Miguel Alcón Doganoc
Algorithms for VLSI
December 5, 2019

Exercises on logic synthesis

1 Two-level minimization

Let us consider the following incompletely specified function, where f represents the on-set and d the don't care set:

$$f = \bar{a}\bar{b}c + bc\bar{d} + \bar{a}b\bar{c}d + a\bar{b}c$$
$$d = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{c}\bar{d} + ab\bar{c}d$$

Compute all prime implicants and identify the essential primes.

In order to do so, I applied the Quine–McCluskey algorithm. First, I created the truth table of functions f, d and its "union" u, as you can see in table 1. When there is a 1 in one of the rows in d, I do not care about the value in the same row for f, so there is a '-' in u. Otherwise, the value of f in the row is the same in u.

	a	b	c	d	f	d	u
$m0$	0	0	0	0	0	1	-
$m1$	0	0	0	1	0	1	-
$m2$	0	0	1	0	1	0	1
$m3$	0	0	1	1	1	0	1
$m4$	0	1	0	0	0	1	-
$m5$	0	1	0	1	1	0	1
$m6$	0	1	1	0	1	0	1
$m7$	0	1	1	1	0	0	0
$m8$	1	0	0	0	0	0	0
$m9$	1	0	0	1	0	0	0
$m10$	1	0	1	0	1	0	1
$m11$	1	0	1	1	1	0	1
$m12$	1	1	0	0	0	0	0
$m13$	1	1	0	1	0	1	-
$m14$	1	1	1	0	1	0	1
$m15$	1	1	1	1	0	0	0

Table 1: Truth table of f, d and u.

After that, I found all prime implicants. First, I selected all rows where the value of u is either '-' or 1 (minterms). Then, I combined the minterms. If two minterms differ only by one digit, the digit is replaced by '-'. If an implicant cannot combine with anyone, is marked with '*'. All marked implicants are prime implicants. The result of this is shown in table 2.

Finally, essential primes were found. A prime implicant is essential if it is the only one that covers some minterm. In this case, $m11$ is covered only by $m(2, 3, 10, 11)$ and $m14$ by $m(2, 6, 10, 14)$. All prime implicants are shown in table 3.

Number of 1s	Minterm	Binary	Implicants of size 2	Implicants of size 4
			$m(0,1)$ 000-	$m(0,1,2,3)$ 00–*
0	$m0$	0000	$m(0,2)$ 00-0	$m(0,1,4,5)$ 0-0-*
			$m(0,4)$ 0-00	$m(0,2,4,6)$ 0–0*
			$m(1,3)$ 00-1	
			$m(1,5)$ 0-01	
	$m1$	0001	$m(2,3)$ 001-	$m(2,3,10,11)$ -01- *
1	$m2$	0010	$m(2,6)$ 0-10	$m(2,6,10,14)$ –10 *
	$m3$	0011	$m(2,10)$ -010	
			$m(4,5)$ 010-	
			$m(4,6)$ 01-0	
	$m4$	0100		
2	$m5$	0101	$m(3,11)$ -011	
	$m6$	0110	$m(5,13)$ -101*	
	$m10$	1010	$m(6,14)$ -110	
	$m11$	1011	$m(10,11)$ 101-	
3	$m13$	1101	$m(10,14)$ 1-10	
	$m14$	1110		

Table 2: Followed process to find all prime implicants. Primes implicants are marked with '*'.

Primes	2	3	5	6	10	11	14	Binary
$m(5,13)$			×					-101
$m(0,1,2,3)$	×	×						00–
$m(0,1,4,5)$			×					0-0-
$m(0,2,4,6)$	×			×				0–0
$m(2,3,10,11)$*	×	×			×	×		-01-
$m(2,6,10,14)$*	×			×	×		×	–10

Table 3: All prime implicants and its binary representation. Essential primes are marked with '*'.

It is easy to see that the essential primes cover all minterms except $m5$, which is covered by both $m(5,13)$ and $m(0,1,4,5)$. Hence, the SOP is

$$m(2,3,10,11)m(2,6,10,14)m(5,13) + m(2,3,10,11)m(2,6,10,14)m(0,1,4,5) = 1$$

Because of that, there are two possible solutions:

$$u = \bar{b}c + c\bar{d} + b\bar{c}d$$
$$u = \bar{b}c + c\bar{d} + \bar{a}\bar{c}$$

Find a cover that minimizes the number of literals of the SOP.

Comparing both solutions, $u = \bar{b}c + c\bar{d} + \bar{a}\bar{c}$ is the one that has less literals, so this represents the cover that minimizes the number of literals of the SOP $(m(2,3,10,11)m(2,6,10,14)m(0,1,4,5))$.

Find a cover that minimizes the number of variables in the support of the SOP.

Comparing both solutions, $u = \bar{b}c + c\bar{d} + b\bar{c}d$ is the one that uses less variables, so this represents the cover that minimizes the number of variables in the support of the SOP $(m(2,3,10,11)m(2,6,10,14)m(5,13))$.

2 Multi-level minimization

Let us consider the following Boolean network with two nodes:

$$x = abd + bc + be + abg + ae$$
$$y = ade + cd + ce + ef + h$$

Calculate the kernels of both functions.

In tables 4 and 5 is described the process I followed to calculate the kernels of both functions, which is the recursive kernel computation. Divisors marked with '*' indicates that I stopped the recursion there (the result is a cube or a sum of cubes of 1 element). The resultant kernels of x and y are showed in tables 6 and 7, respectively.

Extract the best multi-cube common divisor.

As the Brayton and McMullen theorem says, two expressions x and y have a common multiple-cube divisor if and only if there exist kernels $k_x \in K(x)$ and $k_y \in K(y)$ such that the divisor is the sum of two (or more) cubes un $k_x \cap k_y$.

Trying to intersect each kernel of x with each kernel of y, which is the minimum possible intersection, I obtained only one intersection formed by a sum of two (or more) cubes, as you can see in table 8. The only multi-cube common divisor is $ad + c$, which is the result of $\{(ad + c + e + ag)\} \cap \{(ad + c + f)\}$. Since it is the only one, it is the best.

Create a new node with the divisor and substitute in the previous expressions.

The new node is $w = ad + c$, and substituting it in the initial expressions I obtained:

$$x = wb + be + abg + ae$$
$$y = we + wd + ef + h$$

3 Technology mapping

Consider a gate library that only has three gates: 2-input NAND, 2-input NOR and inverter. The area cost of the NAND and NOR gates is 2, whereas the inverter has cost 1. We want to do technology mapping using only 2-input NAND gates and inverters as base functions in the target graph. Answer the following questions:

Divisor	Result	Cube-free
1	abd + bc + be + abg + ae	×
a	bd + bg + e	×
ab*	d + g	×
ad*	b	
ag*	b	
ae*	1	
b	ad + c + e + ag	×
ba*	d + g	×
bd*	a	
bc*	1	
be*	1	
bg*	a	
c*	b	
d*	ab	
e*	b + a	×
g*	ab	

Table 4: Followed process to calculate kernels of x.

Divisor	Result	Cube-free
1	ade + cd + ce + ef + h	×
a	de	
ad*	e	
ae*	d	
c*	d + e	×
d*	a + d + e	×
e	ad + c + f	×
ea*	d	
ed*	a	
ec*	1	
ef*	1	
f*	e	
h*	1	

Table 5: Followed process to calculate kernels of y.

Co-kernel	Kernel
a	bd + bg + e
ab	d + g
b	ad + c + e + ag
e	b + a
1	abd + bc + be + abg + ae

Table 6: Kernels of x.

Co-kernel	Kernel
c	d + e
d	a + d + e
e	ad + c + f
1	ade + cd + ce + ef + h

Table 7: Kernels of y.

Kernel of x	Kernel of y	Intersection
bd + bg + e	d + e	e
	a + d + e	e
	ad + c + f	$\emptyset$
	ade + cd + ce + ef + h	$\emptyset$
d + g	d + e	d
	a + d + e	d
	ad + c + f	$\emptyset$
	ade + cd + ce + ef + h	$\emptyset$
ad + c + e + ag	d + e	e
	a + d + e	e
	ad + c + f	**ad + c**
	ade + cd + ce + ef + h	$\emptyset$
b + a	d + e	$\emptyset$
	a + d + e	a
	ad + c + f	$\emptyset$
	ade + cd + ce + ef + h	$\emptyset$
abd + bc + be + abg + ae	d + e	$\emptyset$
	a + d + e	$\emptyset$
	ad + c + f	$\emptyset$
	ade + cd + ce + ef + h	$\emptyset$

Table 8: Intersection between all kernels of x and y.

Draw the library patterns for the gates in the library using the base functions.

Base functions are:

$$f_{\text{NAND}} = \overline{a \wedge b}$$
$$f_{\text{INV}} = \bar{a}$$

a	b	$\overline{a \wedge b}$ (NAND)	$\bar{a}$ (inverter)	$\overline{a \vee b}$ (NOR)
0	0	1	1	1
0	1	1	1	0
1	0	1	0	0
1	1	0	0	0

Table 9: Truth table of the three gates.

Using the functions and the truth tables of the gates (see table 9), I drew the library patterns showed in figure 1.

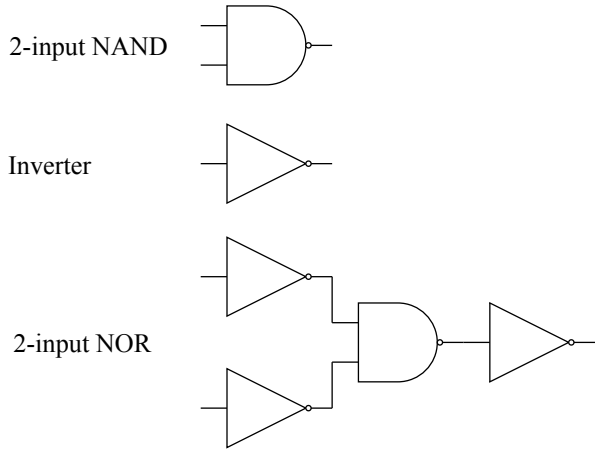

Figure 1: Library patterns for the gates in the library.

Consider the following function:

$$f = a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4$$

Draw the most balanced target graph in terms of the base functions and find the most efficient mapping.

I started drawing the tree of the boolean operations of f (see figure 2). Using the library patterns, I created simple versions of AND ($\wedge$) and OR ($\vee$) gates. AND is formed by NAND and an inverter, and OR by NOR and an inverter. With the tree and these new gates, I drew the initial circuit, which is shown in figure 3. Blue indicates an area cost of 2 and green of 1. Hence, the initial circuit has an area cost of $2 \times 7 + 7 = 21$.

It is easy to see that the part of the initial circuit within the red rectangle (red zone) can be changed by two NAND gates, removing all two consecutive inverters in the zone. The area cost of the red zone goes from 10 to 4. The resultant circuit is shown in figure 4, and have an area cost of $7 \times 2 + 1 = 15$.

Now, the only optimization you can do is to remove the two consecutive inverters at the end of the circuit, with leads to the one in figure 5. Unless this is the circuit with less gates, it has an area cost of $7 \times 2 + 2 = 16$, because removing the inverters it loses the NOR gate, increasing the cost by 1. So, the most efficient mapping is the one in figure 4.

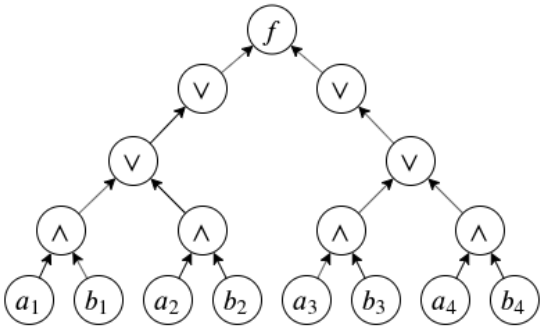

Figure 2: Tree of f.

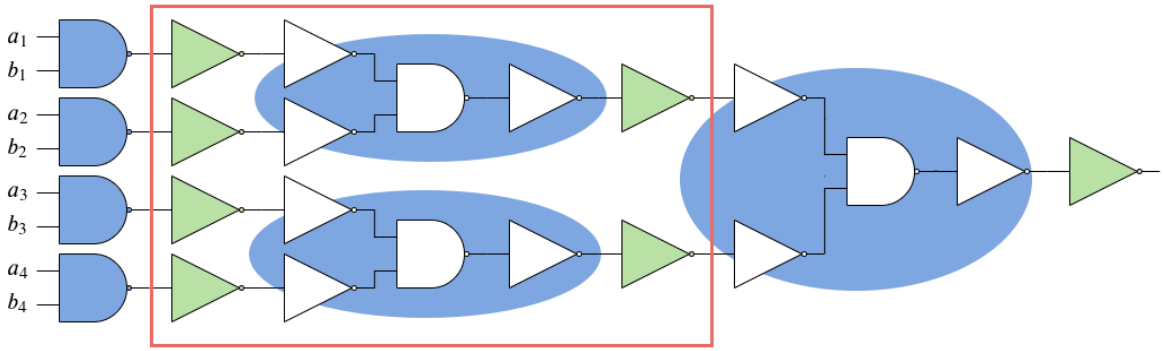

Figure 3: Initial circuit.

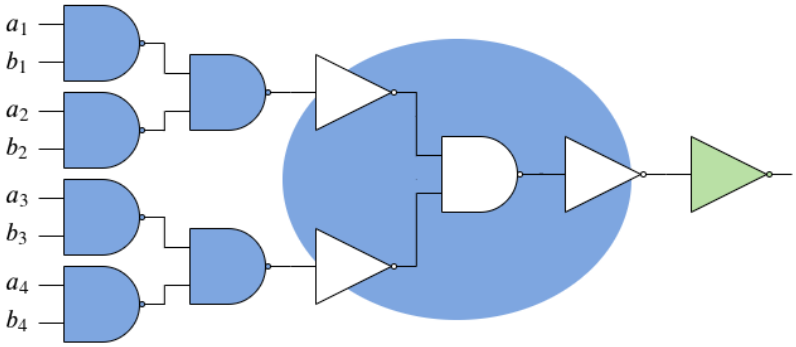

Figure 4: Most efficient mapping.

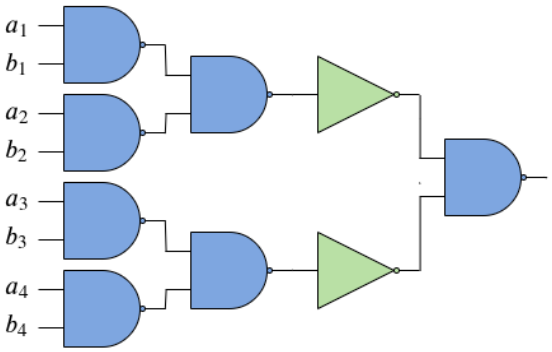

Figure 5: Circuit with less gates.

Let us assume that every gate has delay = 1 and that the arrival time of all signals is $t = 0$, except for b_4 whose arrival time is $t = 10$. Draw the target graph that minimizes delay and find the most efficient mapping.

I started drawing the tree of the boolean operations of f (see figure 6). I build it taking into account that the arrival time of b_4 is 10. Then, I applied the same method that in previous question. You can see the resultant circuit in figure 7. After that, I removed the two consecutive inverters within the red zone. Only one of them is maintained because it will not affect to the final delay (as I show later) and with this one NOR gate is maintained. With this, I obtained the most efficient mapping (area cost of 15, as in the previous question) with the minimum possible delay, which is 12.

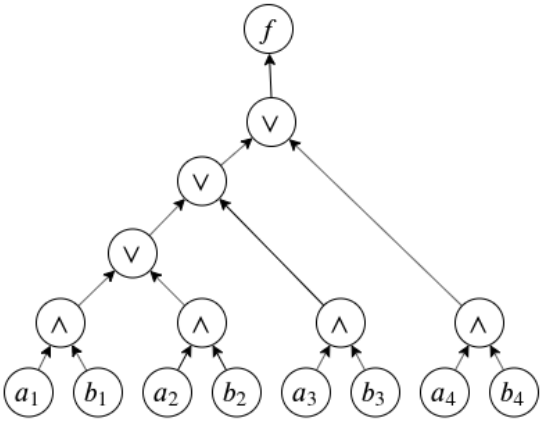

Figure 6: Tree of f with the critical delay path at b_4.

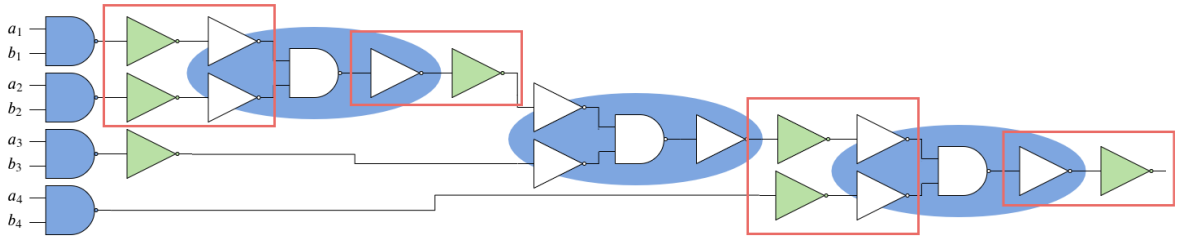

Figure 7: Initial circuit.

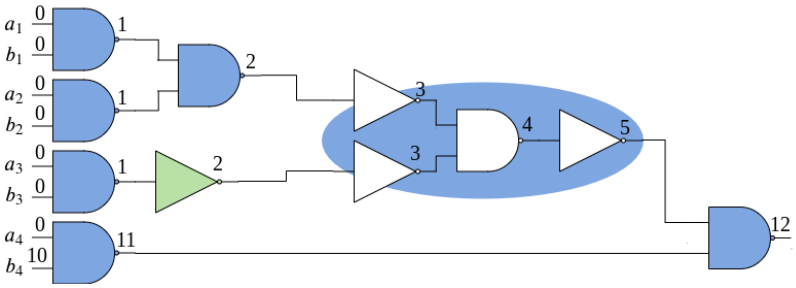

Figure 8: Most efficient mapping with less delay.

4 Binary Decision Diagrams

Let us consider two Boolean vectors, $(a_{n-1}, ..., a_0)$ and $(b_{n-1}, ..., b_0)$, representing the binary encoding of two natural numbers a and b. Let us consider the BDD of a Boolean function that is true when $a < b$

and false when $a \geq b$.

Justify the following answers and give an asymptotic complexity (using $O()$) of the BDD size for each case. Draw the BDD of the function considering the best variable order.

What is the variable order that minimizes the BDD size?

If the BDD starts from bit $n-1$, for each bit i the BDD only has to check whether $a_i < b_i$ to return *true*, $a_i > b_i$ to return *false*, or $a_i = b_i$ to continue. This leads to a BDD like the one in figure 9 (first BDD). It has n nodes for a and $2n$ nodes for b.Then, the size of this BDD is $3n$ ($O(n)$). The BDD of the figure 9 starts with a_{n-1} but can be done starting with b_{n-1} in the same way.

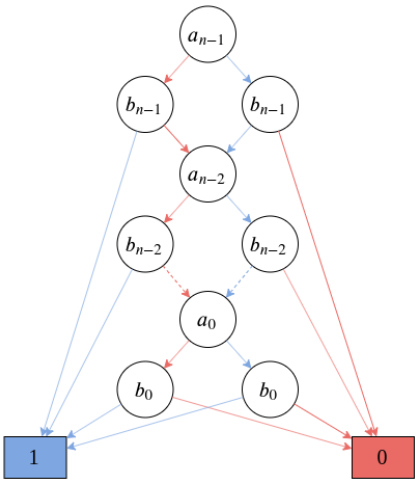

Figure 9: BDD of the function with the order that minimizes the size.

What is the variable order the maximizes the BDD size?

If the BDD starts from bit 0 with variable a, and considering a_j (with $0 < j < n$) as a saving state (saving whether $(a_0, ..., a_j) > (b_0, ..., b_j)$ or $(a_0, ..., a_j) \geq (b_0, ..., b_j)$), the BDD in figure 10 (second BDD) is obtained. It can be also done starting with b and using b_j as a saving state. The size of this BDD is $2 + 4(n-1)$ ($O(n)$).

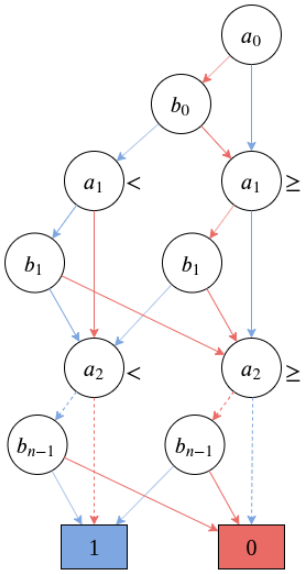

Figure 10: BDD of the function with the order that maximizes the size.

Although the asymptotic complexity of the first BDD is the same than the other, their sizes are not ($3n < 2 + 4(n-1)$ for $n > 1$). Furthermore, the first BDD can stop when checking each bit, only in the worst case (when $a = b$) it goes through all the BDD. The second BDD always does the latter.