## Chapter 1

### turning binary, octal, hexa, etc. back into real number

$(FML)_n = L \times n^0 + M \times n^1 + F \times n^2 =$ number value

$(FML.FML)_n = L \times n^0 + M \times n^1 + F \times n^2 + F \times n^{-1} + M \times n^{-2} + L \times n^{-3}$

2 1 0 -1 -2 -3

### converting base 10 → 2 (or any other)

$(n)_{10} \rightarrow (ABCD)_2$

$n \,|\, 2$ → ① ⓪ (remainder) → backwards

- hexa will have huge remainders. use the table (up to 15)

For decimals, take just the decimal portion (ignore whole num), multiply instead of divide; if it goes over 1, ignore the 1 → forwards

| (10) decimal | (2) binary | (8) octal | (16) hexa |
|---|---|---|---|
| 00 | 0000 | 00 | 0 |
| 01 | 0001 | 01 | 1 |
| 02 | 0010 | 02 | 2 |
| 03 | 0011 | 03 | 3 |
| 04 | 0100 | 04 | 4 |
| 05 | 0101 | 05 | 5 |
| 06 | 0110 | 06 | 6 |
| 07 | 0111 | 07 | 7 |
| 08 | 1000 | 10 | 8 |
| 09 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

### Hexa to binary

$(ABCD)_{16}$ → $(xxxx\ xxxx\ xxxx\ xxxx)_2$

### Binary to hexa

same but backwards. use the table

### 2's complement

- if 2's complement is added and they have the same sign, overflow occurs if the result has opposite sign.
- overflow will never occur with adding opposite signs.

### signed binary nums

| decimal | 2's | 1's | signed mag. |
|---|---|---|---|
| +7 | 0111 | 0111 | 0111 |
| +6 | 0110 | 0110 | 0110 |
| +5 | 0101 | 0101 | 0101 |
| +4 | 0100 | 0100 | 0100 |
| +3 | 0011 | 0011 | 0011 |
| +2 | 0010 | 0010 | 0010 |
| +1 | 0001 | 0001 | 0001 |
| +0 | 0000 | 0000 | 0000 |
| -0 | — | 1111 | 1000 |
| -1 | 1111 | 1110 | 1001 |
| -2 | 1110 | 1101 | 1010 |
| -3 | 1101 | 1100 | 1011 |
| -4 | 1100 | 1011 | 1100 |
| -5 | 1011 | 1010 | 1101 |
| -6 | 1010 | 1001 | 1110 |
| -7 | 1001 | 1000 | 1111 |
| -8 | 1000 | — | — |

+1

if negative, leave first LS ① and flip all else except sign

if negative, flip all except sign

### Binary-Coded decimal

| decimal | BCD |
|---|---|
| 0 | 0000 |
| ⋮ | ⋮ |
| 9 | 1001 |

- we use binary arithmetic to add the digits. If result is more than 9 (1001), add 6 (+0110) to result, and add 1 to next significant digit

## Chapter 2

AND is denoted by "·" Ex: $x \cdot y = x$ AND $y$

OR is denoted by "+" Ex: $x + y = x$ OR $y$

NOT is denoted by "$x'$" or "$\bar{x}$" or "$\neg x$"

Ex: $x = 0 \rightarrow x' = 1$, $x = 1 \rightarrow x' = 0$

SOP: minterm

POS: maxterm

| AND | OR | NOT |
|---|---|---|
| $0 \cdot 0 = 0$ | $0+0=0$ | $0' = 1$ |
| $0 \cdot 1 = 0$ | $0+1=1$ | $1' = 0$ |
| $1 \cdot 0 = 0$ | $1+0=1$ | |
| $1 \cdot 1 = 1$ | $1+1=1$ | |

| OR | $(x')' = x$ | AND |
|---|---|---|
| $x+0=x$ | | $x \cdot 1 = x$ |
| $x+x'=1$ | | $x \cdot x' = 0$ |
| $x+x=x$ | | $x \cdot x = x$ |
| $x+1=1$ | | $x \cdot 0 = 0$ |
| $x+y=y+x$ | | $xy = yx$ |
| $x+(y+z)=(x+y)+z$ | | $x(yz)=(xy)z$ |
| $x(y+z)=xy+xz$ | | $x+yz=(x+y)(x+z)$ |
| $(x+y)'=x'y'$ | | $(xy)'=x'+y'$ |
| $x+xy=x$ | | $x(x+y)=x$ |

term + term + term

literal literal literal literal

3 terms, 4 literals

## Chapter 3

### 3-variable

| x \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| 1 | $m_4$ | $m_5$ | $m_7$ | $m_6$ |

### 4-variable

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| 01 | $m_4$ | $m_5$ | $m_7$ | $m_6$ |
| 11 | $m_{12}$ | $m_{13}$ | $m_{15}$ | $m_{14}$ |
| 10 | $m_8$ | $m_9$ | $m_{11}$ | $m_{10}$ |

### Prime implicant

- the max # of squares in every land

### Exclusive-OR (XOR)

$x \oplus y = xy' + x'y$

| x | y | $F = x \oplus y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$F = x'y + xy'$

### Exclusive-NOR (XNOR)

$x \odot y = (x \oplus y)' = xy + x'y'$

| x | y | $F = x \odot y$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

### I will become NAND.

inverter (NOT), AND, OR, OR-N

### I will become NOR.

inverter (NOT), AND, OR, OR-N

## Chapter 4

### Design of combinational circuits

① determine required # of inputs & outputs and assign a symbol to each (B, D, H, J) (from circuit specifications)

② derive truth table that defines input/output relationship

③ obtain simplified Boolean functions (K-map shit)

④ draw logic diagram and verify design correctness

### Binary Adder-Subtractor

Half-Adder: $S = xy' + x'y$, $C = xy$ = $S = x \oplus y$, $C = x \cdot y$

Full Adder

Full Adder

| $x_i$ | $y_i$ | $z_i$ | $C_{i+1}$ | $S_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Half adder

| $x_i$ | $y_i$ | $C_{i+1}$ | $S_i$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

### 4-bit full adder

| subscript i: | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|
| input carry | 0 | 1 | 1 | 0 | $C_i$ |
| augend | 1 | 0 | 1 | 1 | $A_i$ |
| addend | 0 | 0 | 1 | 1 | $B_i$ |
| sum | 1 | 1 | 1 | 0 | $S_i$ |
| output carry | 0 | 0 | 1 | 1 | $C_{i+1}$ |

### Decoders

$D_0 = x'y'z'$, $D_1 = x'y'z$, $D_2 = x'yz'$, $D_3 = x'yz$, $D_4 = xy'z'$, $D_5 = xy'z$, $D_6 = xyz'$, $D_7 = xyz$

Any function can be created using a decoder and an OR gate

n input = n-1 selection line variables

- Sometimes, decoders are constructed with NAND gates and enable input E (is place of z)

| E | A | B | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|---|
| 1 | X | X | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |

[2x(3+4+8) = 4+4-16]

3x8 decoder (E) → $D_0$ to $D_7$; 3x8 decoder (E) → $D_8$ to $D_{15}$; W

### Multiplexers

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

4x1 MUX: $S_0$, $S_1$, 0, 1, 2, 3 → F

z, z', 0 or 1

Decoder: input: x y z, output $D_0$ - $D_7$

Encoder: input $D_0$ - $D_7$, output: x y z

## Chapter 5

### SR Latch w/ NOR

| S | R | Q | Q' |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

(after S=1, R=0)

(after S=1, R=0)

(forbidden)

Reset, Set, Q, Q'

### w/ NAND

| Q | Q' | S | R |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

S, R, Q, Q'

### w/ Control input

| En | S | R | Next state of Q |
|---|---|---|---|
| 0 | X | X | No change (inactivity) |
| 1 | 0 | 0 | No change |
| 1 | 0 | 1 | Q=0, reset |
| 1 | 1 | 0 | Q=1, set |
| 1 | 1 | 1 | indeterminate |

En, S, R, Q, Q'

### D-Latch (transparent latch)

| En | D | Q(t+1) |
|---|---|---|
| 0 | X | no change |
| 1 | 0 | Q=0, reset |
| 1 | 1 | Q=1, set |

D, En, Q, Q'

### D flip-flop

$Q(t+1) = D$

| D | Q(t+1) |
|---|---|
| 0 | 0 reset |
| 1 | 1 set |

D, CLK, Q, Q'

### JK flip-flop

$Q(t+1) = JQ' + K'Q$

| J | K | Q(t+1) |
|---|---|---|
| 0 | 0 | Q(t) no change |
| 0 | 1 | 0 reset |
| 1 | 0 | 1 set |
| 1 | 1 | Q'(t) complement |

J, K, Q, Q'

### T flip-flop

$Q(t+1) = T \oplus Q = TQ' + T'Q$

| T | Q(t+1) |
|---|---|
| 0 | Q(t) no change |
| 1 | Q'(t) complement |

T, CLK, Q, Q'

### State diagram

Mealy: 00, in/out, 01

Moore: 00, in, 01 out

### Design Procedure

① make state diagram

② reduce states to minimum

③ assign binary coded state table

④ pick flip-flop

⑤ derive simplified FF input and output eq'n's

### 4-bit JR

JI, D, D, D, D, JO, CLK

### 4-bit serial transfer A to B

A, B, CLK, JC

| timing pulse | JR, A | JR, B |
|---|---|---|
| initial value | 1 0 1 1 | 0 0 1 0 |
| after $T_1$ | 1 1 0 1 | 1 0 0 1 |
| after $T_2$ | 1 1 1 0 | 1 1 0 0 |
| after $T_3$ | 0 1 1 1 | 0 1 1 0 |
| after $T_4$ | 1 0 1 1 | 1 0 1 1 |

## Chapter 6

### Serial Addition

1101 → augend

0111 → addend

1000

shift control, (clk), (serial input), shift reg, augend, addend, FA, clear

| present state | inputs | | next state | output | FF inputs | |
|---|---|---|---|---|---|---|
| Q | x | y | Q | S | $J_Q$ | $K_Q$ |
| 0 | 0 | 0 | 0 | 0 | 0 | X |
| 0 | 0 | 1 | 0 | 1 | 0 | X |
| 0 | 1 | 0 | 0 | 1 | 0 | X |
| 0 | 1 | 1 | 1 | 0 | 1 | X |
| 1 | 0 | 0 | 0 | 1 | X | 1 |
| 1 | 0 | 1 | 1 | 0 | X | 0 |
| 1 | 1 | 0 | 1 | 0 | X | 0 |
| 1 | 1 | 1 | 1 | 1 | X | 0 |

### 4-bit universal shift reg.

MSB, LSB, shift reg, CLK, clear

| mode control | | reg. operation |
|---|---|---|
| $I_1$ | $I_0$ | |
| 0 | 0 | no change (memory) |
| 0 | 1 | shift right |
| 1 | 0 | shift left |
| 1 | 1 | parallel load |