
Digital Design: A Systems Approach

Lecture 2: Combinational Logic Design

Readings

- L2: Chapters 6 & 7
- L3: Chapters 8 & 9

Review

- Lecture 1
 - The world is digital
 - Analog at the edges, hardware for demanding problems, 100x per decade
 - Digital signals
 - Encode discrete states in a continuous signal
 - Reject noise
 - Representations
 - Binary, set, continuous, compound
 - Boolean Algebra ($0, 1, \wedge, \vee$)
 - Axioms, properties, duality
 - Logic equations express binary functions
 - Combinational logic
 - Output is a function only of current input
 - Verilog
 - Defines hardware modules, assign, case

Today

- How to implement combinational logic by hand
 - Given a description of a logic function
 - Generate a gate-level circuit that realizes that function
- Everyone needs to do this once
 - To understand how its done
 - Demystifies what the synthesis tools do
 - Better understanding of what synthesis tools can and can't do
- In practice you will rarely have to do this by hand
- General practice is:
 - Design using Verilog
 - Simulate with test cases
 - Generate gates with synthesis program

English language description of a combinational logic function

$F(d,c,b,a)$ is true if input d,c,b,a is prime

Truth Table

$F(d,c,b,a)$ is true if input d,c,b,a is prime

No	dcba	q
0	0000	0
1	0001	1
2	0010	1
3	0011	1
4	0100	0
5	0101	1
6	0110	0
7	0111	1
8	1000	0
9	1001	0
10	1010	0
11	1011	1
12	1100	0
13	1101	1
14	1110	0
15	1111	0

Equation

$F(d,c,b,a)$ is true if input d,c,b,a is prime

$$f = \sum_{dcba} m(1,2,3,5,7,11,13)$$

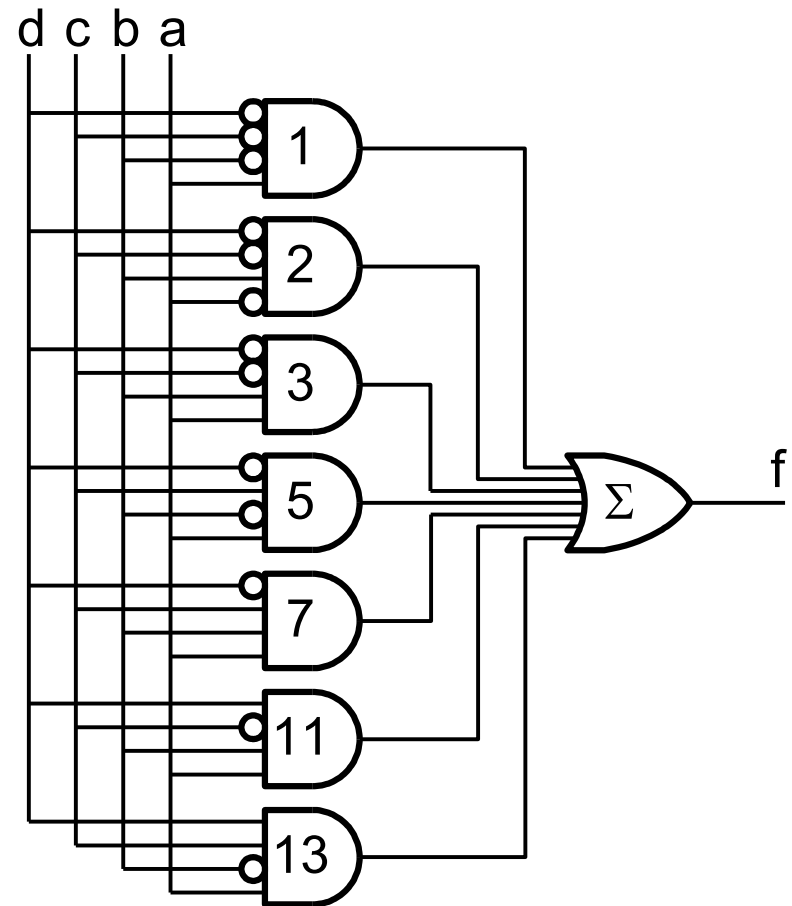
No	dcba	q
0	0000	0
1	0001	1
2	0010	1
3	0011	1
4	0100	0
5	0101	1
6	0110	0
7	0111	1
8	1000	0
9	1001	0
10	1010	0
11	1011	1
12	1100	0
13	1101	1
14	1110	0
15	1111	0

Schematic Logic Diagram

Equation:

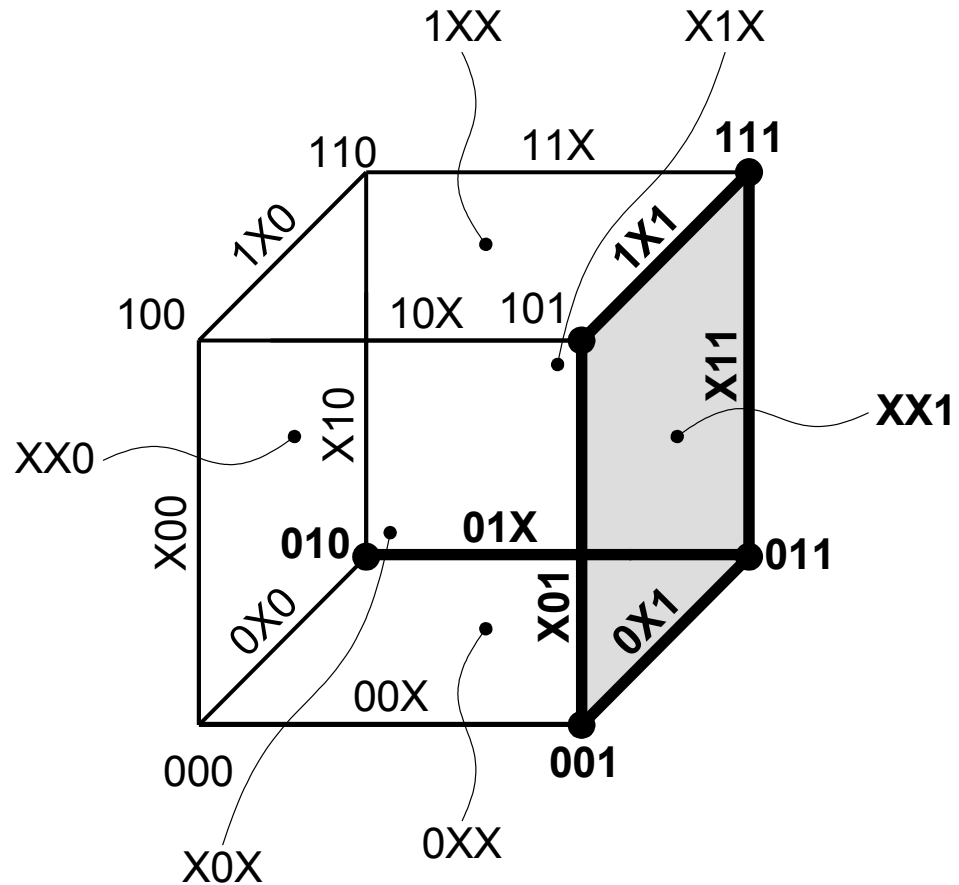
$$f = \sum_{dcba} m(1,2,3,5,7,11,13)$$

Schematic Logic Diagram:



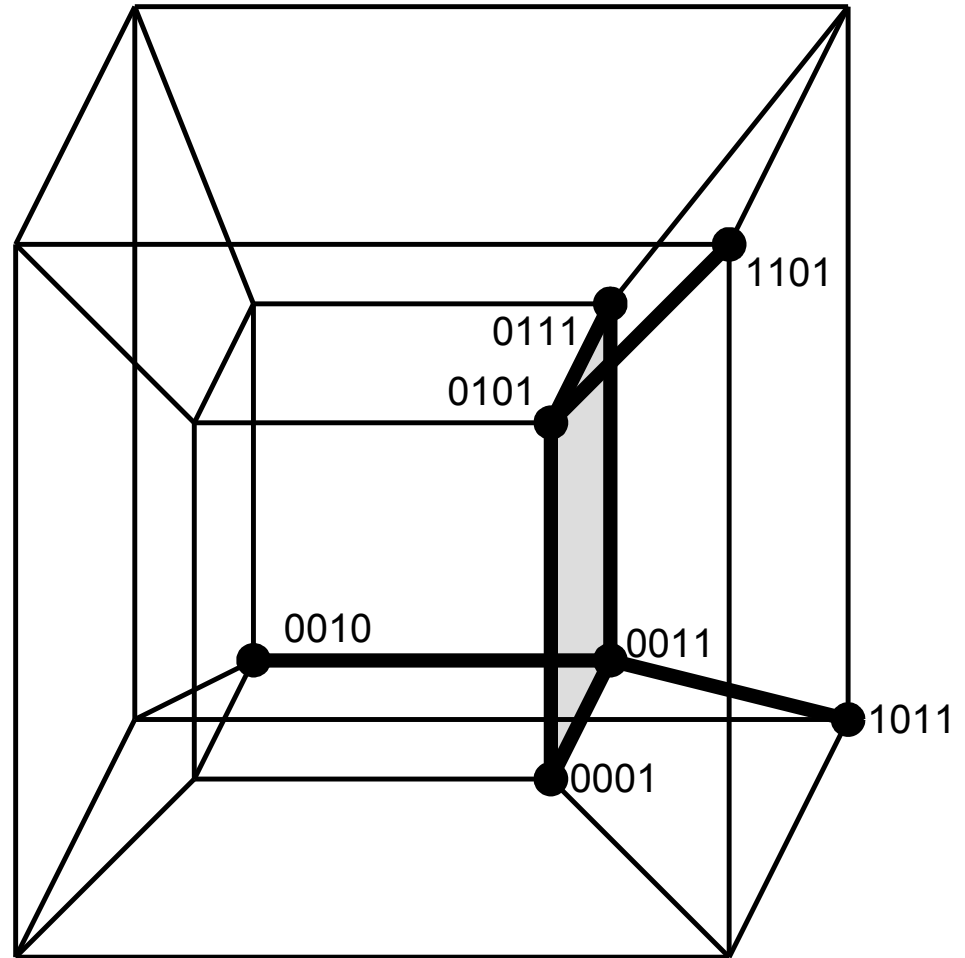
Cube representation (3-bit prime)

$$f = \sum_{cba} m(1,2,3,5,7)$$

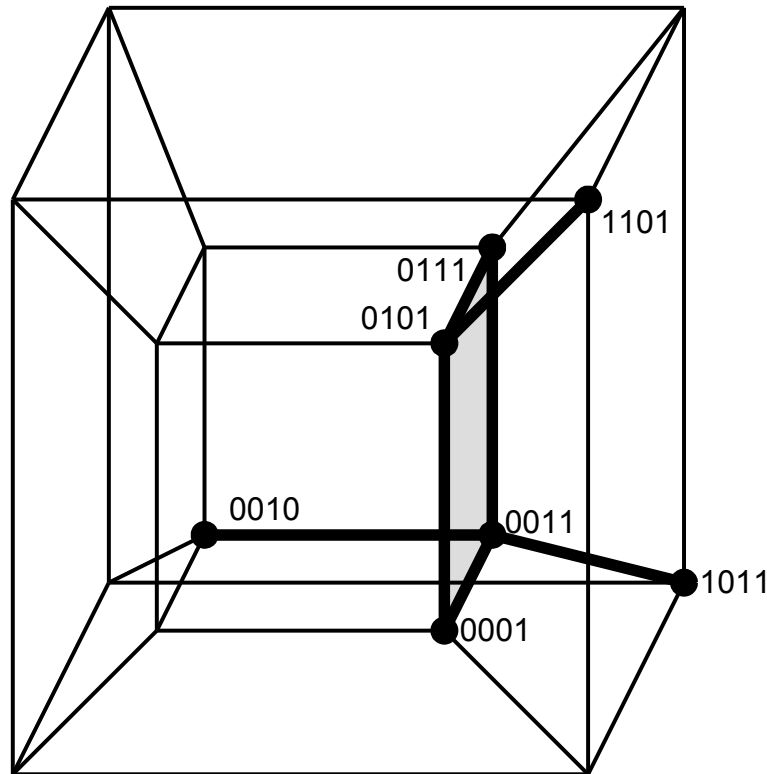


4-D Hypercube (4-bit prime)

$$f = \sum_{dcba} m(1,2,3,5,7,11,13)$$



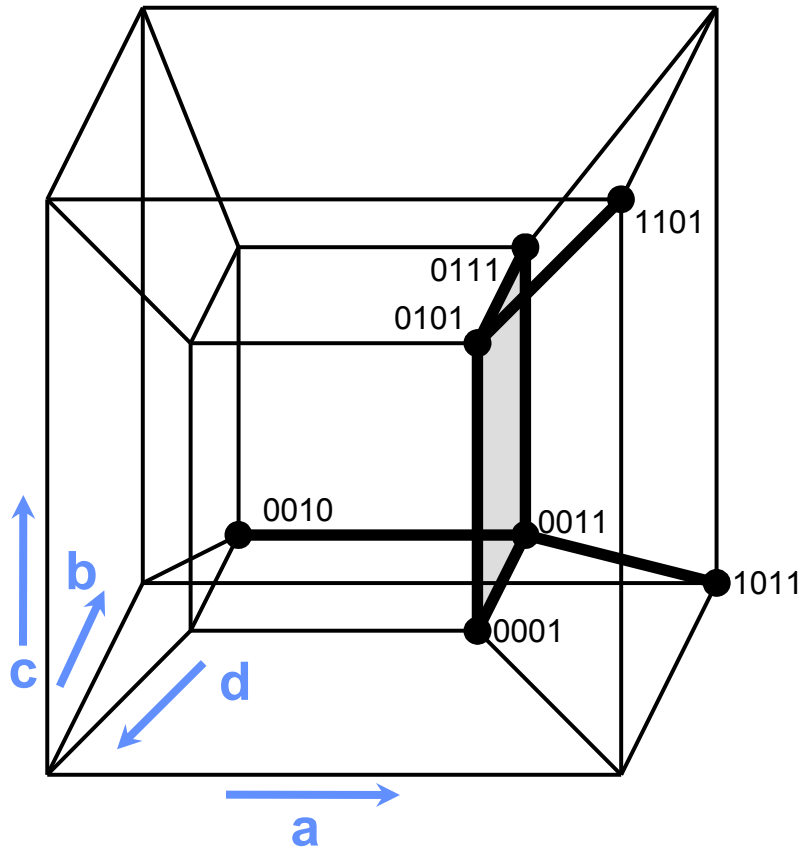
4-bit Prime Number Function (cont)



dcba
0xx1
001x
x101
x011

$$f = (a \wedge \bar{d}) \vee (b \wedge \bar{c} \wedge \bar{d}) \vee (a \wedge \bar{b} \wedge c) \vee (a \wedge b \wedge \bar{c})$$

Karnaugh Map of 4-bit Prime



		<u>a</u>			
		ba			
c	dc	00	01	11	10
	00	0 ₀	1 ₁	1 ₃	1 ₂
	01	0 ₄	1 ₅	1 ₇	0 ₆
	11	0 ₁₂	1 ₁₃	0 ₁₅	0 ₁₄
		<u>b</u>			
		10			
		0 ₈	0 ₉	1 ₁₁	0 ₁₀

Karnaugh Map of 4-bit Prime: Minterms

		a				
		ba	00	01	11	10
c	dc	00	0	1	3	2
	01	4	5	7	6	
	11	12	13	15	14	
	10	8	9	11	10	
		b				
		d				

Position of minterms on a 4-bit Karnaugh map

Karnaugh Map of 4-bit Prime: Implicants

ba

dc

a

c

d

b

00	01	11	10
0000	0001	0011	0010
0100	0101	0111	0110
1100	1101	1111	1110
1000	1001	1011	1010

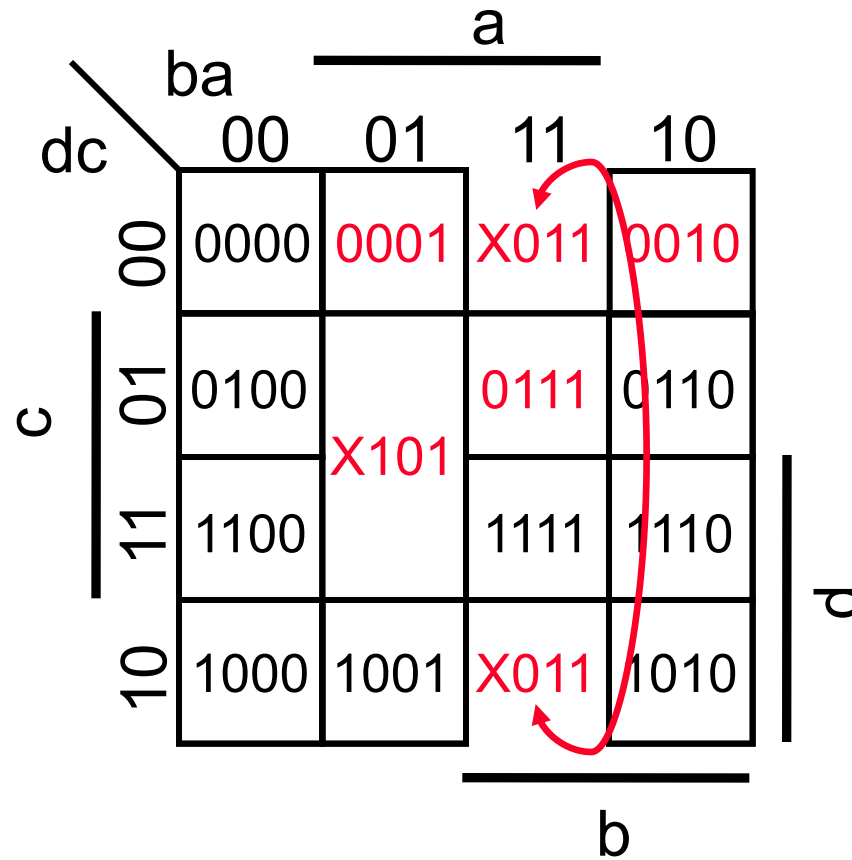
Adjacent minterms differ in exactly one bit
Every positive minterm is an implicant

Karnaugh Map of 4-bit Prime: Larger Implicants

		a			
		ba			
dc	00	01	11	10	
	00	0000	0001	001X	
c	01	0100		0111	0110
	11	1100	X101	1111	1110
	10	1000	1001	1011	1010
		b			
		d			

Can combine adjacent minterms into implicants

Karnaugh Map of 4-bit Prime: Adjacency



Can combine adjacent minterms into implicants
Note edges wrap around

Karnaugh Map of 4-bit Prime: 0XX1

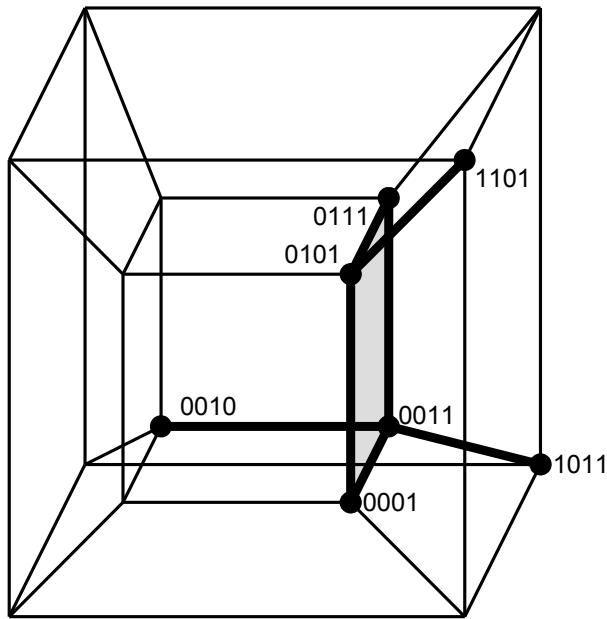
		a				
		ba	00	01	11	10
c	dc	00	0000	0XX1		0010
		01	0100			0110
		11	1100	1101	1111	1110
		10	1000	1001	1011	1010
		b				
		d				

A larger implicant

Why make implicants overlap?

Two reasons:

1. Larger implicants have fewer gates, or gates with fewer inputs.
2. Hazards. More on them later.



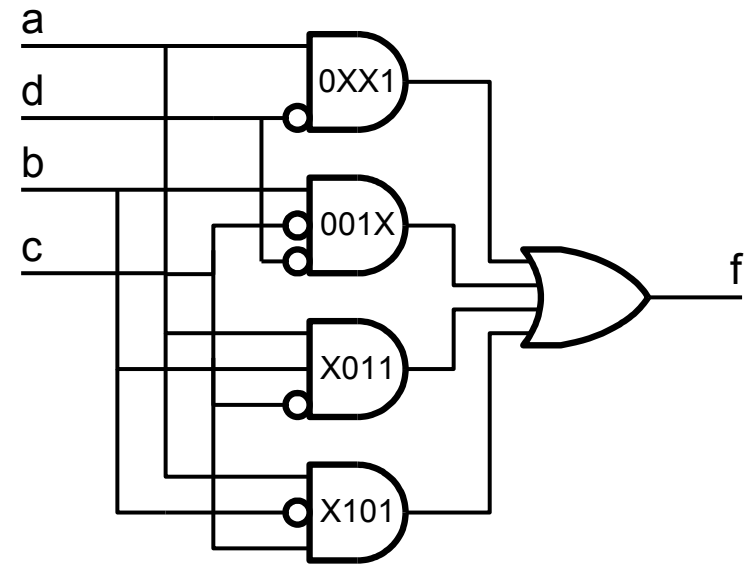
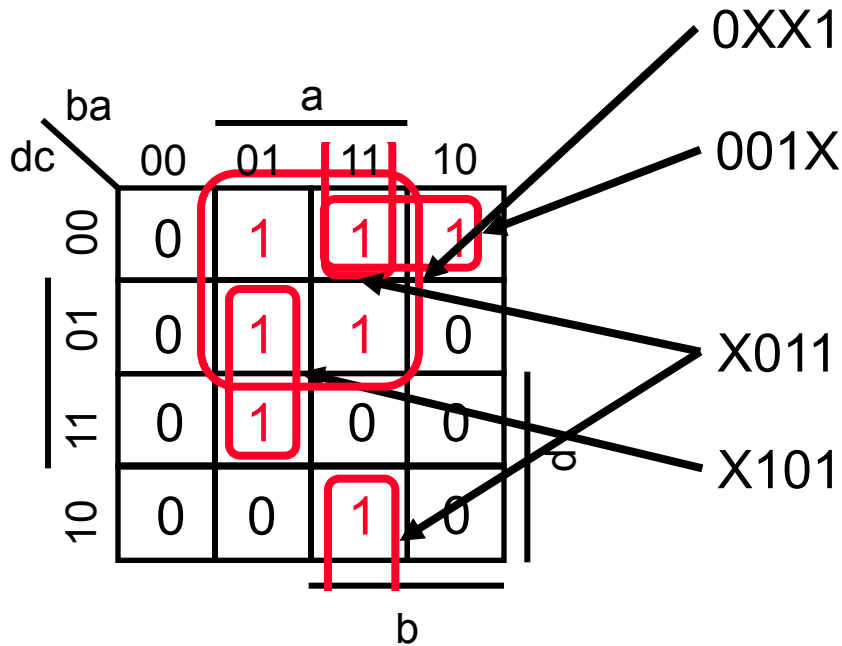
		a			
		00	01	11	10
c	dc	0	1	3	2
	00	4	5	7	6
	01	12	13	15	14
	11	8	9	11	10
		b			

		a			
		00	01	11	10
c	dc	0	1	1	1
	00	0	1	1	0
	01	0	1	0	0
	11	0	0	1	0
		b			

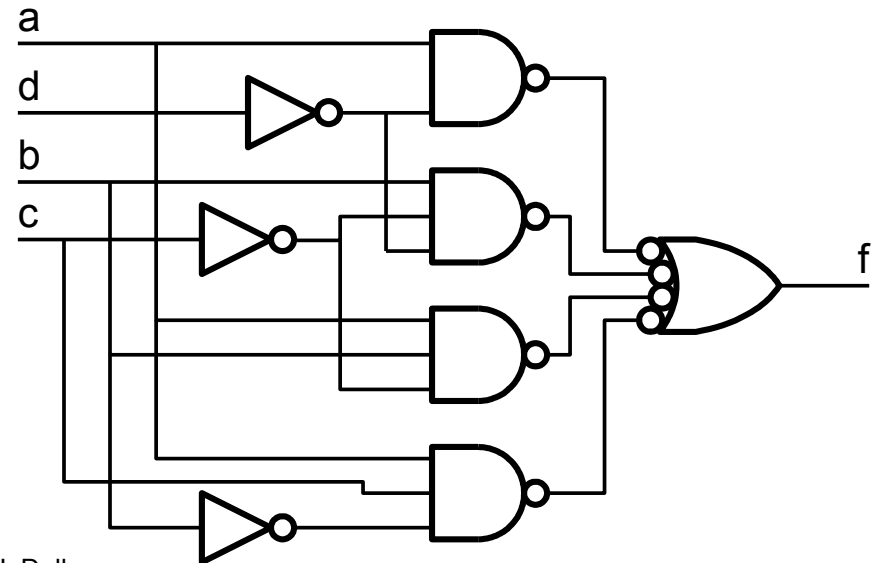
Annotations for the second K-map:

- 0XX1 (points to the top row of 1s)
- 001X (points to the second column of 1s)
- X101 (points to the third column of 1s)
- X011 (points to the bottom-right 1)

dcba
0xx1
001x
x101
x011



In practice, CMOS gates are always inverting, so the real circuit might look like this

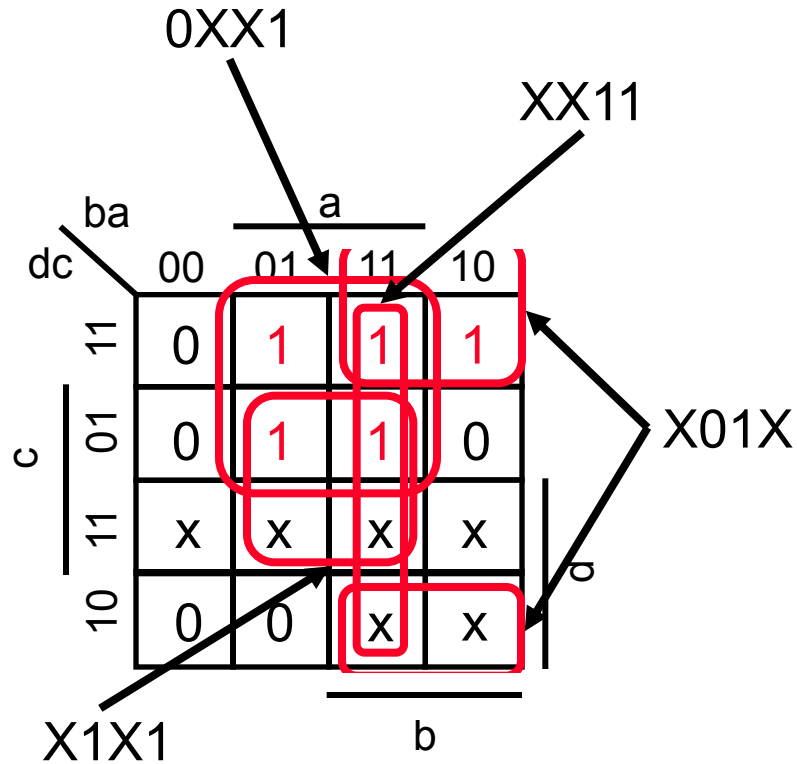


Decimal prime number function – includes don't cares

$$f = \sum_{dcba} m(1,2,3,5,7) + D(10,11,12,13,14,15)$$

dc \ ba		a			
		00	01	11	10
c	00	0	1	1	1
	01	0	1	1	0
	11	x	x	x	x
	10	0	0	x	x
		b			
		d			

Decimal prime number function K-Map

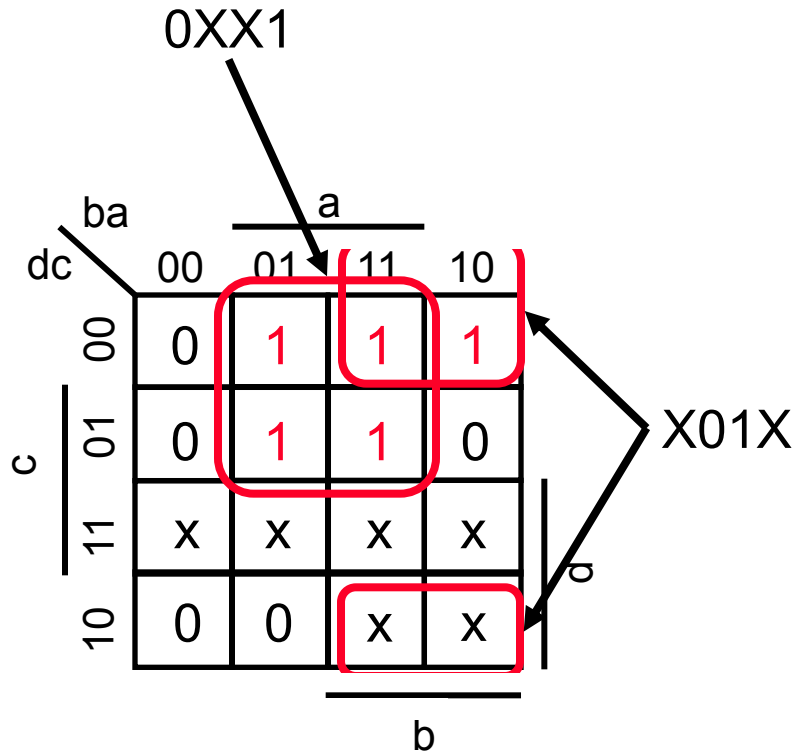


Decimal prime number function – circuit

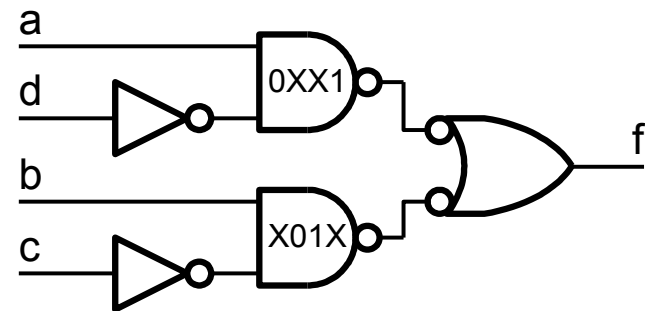
Cover:

0XX1

X01X

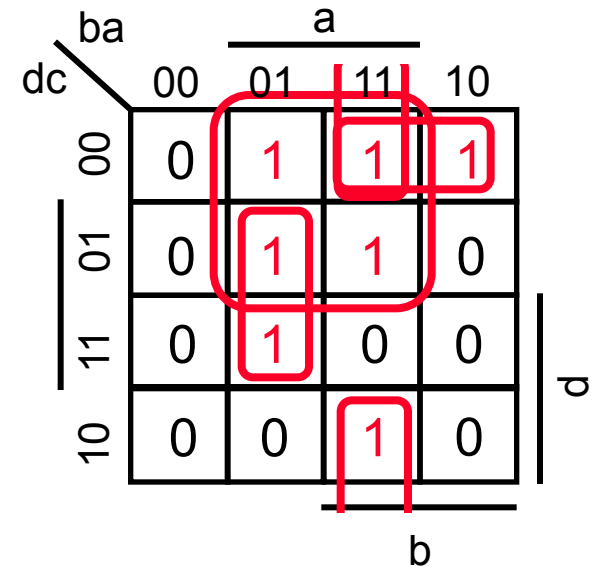


$$f = (a \wedge \bar{d}) \vee (b \wedge \bar{c})$$



Revisiting some definitions (and some new ones)

- Minterm: a product term that includes each input of a circuit or its complement.
- Implicant: a product term that if true implies the function is true.
- Prime Implicant: is an implicant that cannot be made any larger and still be an implicant.
- Essential Prime Implicant: the only prime implicant that contains a particular minterm of the function.
- Distinguished One: is a minterm that is contained in only one implicant.



Product-of-Sums Implementation

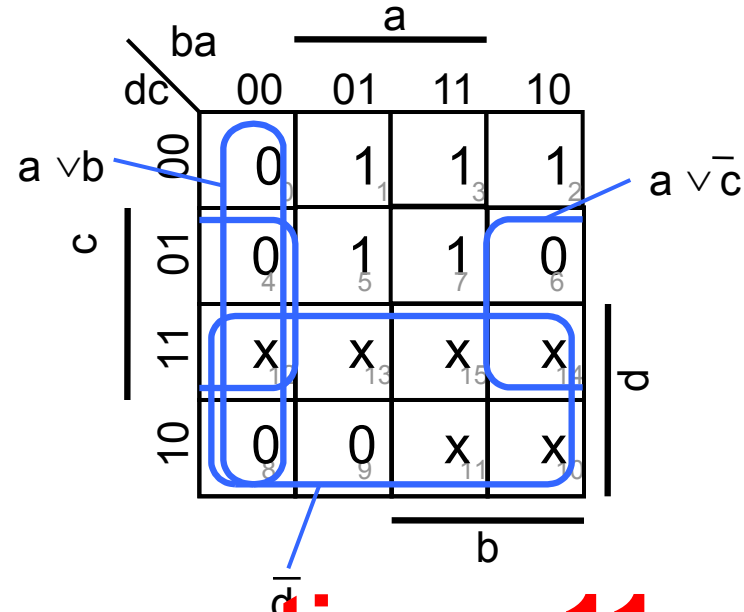
- Sum-of-Products circuit: focus on inputs states where truth table is a 1.
- Product-of-Sums: focus on input states where truth table is a 0.

$$f = \prod_{dcba} M(0,2)$$

		<u>a</u>			
		ba			
dc	00	00	01	11	10
	00	1 ₀	1 ₁	1 ₃	1 ₂
01	00	1 ₄	1 ₅	1 ₇	1 ₆
	01	1 ₁₂	0 ₁₃	0 ₁₅	1 ₁₄
11	00	1 ₈	1 ₉	1 ₁₁	1 ₁₀
	01	1 ₄	1 ₅	1 ₇	1 ₆

$\bar{a} \vee \bar{c} \vee \bar{d}$

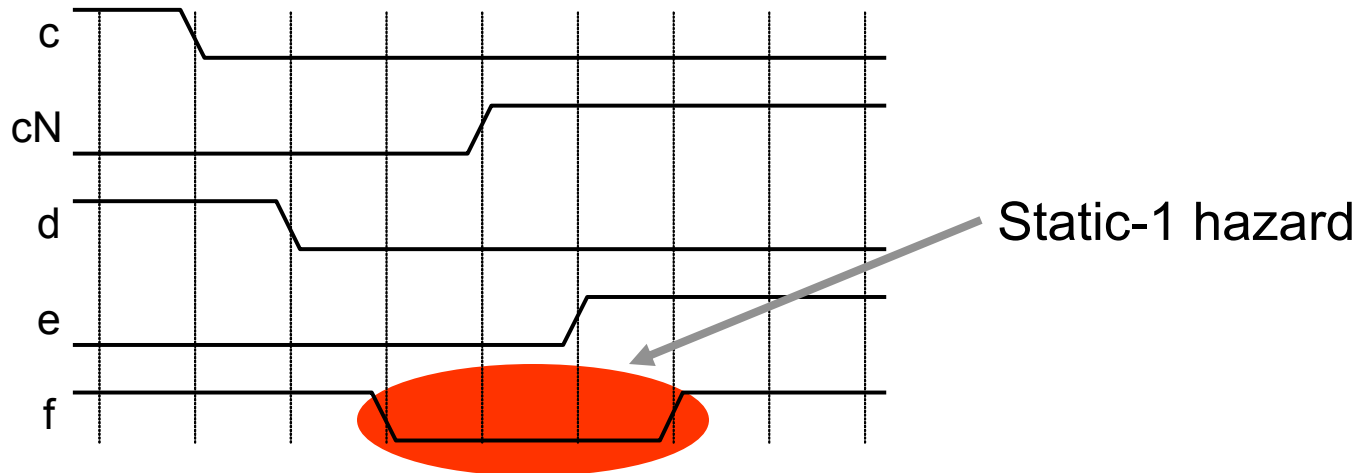
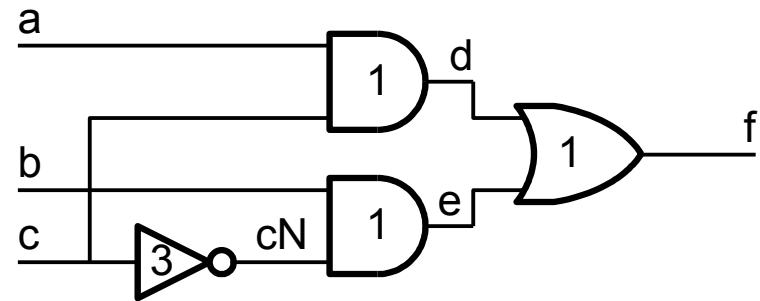
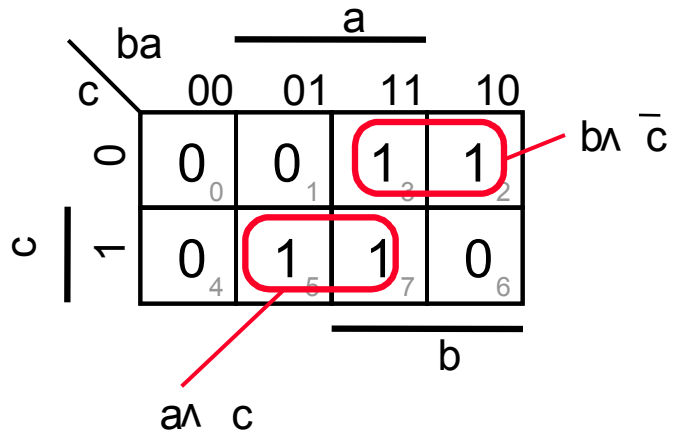
Product-of-Sums Example: Decimal Prime



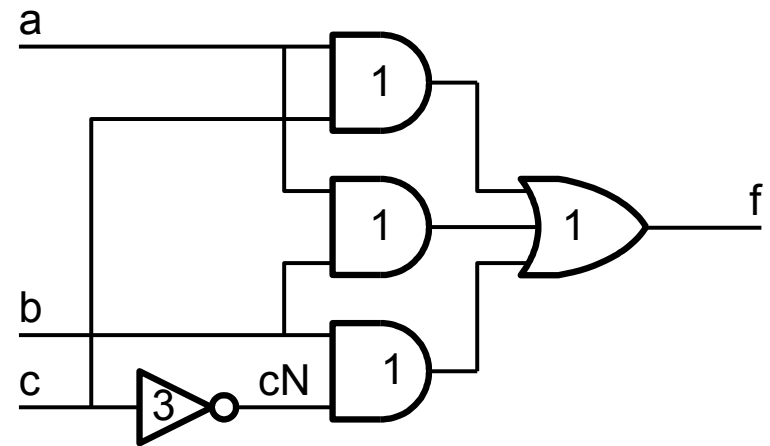
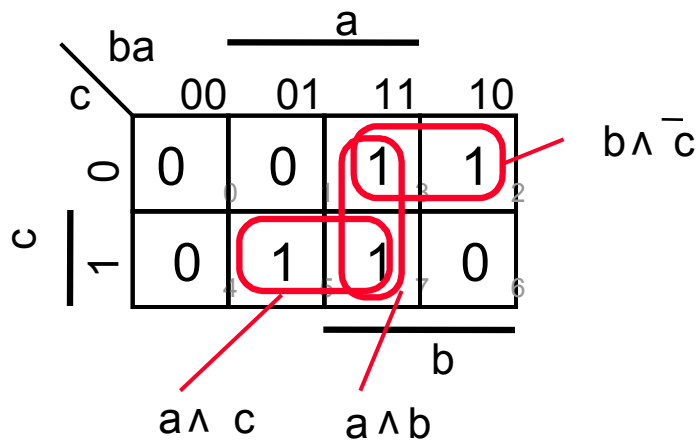
Fix this equation – 11 and 15 in both

$$f = \prod_{dcba} M(6,7,9,11,15) + D(10,11,12,13,14,15)$$

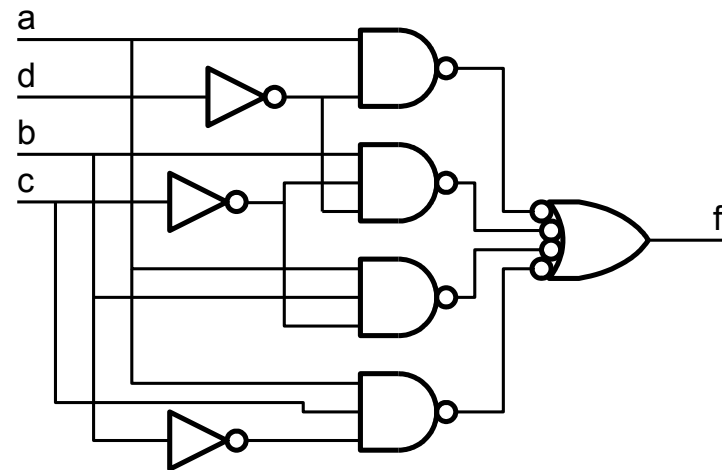
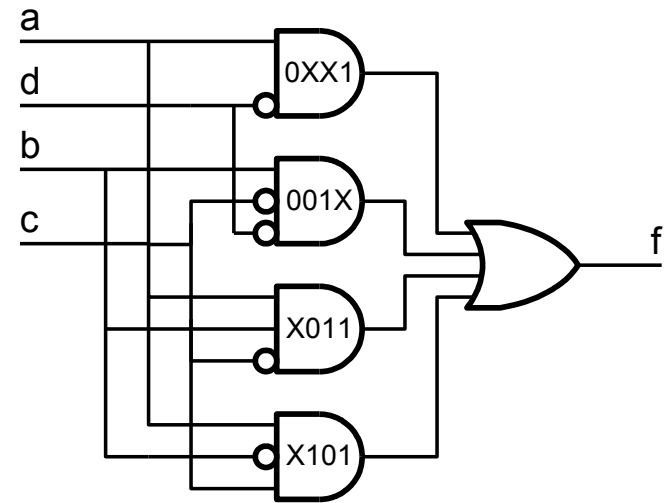
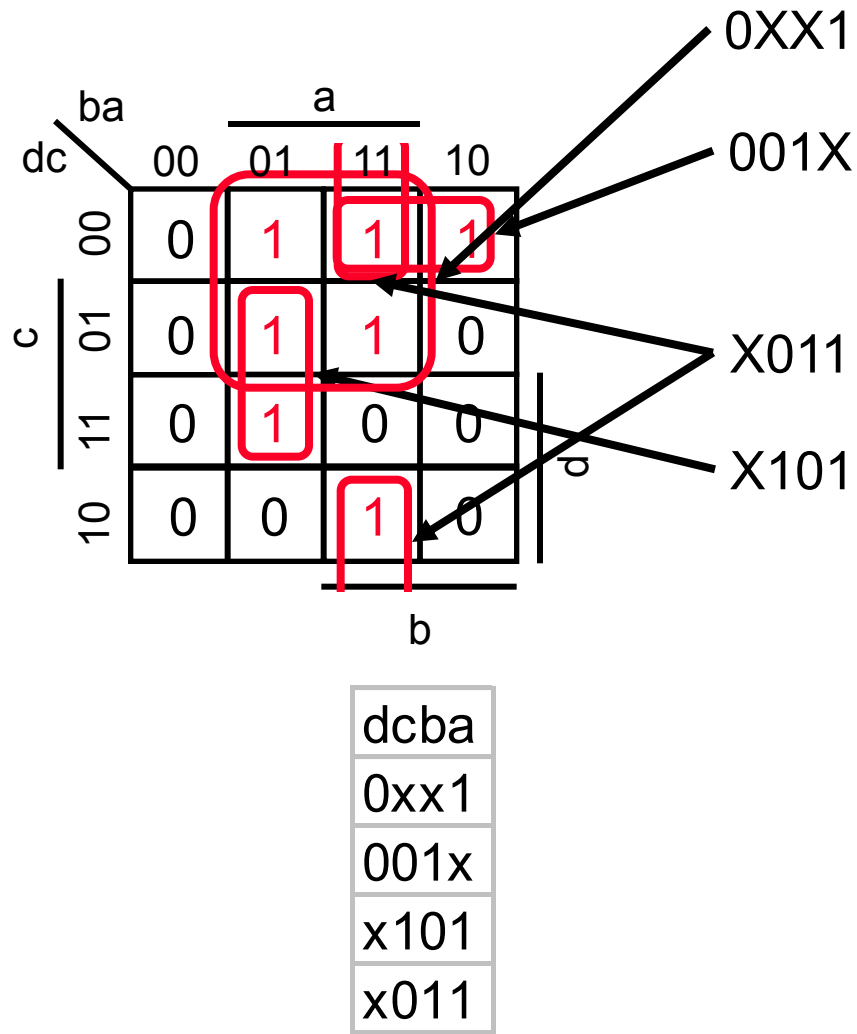
Hazards



Cover transitions to eliminate hazards



Reminder: 4-bit Prime Number Function



4-bit Prime Number Function in Verilog Code – Using case

```
module prime(in, isprime) ;
    input [3:0] in ;           // 4-bit input
    output      isprime ; // true if input is prime
    reg        isprime ;

    always @(in) begin
        case(in)
            1,2,3,5,7,11,13: isprime = 1'b1 ;
            default:         isprime = 1'b0 ;
        endcase
    end
endmodule
```

4-bit Prime Number Function in Verilog Code – Using casex

```
module prime1(in, isprime) ;
    input [3:0] in ;                // 4-bit input
    output      isprime ; // true if input is prime
    reg         isprime ;

    always @(in) begin
        casex(in)
            4'b0xx1: isprime = 1 ;
            4'b001x: isprime = 1 ;
            4'bx011: isprime = 1 ;
            4'bx101: isprime = 1 ;
            default: isprime = 0 ;
        endcase
    end
endmodule
```

4-bit Prime Number Function in Verilog Code – Using assign

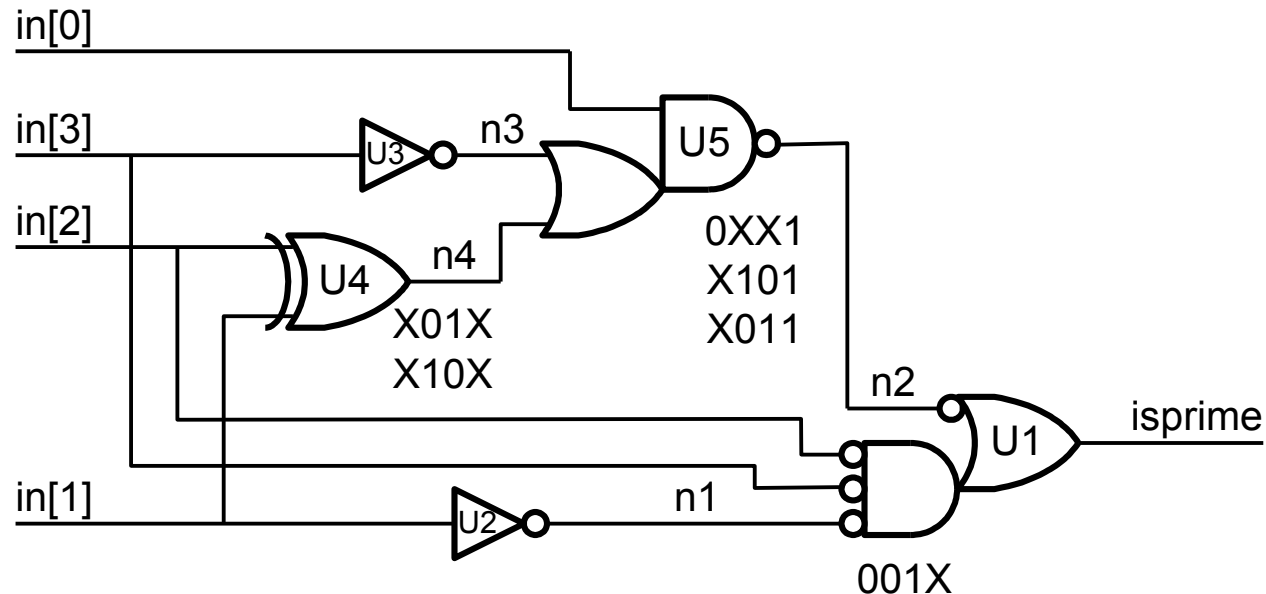
```
module prime(in, isprime) ;  
    input [3:0] in ;           // 4-bit input  
    output      isprime ; // true if input is prime  
  
    wire isprime = (in[0] & ~in[3]) |  
                   (in[1] & ~in[2] & ~in[3]) |  
                   (in[0] & ~in[1] & in[2]) |  
                   (in[0] & in[1] & ~in[2]) ;  
  
endmodule
```

Which is a better description?

- Using case
- Using casex
- Using assign

4-bit Prime Number Function in Verilog Code – Result of synthesizing description using case

```
module prime ( in, isprime );  
input  [3:0] in;  
output isprime;  
    wire n1, n2, n3, n4;  
    OAI13 U1 ( .A1(n2), .B1(n1), .B2(in[2]), .B3(in[3]), .Y(isprime) );  
    INV U2 ( .A(in[1]), .Y(n1) );  
    INV U3 ( .A(in[3]), .Y(n3) );  
    XOR2 U4 ( .A(in[2]), .B(in[1]), .Y(n4) );  
    OAI12 U5 ( .A1(in[0]), .B1(n3), .B2(n4), .Y(n2) );  
endmodule
```



Synthesis Reports

Report : area
Design : prime
Version: 2003.06
Date : Sat Oct 4 11:38:08 2003

Library(s) Used:

XXXXX

Number of ports: 5
Number of nets: 9
Number of cells: 5
Number of references: 4

Combinational area: 7.000000
Noncombinational area: 0.000000
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 7.000000
Total area: undefined

Report : timing
-path full
-delay max
-max_paths 1
Design : prime
Version: 2003.06
Date : Sat Oct 4 11:38:08 2003

Operating Conditions:
Wire Load Model Mode: enclosed

Startpoint: in[2] (input port)
Endpoint: isprime (output port)
Path Group: (none)
Path Type: max

Des/Clust/Port	Wire Load Model	Library
prime	2K_5LM	XXXXX

Point	Incr	Path
-		
input external delay	0.000	0.000 r
in[2] (in)	0.000	0.000 r
U4/Y (EX210)	0.191	0.191 f
U5/Y (BF051)	0.116	0.307 r
U1/Y (BF052)	0.168	0.475 f
isprime (out)	0.000	0.475 f
data arrival time		0.475

-
(Path is unconstrained)

Constraint File

```
create_clock "clk" -name clk -period 2 -waveform {0 1.7}
set_clock_uncertainty 0.2 clk
set_fix_hold all_clocks()
set_load -pin_load 5 {isprime}

set_input_delay 0.5 -clock clk {in}

set_output_delay -max 0.8 -clock clk {isprime}
```

Test bench

```
module test_prime ;
    reg [3:0] in ;
    wire isprime ;

    // instantiate module to test
    prime p0(in, isprime) ;

    initial begin
        in = 0 ;
        repeat (16) begin
            #100
            $display("in = %2d isprime = %1b",in,isprime) ;
            in = in+1 ;
        end
    end
endmodule
```

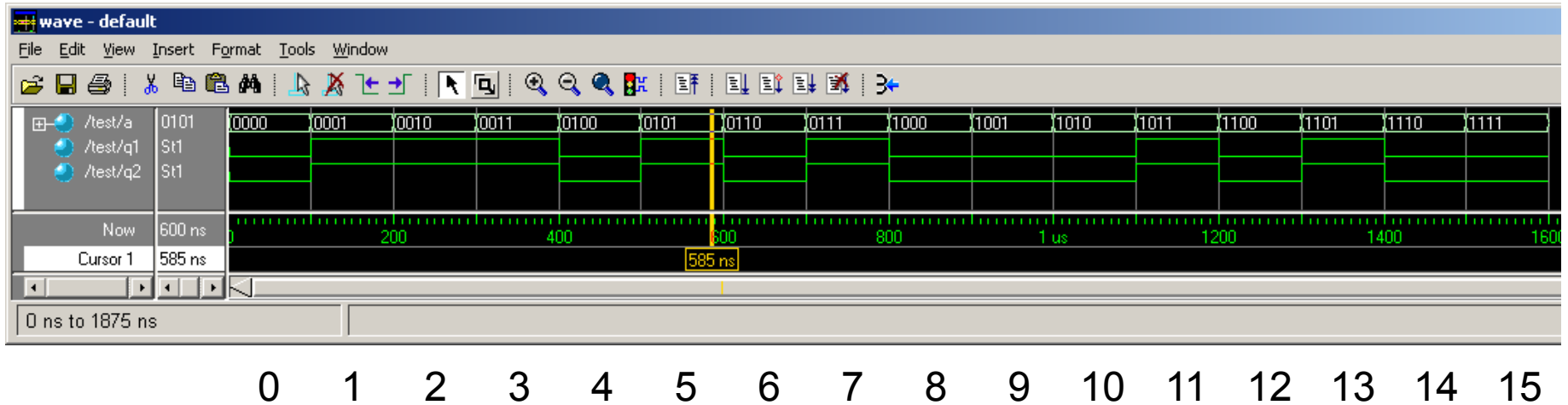
Test benches use a very different style of verilog

- Initial statements
 - \$display
 - Repeat and other looping constructs
 - #delay
-
- Don't use these constructs in synthesizable modules

Testing Result

```
# in = 0 isprime = 0
# in = 1 isprime = 1
# in = 2 isprime = 1
# in = 3 isprime = 1
# in = 4 isprime = 0
# in = 5 isprime = 1
# in = 6 isprime = 0
# in = 7 isprime = 1
# in = 8 isprime = 0
# in = 9 isprime = 0
# in = 10 isprime = 0
# in = 11 isprime = 1
# in = 12 isprime = 0
# in = 13 isprime = 1
# in = 14 isprime = 0
# in = 15 isprime = 0
```

Wave output



```

module prime_dec(in, isprime) ;
    input [3:0] in ;           // 4-bit input
    output      isprime ;     // true if input is prime
    reg         isprime ;

    always @(in) begin
        casex(in)
            0,4,6,8,9: isprime = 1'b0 ;
            1,2,3,5,7: isprime = 1'b1 ;
            default:   isprime = 1'bx ;
        endcase
    end
endmodule

```


Summary

- To minimize logic
 - Write K-map
 - Find all prime implicants
 - Pick a minimal set of prime implicants that *covers* the function
- Hazards (output glitches) can be eliminated by covering transitions
- Verilog
 - Can represent with case, casex, assign, or structurally
 - Use representation that is readable and maintainable
 - case for truth tables
 - assign for equations
 - Don't try to do the logic design yourself
 - Synthesis tool will do the optimization
 - Test benches check that implementation meets its specification
 - Test benches use a different style of Verilog

Next Time

- Combinational building blocks