EE292: Fundamentals of ECE

Fall 2012

TTh 10:00-11:15 SEB 1242

Lecture 22

121115

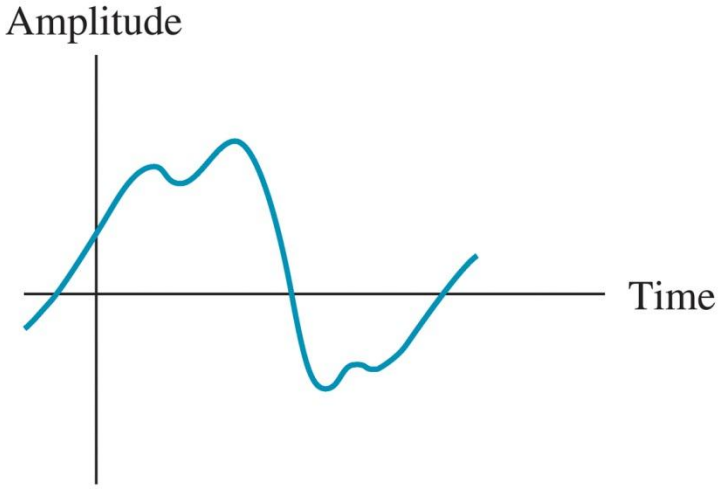
http://www.ee.unlv.edu/~b1morris/ee292/

Outline

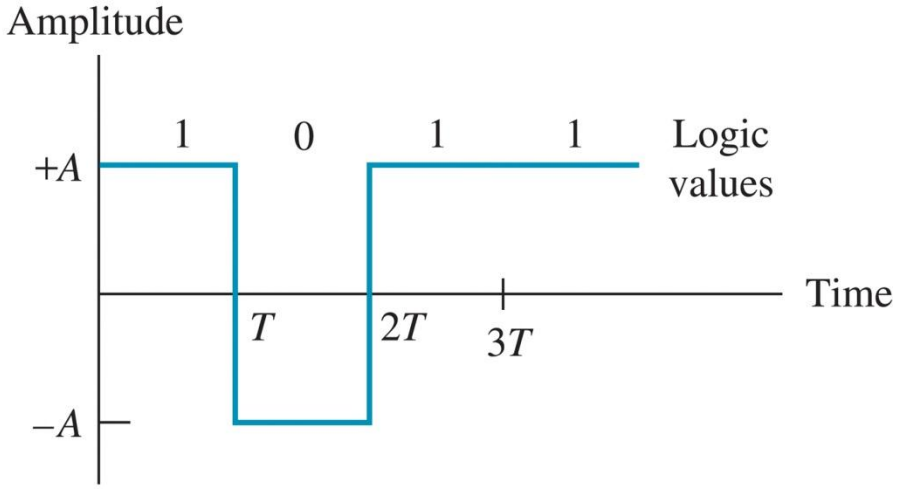
- Review
  - Binary Number Representation
  - Binary Arithmetic
- Combinatorial Logic

Digital Signals

- A signal with discrete "time" variable and only a few restricted amplitude values



(a) Analog signal



(b) Digital signal

- Binary signals are the most common type of signal
  - The output takes only two possible values
  - The two output values are often given positive "logical values" of a 1 (high) or 0 (low)

Digital Words

- Bit – a single binary digit
  - Smallest amount of information that can be represented in a digital system
  - Represents a yes/no for a digital variable
  - E.g. $R = 0$, represents not raining while $R = 1$, represents raining
- In order to represent more complex information, bits can be combined into digital words
  - A byte is 8 bits and a nibble is 4 bits (used often in computers, e.g. a byte to represent each key on a keyboard)
- Example RWS
  - R for rain, W for wind, S for sunny
  - $RWS = 110$ indicates it is raining, with winds, and cloudy (e.g. not sunny)

Positional Notation for Numbers

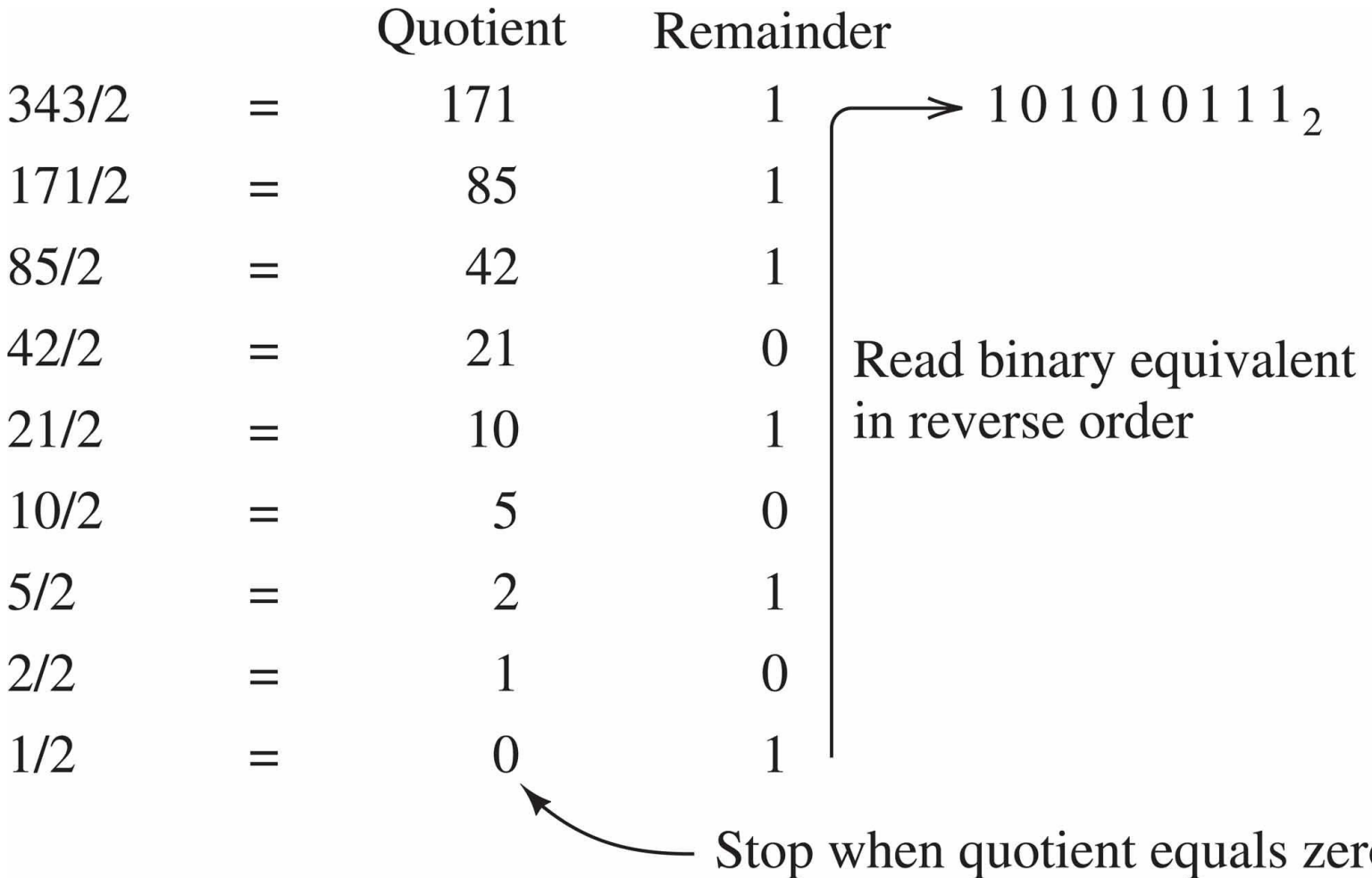
- Base B number →B symbols per digit
  - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6 , 7, 8, 9
  - Base 2 (binary) 0, 1
- Number representation
  - $d_{N-1}d_{N-2}\ldots d_2d_1d_0$ is N digit number
    - 2^N different numbers can be represented
  - Value $=d_{N-1}\times B^{N-1} + d_{N-2}\times B^{N-2} + \ldots + d_1\times B^1 + d_0\times B^0$
- Examples
  - (Decimal): 90
    - $= 9\times10^1 + 0\times10^0$
  - (Binary): 1011010
    - $= 1\times2^6 + 0\times2^5 + 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$
    - $= 64 + 16 + 8 + 2$
    - $= 90$
  - 7 binary digits needed for 2 digit decimal number

Conversion from Decimal to Base B

- Integer conversion is done by repeatedly dividing by the decimal number by base B
  - The remainder is a base B digit
  - Continue dividing until quotient equals zero
  - Arrange into digital word from right to left

Conversion from Decimal to Binary

- Convert decimal 343_{10} to binary (base 2)



Hexadecimal Number: Base 16

- More human readable than binary
- Base with easy conversion to binary
  - Any multiple of 2 base could work (e.g. octal)
- Hexadecimal digits

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Hex (16)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
octal (8)	0	1	2	3	4	5	6	7								

  - 1 hex digit represents 16 decimal values or 4 binary digits
    - Will use `0x` to indicate hex digit

Hex/Binary Conversion

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

- Convert between 4-bits and a hex digit using the conversion table above
- Examples
  - `1010 1100 0101` (binary)
    - = `0xAC5`
  - `10111` (binary)
    - = `0001 0111` (binary)
    - = `0x17`
  - `0x3F9`
    - = `0011 1111 1001` (binary)
    - = `11 1111 1001` (binary)

Binary Arithmetic

- Addition in binary is the same as with decimal
  - Only have 2 values (0, 1) in binary

		Sum	Carry
0 + 0	=	0	0
0 + 1	=	1	0
1 + 1	=	0	1
1 + 1 + 1	=	1	1

Signed Numbers

- N bits represents 2^N values
- Unsigned integers
  - Range `[0, 2^N-1]`
- How can negative values be indicated?
  - Use a sign-bit
  - Boolean indicator bit (flag)

Sign and Magnitude

- 16-bit numbers
  - +1 (decimal) = 0000 0000 0000 0001 = 0x0001
  - -1 (decimal) = 1000 0000 0000 0001 = 0x8001
- Problems
  - Two zeros
    - 0x0000
    - 0x8000
  - Complicated arithmetic
    - Special steps needed to handle when signs are same or different (must check sign bit)

Ones Complement

- Complement the bits of a number
  - +1 (decimal) = `0000 0000 0000 0001 = 0x0001`
  - -1 (decimal) = `1111 1111 1111 1110 = 0xFFFE`
- Positive number have leading zeros
- Negative number have leading ones
- Arithmetic not too difficult
- Still have two zeros

Two's Complement

- Subtract large number from a smaller one
  - Borrow from leading zeros
  - Result has leading ones
- Unbalanced representation
  - Leading zeros for positive
    - 2^{N-1} non-negatives
  - Leading ones for negative number
    - 2^{N-1} negative number
  - One zero representation
- First bit is sign-bit (must indicate width)
  - Value = $d_{31} \times -2^{31} + d_{30} \times 2^{30} + \ldots + d_1 \times 2^1 + d_0 \times 2^0$

Binary	Decimal
… 0011	3
… 0100	4
… 1111	-1

Negative value for sign bit

Two's Complement Negation

- Shortcut = invert bits and add 1
  - Number + complement = 0xF..F = -1
    - $x + \bar{x} = -1$
    - $\bar{x} + 1 = -x$
- Example

x	1111 1110
$\bar{x}$	0000 0001
$\bar{x} + 1$	0000 0010

Two's Complement Sign Extension

- Machine's have fixed width (e.g. 32-bits)
  - Real numbers have infinite width (invisible extension)
    - Positive has infinite 0's
    - Negative has infinite 1's
- Replicate sign bit (msb) of smaller container to fill new bits in larger container
- Example
  - 1111 1111 1111 1111 → 1111 1111 1111 1110 1111 1111 1111 1110

Overflow

- Fixed bit width limits number representation
- Occurs if result of arithmetic operation cannot be represented by hardware bits
- Example
  - 8-bit: 127 + 127

Binary	Decimal
0111 1111	127
0111 1111	127
1111 1110	**-2 (254)**

Two's Complement Subtraction

- Subtraction $(x - y)$ can be performed by adding x and the two's complement of y
- Example
  - 8-bit 29 - 27

Binary	Decimal
0001 1101	29
0001 1011	27
1110 0100	$\overline{27}$
1110 0101	$\overline{27} + 1 = -27$
0000 0010	2

Combinatorial Logic Circuits

- Combine logical variable inputs to produce a logic-variable output
- Logic can be specified by enumerating the output for all possible input combinations in a truth table

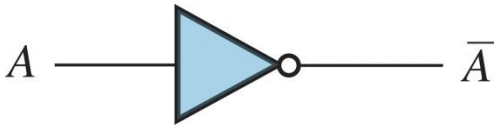
- Considered memoryless circuits
  - The output at a given time instant are only dependent on the input at the same time instant

Inverter

- NOT operation inverts a logic variable
  - We have seen this as complement already
    - $\text{NOT}(A) = \bar{A}$

A	$\bar{A}$
0	1
1	0

(a) Truth table



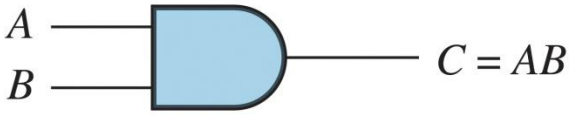
(b) Symbol for an inverter

AND Gate

- Logic gate computes the logical multiplication of input variables
  - $\text{AND}(A, B) = AB$

A	B	$C = AB$
0	0	0
0	1	0
1	0	0
1	1	1

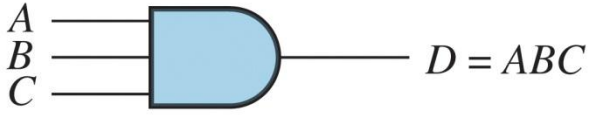
(a) Truth table



(b) Symbol for two-input AND gate

A	B	C	$D = ABC$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(a) Truth table



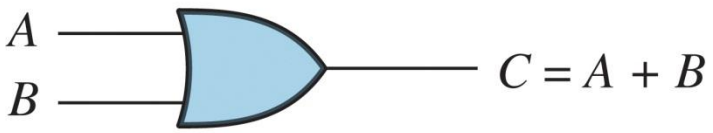
(b) Symbol for three-input AND gate

OR Gate

- Logic gate computes the logical addition of input variables
  - $\mathrm{OR}(A, B) = A + B$

A	B	$C = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

(a) Truth table

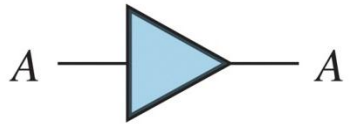


(b) Symbol for two-input OR gate

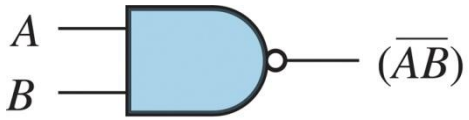
More Logic Gates

- In silicon chips, other gates are simpler to implement
- Buffer = NOT followed by NOT
  - Returns the same value
- NAND = AND followed by NOT
- NOR = OR followed by NOT
- XOR is the exclusive-OR gate
  - $\text{XOR}(A, B) = A \oplus B$

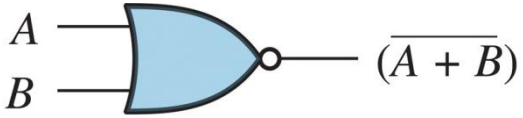
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



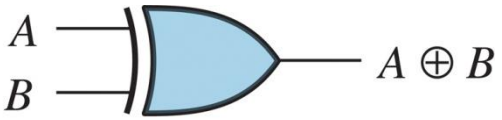
(d) Buffer



(a) NAND gate



(b) NOR gate



(c) XOR gate

Boolean Algebra

- Mathematical theory of logical variables
- Use basic AND, OR, and NOT relationships to prove a Boolean expression
  - Can generate a truth table to specify the output relationship for all possible input values

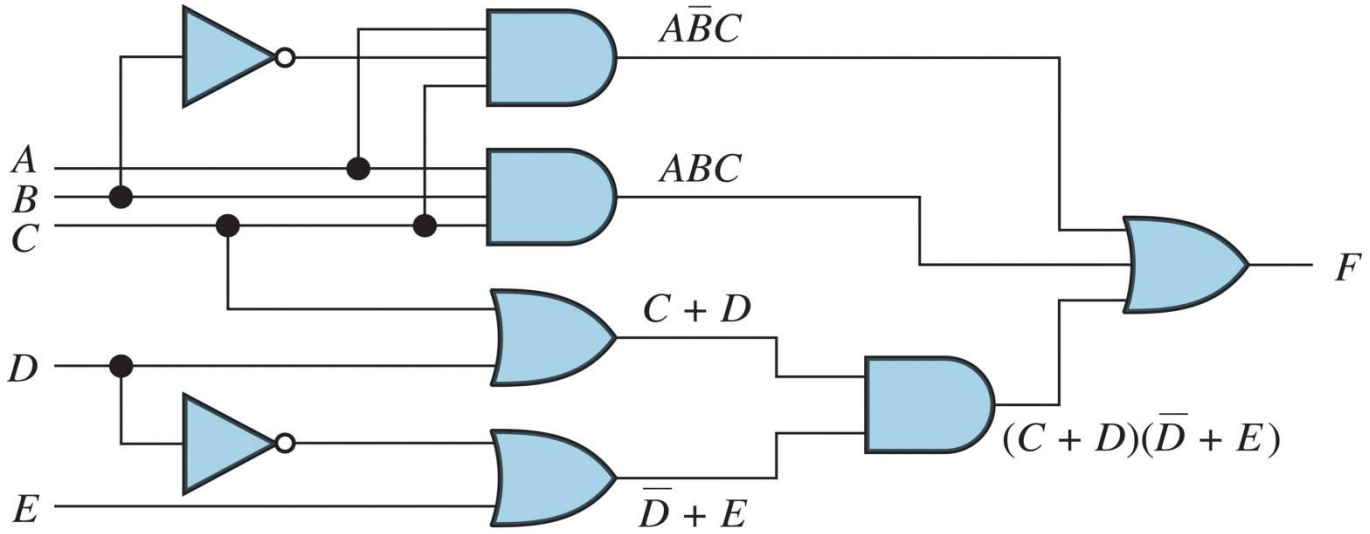
De Morgan's Laws

- Provides a way to convert an AND relationship into an OR relationship and vice versa
- $ABC = \overline{\bar{A} + \bar{B} + \bar{C}}$
- $(A + B + C) = \overline{\bar{A}\bar{B}\bar{C}}$
- Implications:
  - Any logic function can be implemented using AND gates and inverters
  - Any logic function can be implemented using OR gates and inverters
  - Only need either AND gates or OR gates (not both)

Implementation of Boolean Expressions

- A logical variable can be composed of Boolean relationships
  - AND, OR, NOT, etc.
- Gate level implementation is straightforward

- Example
- $F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$



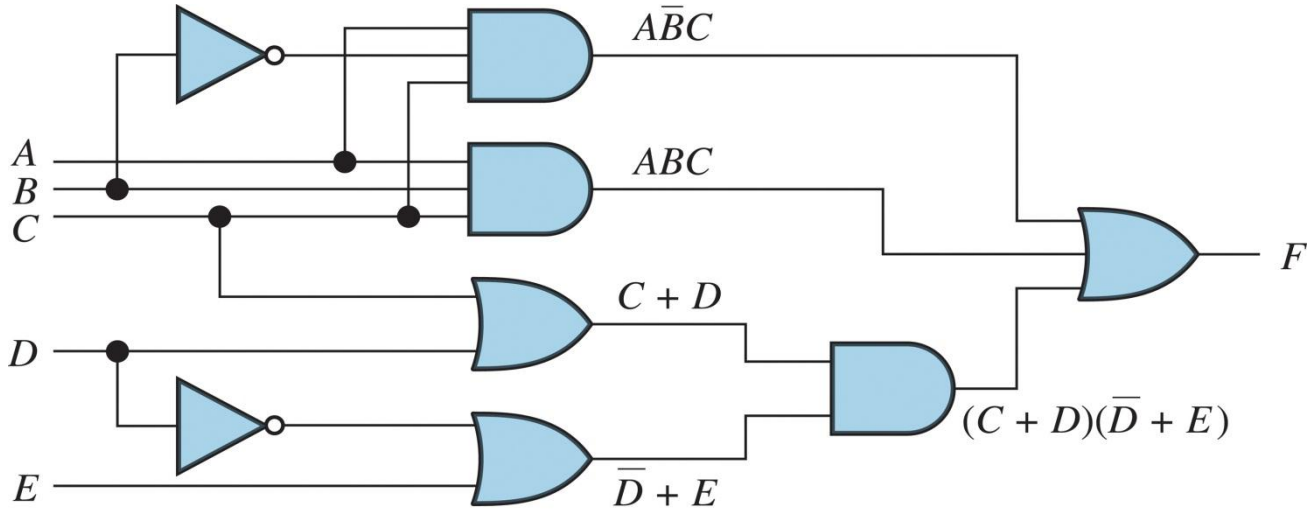
Simplifying Boolean Expression

- Find simpler equivalent expressions by manipulation of equation and Boolean relations

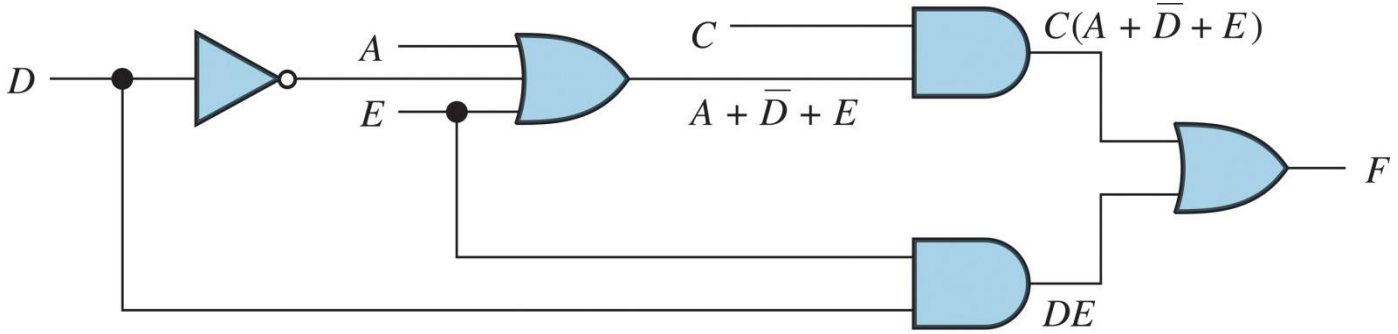
- Example
- $F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$
- $F = A\bar{B}C + ABC + (C\bar{D} + CE + D\bar{D} + DE)$
- $F = A\bar{B}C + ABC + (C\bar{D} + CE + 0 + DE)$
- $F = AC(\bar{B} + B) + (C\bar{D} + CE + 0 + DE)$
- $F = AC(1) + (C\bar{D} + CE + 0 + DE)$
- $F = C(A + \bar{D} + E) + DE$

Comparison of Implementations

- $F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$



- $F = C(A + \bar{D} + E) + DE$



Synthesis of Logic

- Require methods to convert logic circuit specifications into a practical gate level implementation
  - Often the logic is specified in a natural language

- We will use truth tables to develop Boolean logic expressions that can be implemented with gates

Example Truth Table

- 3 Logical variable input and 1 output
  - Enumerate all possible input values and specify the corresponding output
  - A, B, C input and D output

A	B	C	D
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Sum-of-Products Implementation

- Find all output rows that have a 1 output
  - Determine AND relationship between inputs
- OR the AND terms from each row

A	B	C	D		AND Term
0	0	0	1		$\bar{A}\bar{B}\bar{C}$
0	0	1	0		
0	1	0	1		$\bar{A}B\bar{C}$
0	1	1	0		
1	0	0	0		
1	0	1	0		
1	1	0	1		$AB\bar{C}$
1	1	1	1		ABC

Can this be simplified?

$$D = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + ABC$$

Product-of-Sums Implementation

- Find all output rows that have a 0 output
  - Determine OR relationship between inputs
- AND the OR terms from each row

A	B	C	D		**OR Term**
0	0	0	1		
0	0	1	0		$A + B + \bar{C}$
0	1	0	1		
0	1	1	0		$A + \bar{B} + \bar{C}$
1	0	0	0		$\bar{A} + B + C$
1	0	1	0		$\bar{A} + B + \bar{C}$
1	1	0	1		
1	1	1	1		

$$D = (A + B + \bar{C})(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + B + \bar{C})$$