* ASCII Sending Receiving
  + "A" -> 01000001 even error->odd
  + "$" -> 00100100
  + Partity bit is the leading bit for error detectiojn
  + Binary Cell can only hold 1 bit of information
  + N\_binary cells -> holds n-bits
* Binary Logic
  + Defintion – consists of binary variable (a,b,c,x,y,z) and a set of logic operations ( and, or, not)
  + 1) And (dot) for the symbol
    - X and y = z
    - X (dot) y = z
    - X y = z

|  |  |  |
| --- | --- | --- |
| x | y | z |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* + 2) or +
    - X or y = z
    - X + y = z

|  |  |  |
| --- | --- | --- |
| x | y | z |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

* + 3) Not
    - Not (x) = z
    - X (with bar over it) = z
    - X' = z

|  |  |
| --- | --- |
| x | z |
| 0 | 1 |
| 1 | 0 |

* + (look up basic and gates, or gates and not gates)
  + Circuit analysis
    - X - 01100
    - Y - 00110
    - And: x\*y = 00100
    - Or: x+y = 01110
    - Not(x) = 10011
    - End of chapter 1
* Chapter 2 Boolean Alegbra
* Closure – set is closed to "+" operation if a,b (set) S. And a + b = c (set) S. then S is closed to + operator
  + Ex: N = {1,2,3,….}
    - A) "+" 1 + 2 = 3 (set) N: Therefore N is closed to "+" operator
    - B) "-" 1 –2 = -1 (Not set) N: therefore n is not closed to "-" operator
  + Ex: Boolean algebra: B { 0,1 }

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X y | X\*y | X+y | X' | Y' |
| 0 0 | 0 | 0 | 1 | 1 |
| 01 | 0 | 1 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 0 |

* + - B is closed to the and, or, not operators
* Associative Law
  + (X\*Y) \* z = X\*(Y\*Z)
  + (X+)+Z = X+(Y+Z)
  + Ex : x = 1 y = 0 z = 1
    - (1\*0)\*1 = 1\*(0\*1)
    - (1+0)+1 = 1+(0+1)
* Commutative law
  + X\*y = y\*x
  + X+Y = y+x
* Identity
  + X + 0 = x -> "0" for or(+)
  + X \* 1 = x -> "1" for and(\*)
* Inverse
  + X \* X' = 0
  + X + X' = 1
* Distributive Law
  + X\*(Y\*Z) = (X\*Y)\*(X\*Z)
* Idempotent
  + X + X = X
  + X \* X = X
* Involution
  + (x')' = Not(not(x)) = x
* DeMorgan Thm
  + A) Not(x+y) = Not(x) \* Not(Y)
  + B) not(x\*Y) = not(x) + not(y)
* Domination
  + 1 + x = 1
  + 0 \* x = 0
* Absorption
  + X + xy = x
    - X(1+y) where (1 + y ) is always 1
    - X \* 1 (where that is always x)
    - X
* Operator Precedence
  + Parenthesis
  + Not
  + And
  + Or
* Boolean Functions
  + F(1) = x'y'z + x'yz + xy'
  + F(2) = xy' + x'z

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| X y z | X' y' z' | X' y' z | X' y z | X y' | X' z | F1 | F2 |
| 000 | 111 | 0 | 0 | 0 | 0 | 0 | 0 |
| 001 | 110 | 1 | 0 | 0 | 1 | 1 | 1 |
| 010 | 101 | 0 | 0 | 0 | 0 | 0 | 0 |
| 011 | 100 | 0 | 1 | 0 | 1 | 1 | 1 |
| 100 | 011 | 0 | 0 | 1 | 0 | 1 | 1 |
| 101 | 010 | 0 | 0 | 1 | 0 | 1 | 1 |
| 110 | 001 | 0 | 0 | 0 | 0 | 0 | 0 |
| 111 | 000 | 0 | 0 | 0 | 0 | 0 | 0 |

* + F(1) = x'y'z + x'yz + xy'
    - X'z(y'+y) + xy' where (y'+y) is always 1
    - X'z\*1 + xy'
    - X'z + xy' which is equal to F(2)
    - Look at figure 2.2 in the book
* Boolean functions can be represented in 3 ways
  + Algebraic expression
  + Truth table
  + Logic circuit (ckt)