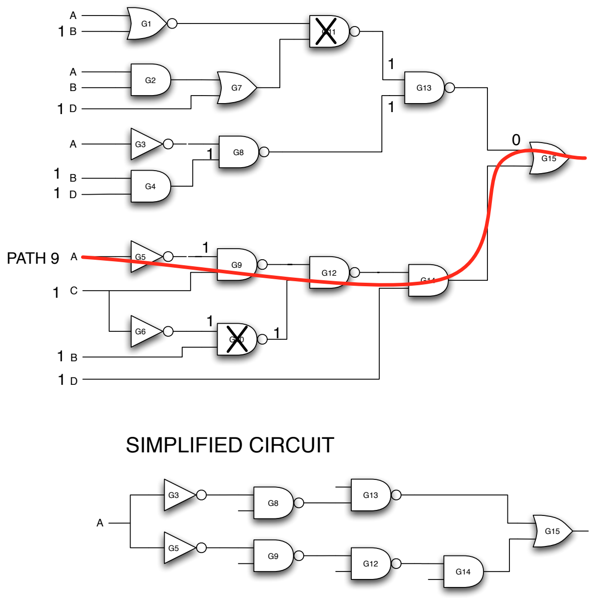


Figure 24: Path 9

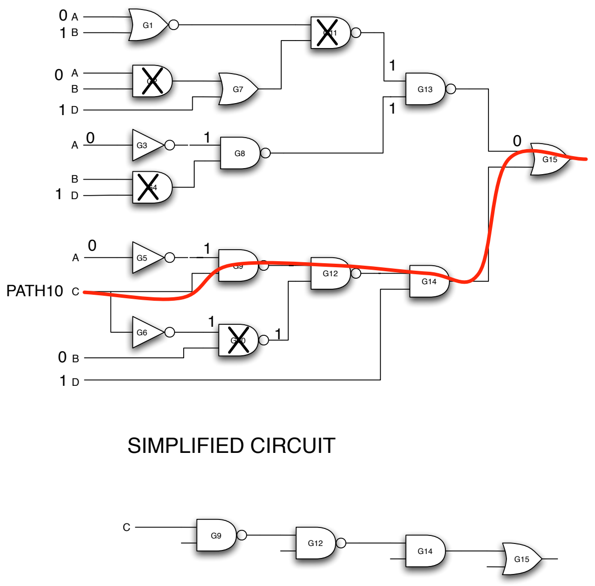


Figure 25: Path 10

Path	Condition	Test Vector
1,2,3,4,5,6,7	Blocked	
8, 9,11,13	Statically sesitizable	
10	HFRPDFT	$<(0,0,0,1),(0,0,1,1)>$
12	HFRPDFT	$<(1,0,0,1),(1,1,0,1)>$

Table 11: Result

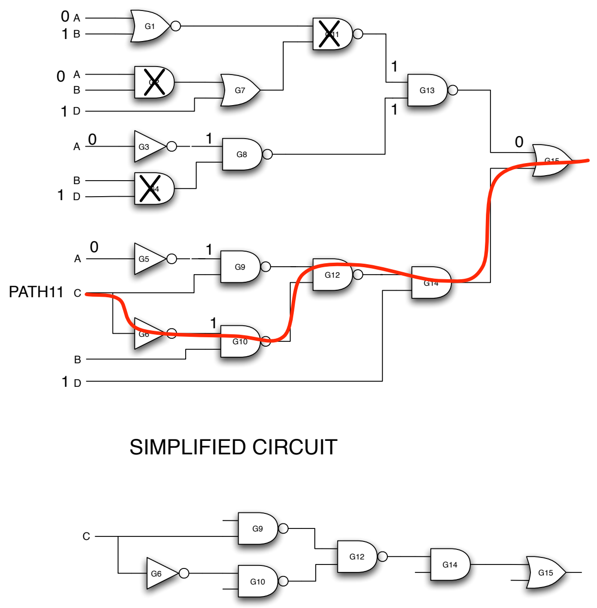


Figure 26: Path 11

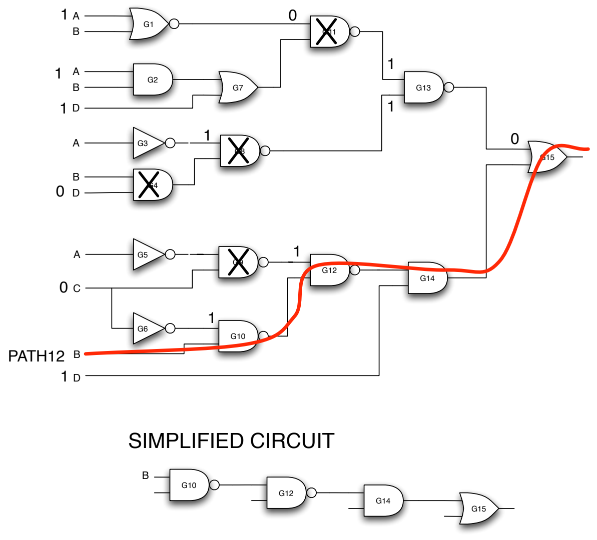


Figure 27: Path 12

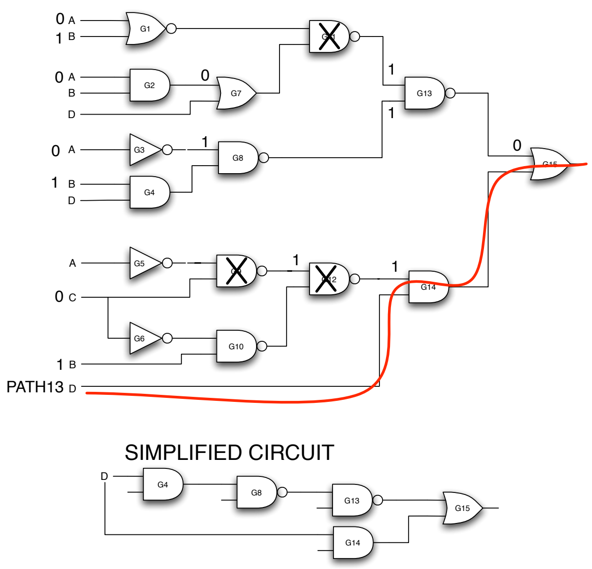


Figure 28: Path 13

A. Bellman-Ford algorithm implementation

```
# maxpath BIG <
# minpath SMALL >
__INF__= 400000
class Vertex:
  def __init__(self, number_):
    self.number  = number_;
    self.list = []
  def add(self, edge_):
    self.list.append(edge_);
  def printme(self):
    print "--------------------\n";
    for edge  in self.list:
      edge.printme();
    print "--------------------\n";

class VertexList:
  def __init__(self):
    self.list = {}
  def add(self, name, vertext):
    self.list[name] = vertext;
  def length(self):
    return len(self.list);
  def __getitem__(self, name_):
    return self.list[name_];

class Edge:
  def __init__(self, start, stop, weight):
    self.start  = start;
    self.stop  = stop;
    self.weight = weight;
  def printme(self):
    print "START:%d STOP:%d - WEIGHT:%d\n"% (self.start, self.stop, self.
                                                  weight);

def BF(vertexList_):
  number = vertexList_.length();
  distance = {}

  # initialize
  for x in range(number):
    distance[x] = __INF__;
  distance[0] = 0;
  count = 0;

  # First
  v = vertexList_['v0'];
  S1 = []; S1.append(v);
  S2 = [];

  while (count <= number and len(S1) > 0):
    # Get every vertex
    for vertex in S1:
      # get every edge connected with the vertex
      for edge in vertex.list:
        vj = "v%d"% edge.stop;
        if (distance[edge.stop] > distance[edge.start] + edge.weight):
          distance[edge.stop] = distance[edge.start] + edge.weight;
          S2.append(vertexList_[vj])

    S1 = S2;
    S2 = []
    count += 1
    printlist(distance);
    print("\n");

  if (count > number):
    print("ERROR: Positive cycle");

  return distance

def printlist(list_):
  val = "";
  for i in list_:
     val = "%s %d"% (val,list_[i])
  print val

if __name__== "__main__":
  # Edge
  e1 = Edge(1,0,0);
  e2 = Edge(2,0,0);
  e3 = Edge(3,0,0);
  e4 = Edge(4,0,0);

```

```
e5  = Edge(0,1,1);
e6  = Edge(2,1,-1);
e7  = Edge(3,1,0);
e8  = Edge(4,1,0);

e9  = Edge(0,2,2);
e10  = Edge(1,2,3);
e11  = Edge(3,2,-1);
e12  = Edge(4,2,0);

e13  = Edge(1,3,2);
e14  = Edge(2,3,2);

e15  = Edge(0,4,3);
e16  = Edge(1,4,4);
e17  = Edge(2,4,0);
e18  = Edge(3,4,1);

# Vertex
v0  = Vertex(0); #A
v0.add(e5); v0.add(e9);v0.add(e15);
v1  = Vertex(1); #B
v1.add(e1); v1.add(e10); v1.add(e13); v1.add(e16);
v2  = Vertex(2); #C
v2.add(e2); v2.add(e6); v2.add(e14); v2.add(e17);
v3  = Vertex(3); #D
v3.add(e3); v3.add(e7); v3.add(e11); v3.add(e18);
v4  = Vertex(4); #E
v4.add(e4); v4.add(e8); v4.add(e12);

vlist  = VertexList();
vlist.add('v0',v0);
vlist.add('v1',v1);
vlist.add('v2',v2);
vlist.add('v3',v3);
vlist.add('v4',v4);

res  = BF(vlist);
print(res);
```

References

[1] William K. Lam, *Hardware design verification*, 1 ed., Prentice Hall Modern Semiconductor Design Series, Prentice Hall, 2005.

[2] Tracy Larrabee, *Test pattern generation using boolean satisfiability*, IEEE Trans. Computer-Aided Design **II** (1992), no. No. 1, 4 – 15.

[3] Giovanni De Micheli, *Synthesis and optimization of digital circuits*, McGraw-Hill, 1994.

[4] Kurt Keutzer Srinivas Devadas, Abhijit Ghosh, *Logic synthesis*, McGraw-Hill Series on Computer Engineering, McGraw-Hill, 1994.