

Digital Logic Design

ENEE 244-020x

Lecture 10

Announcements

- Midterms back at the end of class
 - Median: 83.5
 - Solutions posted on Canvas/ELMS

Agenda

- Last time
 - Minimal expressions for incomplete Boolean functions (3.5)
 - 5 and 6 variable K-Maps
 - Gate Properties (2.8)
- This time
 - Quine-McCluskey method
 - Verilog Intro (3.9)

Quine-McCluskey method

Tabular Representations

		yz			
		00	01	11	10
wx	00	0	0	0	1
	01	1	1	1	1
	11	0	1	1	1
	10	0	1	0	0

$\bar{w}x$
01 – –

$\bar{w}y\bar{z}$
0 – 10

$w\bar{y}z$
1 – 01

xy
– 11 –

$$f(w, x, y, z) = xy + \bar{w}y\bar{z} + w\bar{y}z + \bar{w}x$$

Prime Implicants

- $f(w, x, y, z) = xy + \overline{w}y\overline{z} + w\overline{y}z + \overline{w}x$
- Each product term is an implicant.
- Prime implicant: A product term that cannot have any of its literals removed and still imply the function.

Prime Implicants

		yz			
		00	01	11	10
x	0	0	0	0	1
	1	1	1	1	1

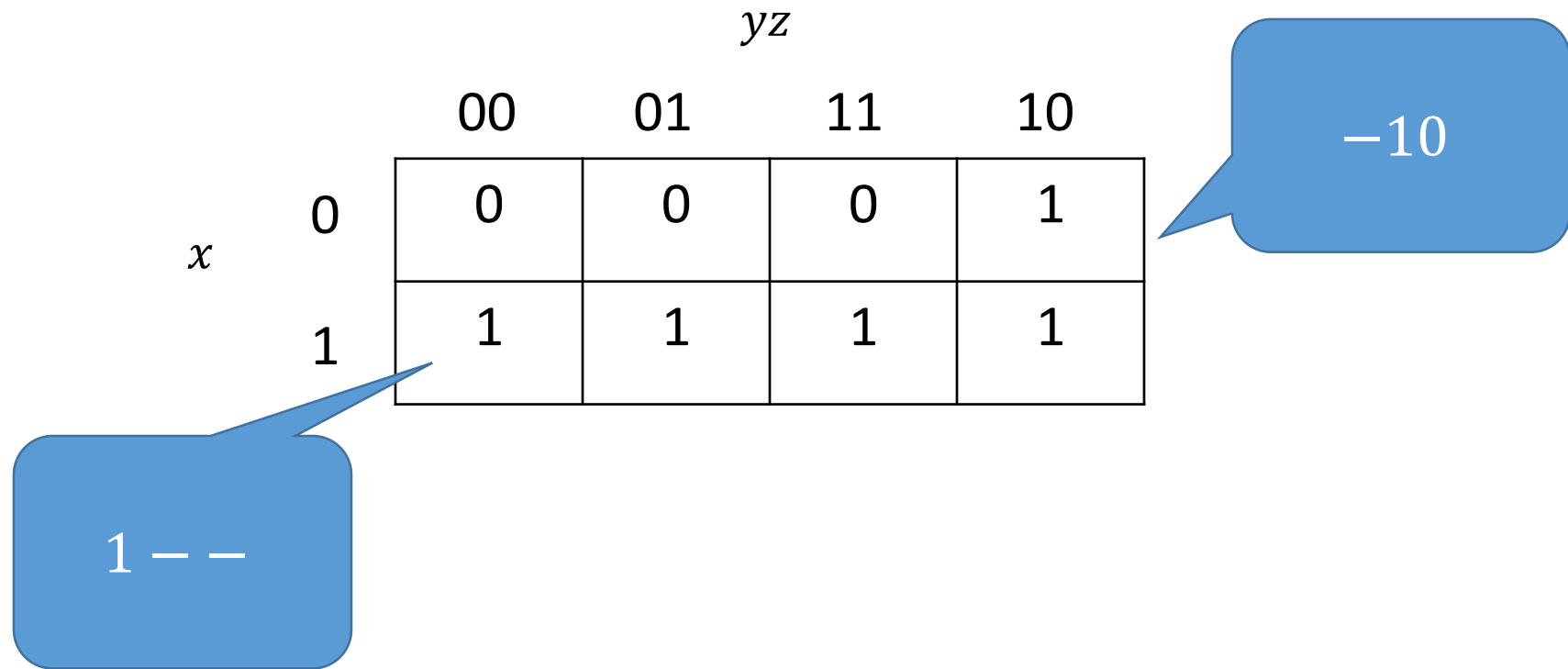


Diagram illustrating Prime Implicants for a 3-variable Karnaugh map (variables x, y, z).

The Karnaugh map is shown as a 2x4 grid. The rows are labeled x (0 and 1) and the columns are labeled yz (00, 01, 11, 10). The values in the cells are:

x	yz	00	01	11	10
0		0	0	0	1
1		1	1	1	1

Two prime implicants are highlighted with blue callouts:

- The prime implicant $1--$ (where $x=1$) is indicated by a blue callout bubble.
- The prime implicant -10 (where $yz=10$) is indicated by a blue callout bubble.

Prime Implicants

Minterm	X	Y	Z	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

x

		<i>yz</i>			
		00	01	11	10
0		0	0	0	1
1		1	1	1	1

yz

x

10

1 --

$$f(x, y, z) = y\bar{z} + x$$

Finding Prime Implicants

Step 1					Step 2				Step 3			
2	0	1	0	✓	(2,6)	—	1	0	(4,5,6,7)	1	—	—
4	1	0	0	✓	(4,5)	1	0	—	(4,6,5,7)	1	—	—
5	1	0	1	✓	(4,6)	1	—	0				
6	1	1	0	✓	(5,7)	1	—	1				
7	1	1	1	✓	(6,7)	1	1	—				

All unchecked entries are Prime Implicants

— 1 0 $y\bar{z}$

1 — — x

Prime Implicants

Minterm	X	Y	Z	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

x

		<i>yz</i>			
		00	01	11	10
0		0	0	0	1
1		1	1	1	1

—10

1 — —

$$f(x, y, z) = y\bar{z} + x$$

Essential Prime Implicants

		yz			
		00	01	11	10
wx	00	1	1	1	1
	01	0	1	1	0
	11	0	0	1	1
	10	1	0	0	1

Find the Essential
Prime Implicants
using Quine-
McClusky

Essential Prime Implicants

		yz			
		00	01	11	10
wx	00	1	1	1	1
	01	0	1	1	0
	11	0	0	1	1
	10	1	0	0	1

minterms				
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
8	1	0	0	0
3	0	0	1	1
5	0	1	0	1
10	1	0	1	0
7	0	1	1	1
14	1	1	1	0
15	1	1	1	1

Finding Prime Implicants

0	0	0	0	0	✓	(0,1)	0	0	0	—	✓	(0,1,2,3)	0	0	—	—
1	0	0	0	1	✓	(0,2)	0	0	—	0	✓	(0,2,1,3)	0	0	—	—
2	0	0	1	0	✓	(0,8)	—	0	0	0	✓	(0,2,8,10)	—	0	—	0
8	1	0	0	0	✓	(1,3)	0	0	—	1	✓	(0,8,2,10)	—	0	—	0
3	0	0	1	1	✓	(1,5)	0	—	0	1	✓	(1,3,5,7)	0	—	—	1
5	0	1	0	1	✓	(2,3)	0	0	1	—	✓	(1,5,3,7)	0	—	—	1
10	1	0	1	0	✓	(2,10)	—	0	1	0	✓					
7	0	1	1	1	✓	(8,10)	1	0	—	0	✓					
14	1	1	1	0	✓	(3,7)	0	—	1	1	✓					
15	1	1	1	1	✓	(5,7)	0	1	—	1	✓					
						(10,14)	1	—	1	0						
						(7,15)	—	1	1	1						
						(14,15)	1	1	1	—						

6 Prime Implicants

1 — 10 00 — —
 — 111 — 1 — 1
 111 — 0 — — 1

Find Essential Prime Implicants

Prime Implicant Table

	Prime Implicant	Covered Minterms	Minterms									
			0	1	2	3	5	7	8	10	14	15
	1-10	10,14								X	X	
	-111	7,15						X				X
✓	111-	14,15									X	X
	00- -	0,1,2,3	X	X	X	X						
✓	-0-0	0,2,8,10	X		X				X	X		
✓	0- -1	1,3,5,7		X		X	X	X				














3 Prime Implicants

		yz			
		00	01	11	10
wx	00	1	1	1	1
	01	0	1	1	0
	11	0	0	1	1
	10	1	0	0	1

0- -1

111-

-0-0

3 Prime Implicants

0— —1

		<i>yz</i>			
		00	01	11	10
<i>wx</i>	00	1	1	1	1
	01	0	1	1	0
	11	0	0	1	1
	10	1	0	0	1

111—

—0—0

$$f(w, x, y, z) = \bar{w}z + \bar{x}\bar{z} + wxy$$

Verilog Intro

Simulator: <https://ecrionix.org/tools/verilog-simulator/>

Verilog Example 3.1

[Figure 3.35](#) is a logic diagram for a simple circuit in which the output of an OR gate is one of two inputs to an AND gate. The Boolean equation for the output of the circuit can be written directly from the diagram:

$E = (A + B)C$. We'll use its Verilog description to introduce key details of the language.

module...endmodule

--basic building block
--defines a circuit's
inputs, outputs, and
internal logic

```
module or_and (  
    output    E,  
    input     A, B, C  
);  
    wire D;  
    assign D = A || B;    // | is logical "OR" operator  
    assign E = C && D;    // & is the logical "AND" operator  
    // This is a single-line comment  
    /* The text here and below  
       form a multi-line comment  
    */  
endmodule
```

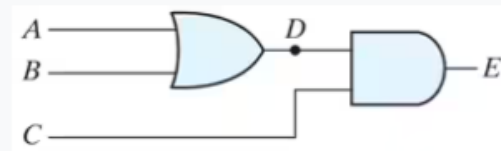
&&--logical AND
||--logical OR
!—logical NOT

assign keyword

defines a *continuous assignment*.
Continuously updates a
signal whenever the
inputs change.

FIGURE 3.35

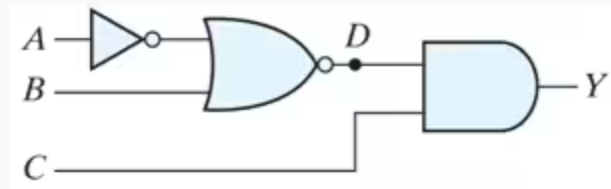
A logic diagram (schematic) for the Boolean equations $D = A + B$ and $E = CD$



Practice Exercise 3.12 – Verilog

Write a continuous assignment statement that describes Y in the logic diagram in [Fig. PE3.12](#), where A , B , C , and D are Boolean variables.

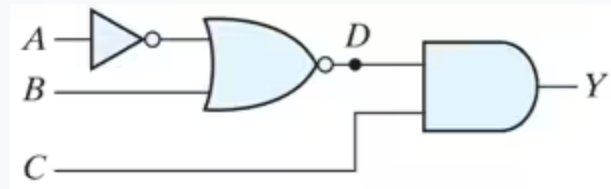
FIGURE PE3.12



Practice Exercise 3.12 – Verilog

Write a continuous assignment statement that describes Y in the logic diagram in [Fig. PE3.12](#), where A , B , C , and D are Boolean variables.

FIGURE PE3.12



Answer: assign $Y = (!((!A) || B)) \& \& C$

Verilog Example 3.2

This example develops a Verilog model of a circuit having inputs A , B , C , D and outputs E , F , with functionality specified by the following Boolean expressions:

$$E = A + BC + B'D$$

$$F = B'C + BC'D'$$

Verilog Example 3.2

This example develops a Verilog model of a circuit having inputs A , B , C , D and outputs E , F , with functionality specified by the following Boolean expressions:

$$E = A + BC + B'D$$

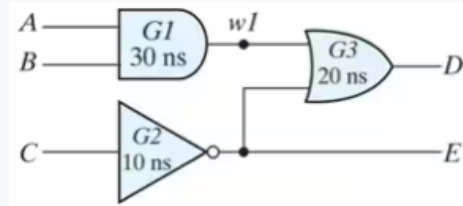
$$F = B'C + BC'D'$$

Answer:

```
module Circuit_Boolean_CA (E, F, A, B, C, D);  
  output      E, F;  
  input       A, B, C, D;  
  assign E = A || (B && C) || ((!B) && D);  
  assign F = ((!B) && C) || (B && (!C) && (!D));  
endmodule
```

FIGURE 3.37

Schematic for *and_or_prop_delay*



The gates within a module may be listed in any order. They operate concurrently in simulation. A signal can affect simultaneously all of the gates to which it is connected as an input. Each affected gate independently determines and schedules events¹¹ for its output.

Verilog primitives have built-in logic determining their behavior. The optional user-specified propagation delays (e.g., 30 ns) determine the time interval between a change to the input signal of a gate and the effect apparent at the output of the gate, that is, the model reflects the fact that the input/output time response of a physical logic gate is not instantaneous.¹²

```
'timescale 1 ns / 1 ps // time units / resolution
module and_or_prop_delay (
    input A, B, C;
    output D, E;
);
    wire w1;
    and G1 #30 (w1, A, B); // Prop delay: 30 ns
    not G2 #10 (E, C); // Prop delay: 10 ns
    or G3 #20 (D, w1, E); // Prop delay: 20 ns
endmodule
```

Wire keyword

Primitive gates have keyword names: **and**, **nand**, **or**, **nor**, **xor**, **xnor**, **but**, **not**

Gate instances and corresponding propagation delays

A test bench for *and_or_prop_delay* is given below:

```
// Test bench for and_or_prop_delay
module t_and_or_prop_delay;
  wire    D, E;
  reg     A, B, C;
  and_or_prop_delay M_UUT (A, B, C, D, E); // Instance name (M_UUT) is required

  initial begin
    A = 1'b0; B = 1'b0; C = 1'b0;
    #100 A = 1'b1; B = 1'b1; C = 1'b1;
  end

  initial #200 $finish;
endmodule
```

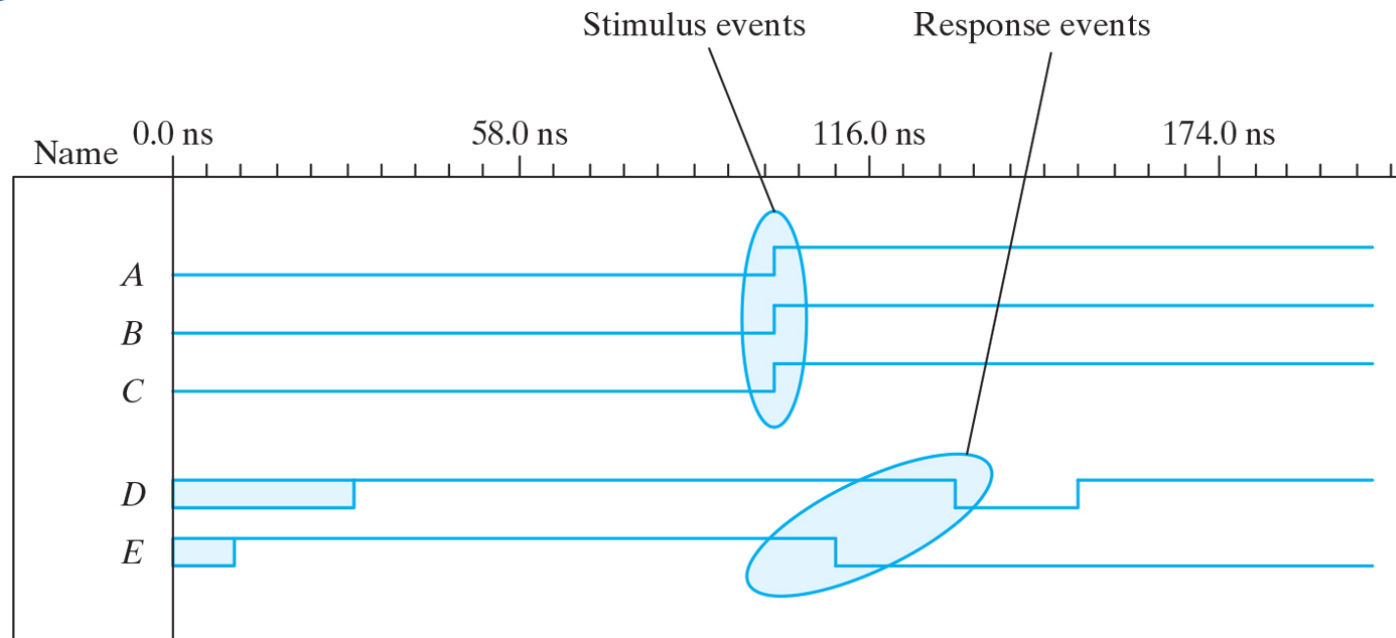
begin...end block
block statement, executed
sequentially

1'b0: 1 binary digit set to 0
1'b1: 1 binary digit set to 1

\$finish: system task to
terminate the simulation

Module Unit
Under Test
(M_UUT)

initial keyword
Statements begin executing when
simulation is initialized



Verilog Example 3.4 (User-Defined Primitive)

```
// Verilog model: User-defined Primitive
primitive UDP_02467 (D, A, B, C);
output D;
input  A, B, C;
//Truth table for D = f (A, B, C) =  $\Sigma(0, 2, 4, 6, 7)$ ;
table
//A B C : D // Column header comment
0 0 0 : 1;
0 0 1 : 0;
0 1 0 : 1;
0 1 1 : 0;
1 0 0 : 1;
1 0 1 : 0;
1 1 0 : 1;
1 1 1 : 1;
endtable
endprimitive
// Instantiate primitive
// Verilog model: Circuit instantiation of Circuit_UDP_02467
module Circuit_with_UDP_02467 (e, f, a, b, c, d);
output e, f;
input  a, b, c, d;
UDP_02467 (e, a, b, c);
and      (f, e, d); // Optional gate instance name omitted
endmodule
```

Note that the variables listed at the top of the table are part of a comment and are shown only for clarity. The system recognizes the variables by the order in which they are listed in the input declaration. A user-defined primitive can be instantiated in the construction of other modules (digital circuits), just as the system primitives are used. For example, the declaration

```
Circuit_with_UDP_02467(E,F,A,B,C,D);
```

will produce a circuit that implements the hardware shown in [Fig. 3.40](#).

FIGURE 3.40

Schematic for *Circuit with_UDP_02467*

