**Informal Algorithm**

What we are going to do is circle groups of “1s” that are rectangles[[1]](#footnote-1) where each side is a power of 2 in length. The first K-map on this page had a 2x1 and a 1x2 rectangle, the second a 1x4 and the third a 1x2. We never circle a rectangle that is part of a larger legal rectangle. We only circle enough rectangles so that every 1 is circled.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |

Notice that in the K-map above there are three rectangles we could circle, but two of them cover all the “1s”.

Let’s practice a bit.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**An interesting example**

Circle all rectangles:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
*Answer 1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Answer 2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 **Terminology (p308-310)**

Notice that we are finding sum-of-products solutions.

* Recall that a ***minterm*** is a product term that includes all of the functions variables exactly once.
* The ***on-set*** of a function is the set of minterms that define when the function should evaluate to 1 (the minterms that have a 1 in the truth table.)
	+ The ***off-set*** is the set of minterms that evaluate to zero.
* An ***implicant*** of a function is a product term that evaluates to 1 only in places that function evaluates to 1. (The on-set of an implicant of a function is a subset of the on-set of the function.)

	+ Graphically, in a K-map an implicant is: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* An implicant ***covers*** those minterms that appears in its on-set.

	+ What is the on-set of the function F(a,b)=a? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
	+ What minterms does that function cover? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* Removing a variable from a term is known as ***expanding*** the term. This is the same as expanding the size of a circle on a K-map.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

* ***Prime implicant***:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* ***Essential one[[2]](#footnote-2)***: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* ***Essential prime implicant***: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab/c |  00 |  01 |  11 |  10 |
| 0 |  |  |  |  |
| 1 |  |  |  |  |





## More Formal Algorithm

* Identify all prime implicants
* Identify all essential ones.
* Circle all essential prime implicants
* Cover the remaining minterms using a minimal number of remaining prime implicants.

Notice that there may be more than one solution. Also notice that the last step is a bit vague ☺

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab/c** |  00 |  01 |  11 |  10 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

## 4-variable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab/cd** |  00 |  01 |  11 |  10 |
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 |

And some practice with these:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab/cd** |  00 |  01 |  11 |  10 |
| 00 | 1 | 1 | 1 | 1 |
| 01 | 0 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 0 |
| 10 | 1 | 1 | 0 | 0 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab/cd** |  00 |  01 |  11 |  10 |
| 00 | 1 | 1 | 0 | 1 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 1 |
| 10 | 1 | 0 | 0 | 1 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab/cd** |  00 |  01 |  11 |  10 |
| 00 | 0 | 1 | 0 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 1 | 0 | 0 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## What’s left?

* Don’t cares
* 5+ variable
* Product-of-sums
* Programmable techniques
* Lots of practice.
* More context. Remember this is only for 2-level logic…

## Don’t cares:

Note: I’m leaving the zeros blank to make things more readable!



* Use d cells to make prime implicants as large as possible.
* No PI should include only d’s
* Only 1-cells should be considered when finding the minimal cover set.

## 5-variable



## Product of Sums:

Let F be: Then F’ is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab/cd** |  00 |  01 |  11 |  10 |
| 00 |  | 1 | 1 | 1 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 |  |  |
| 10 | 1 | 1 |  |  |



F🡺F’

Find minimal SoP for F’.

Use deMorgans.

## Programmable techniques

* Later in the semester, time allowing.

## More context

Basically, just remember that this doesn’t find the “minimal” solution. If finds the minimal sum-of-products. It’s not even clear how we would measure “minimal” over all. Least delay? Least number of gates? Least “gate inputs”? Recall we did a “least delay” solution for GA1. This would have helped find a nice starting point, but wouldn’t have solved the problem.

But this technique does let us find the minimal two-level solution (SoP or PoS). Which is pretty cool.

### Questions:

1. Why isn’t this a great technique for a computer? Why is it good for people?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab/cd** |  00 |  01 |  11 |  10 |
| 00 |  |  | 1 |  |
| 01 | 1 | 1 | 1 |  |
| 11 |  | 1 | 1 | 1 |
| 10 |  | 1 |  |  |

1. What might be easier for a computer?
2. Can you define all the terms we’ve seen?
	* On-set, Off-set?
	* Implicant, prime implicant?
	* Essential one, essential prime implicant?
	* Cover?
3. Why is the instance to the right tricky?
1. Recall a square is special case of a rectangle. [↑](#footnote-ref-1)
2. This term isn’t used by our text, they skip from prime implicant directly to essential prime implicant. [↑](#footnote-ref-2)