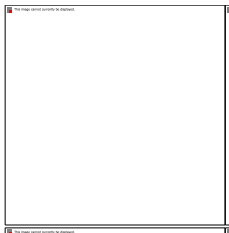
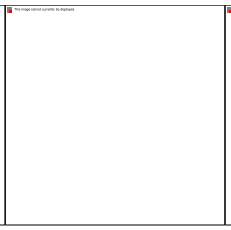
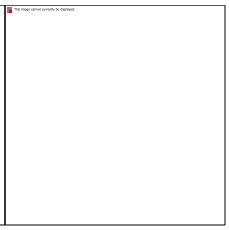
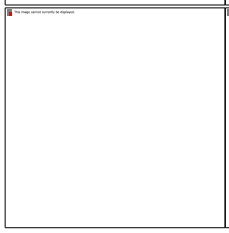
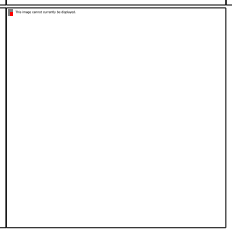

This image cannot currently be displayed.

This image cannot currently be displayed.

This image cannot currently be displayed.

This image cannot currently be displayed.

This image cannot currently be displayed.

This image cannot currently be displayed.

This image cannot currently be displayed.

1.5 VHDL description of digital circuits and systems

VHDL is a hardware description language (HDL)
- used to describe digital circuits and systems.
- if system described in VHDL, you can simulate ~~using~~ operations using VHDL simulator.
  - also possible to synthesize circuit using synthesis program (like compiler) to convert VHDL description to gate level description to be mapped into standard cells or an FPGA

*what do they mean, synthesis prog. or hardware can be used to execute VHDL?

VHDL & Verilog most common HDLs today.

VHDL
- describes behaviour of circuit but not how circuit is implemented.
- strongly typed language
  - for each signal, declaration must be made ~~to~~ of a type indicating representation of the signal.
- includes a set of standard logic types defined by IEEE.
  - to enable use, first line of code must specify design will use VHDL library called IEEE
    - Library includes previously compiled VHDL code organized in set of packages.

`(library ieee;)`

`(use ieee.std_logic_1164.all;)`

- second line must include words to use all design units defined in packages 'std_logic_1164' within IEEE
  - std_logic represents single bit.
  - std_logic_vector represents multibit signal where each bit has type std_logic.

VHDL design entity (module)
- consists of 2 parts
  - entity declaration
    - defines interface between module & outside world
  - architecture body
    - defines internal operation of module.

Entity declaration

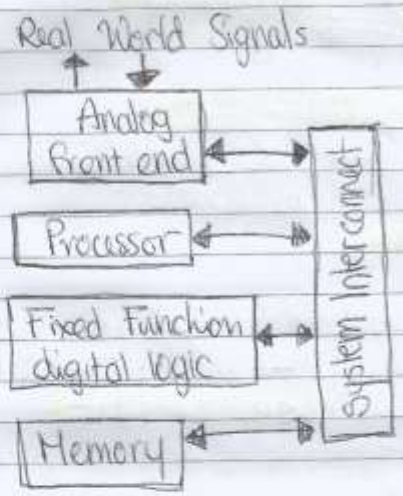
- Begins with keyword "entity" and the name of the module, example - (entity THERMOSTAT is)
  - VHDL is case insensitive
- inputs/outputs are specified in port list,
  - begins with word "port"

Architecture body

- begins with "architecture" followed by "impl" (implementation) and "of" followed by name of entity declaration "Thermostat"
  - VHDL allows for multiple architecture bodies.
    - would have unique identifiers.
  - Concurrent assigment statement
    - <= indicates expression on right is assigned to signal on left.
  - Conditional ~~assignment~~ signal assigment
    - indicated by use of when and else
      - (C equivalent of if else statement)

* In VHDL all of the statements are executed simultaneously, all of the time.

1.6 Digital Logic in Systems



- cell phones, tv, engine controllers in cars use this system.
- represents hardware of system.
- analog builds up noise, so use little as possible.
  ↳ limited to periphery of system & signal conditioning & conversion from analog to digital.

- processors are programmable, built using digital logic, perform ~~syste~~ complex system functions
  ↳ complexity in software on the processor

- Fixed Function logic block, aids processor, performs majority of system's computation.

Chapter 3 Boolean Algebra

3.1 Axioms

→ Boolean algebra derived from definitions of AND, OR, and NOT functions
  → expressed in terms of axioms
    → math statements we assert to be true.

Truth Tables

AND

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

NOT

a	$\bar{a}$
0	1
1	0

→ These truth tables derive from the following axioms:

AND → identity $1 \wedge x = x$ $0 \vee x = x$
OR → annihilation $0 \wedge x = 0$ $1 \vee x = 1$
NOT → negation $\bar{0} = 1$ $\bar{1} = 0$

→ duality of Boolean Algebra
  → if you replace all the $\vee$s with $\wedge$s & vice versa and if you replace 1s with 0s & vice versa then equation is true.

3.2 Properties

commutative	$x \wedge y = y \wedge x$	$x \vee y = y \vee x$
associative	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \vee (y \vee z) = (x \vee y) \vee z$
distributive	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
idempotence	$x \wedge x = x$	$x \vee x = x$
complementation	$x \wedge \bar{x} = 0$	$x \vee \bar{x} = 1$
absorption	$x \wedge (x \vee y) = x$	$x \vee (x \wedge y) = x$
combining	$(x \wedge y) \vee (x \wedge \bar{y}) = x$	$(x \vee y) \wedge (x \vee \bar{y}) = x$
De Morgan	$\overline{(x \wedge y)} = \bar{x} \vee \bar{y}$	$\overline{(x \vee y)} = (\bar{x} \wedge \bar{y})$

* test question *

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

apply idempotence twice to second term.

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

absorption — combine — combine

$= (a \wedge c) \vee (b \wedge c) \vee (a \wedge b)$

* test question * Proof of De Morgan using perfect induction.

x	y	$\overline{(x \wedge y)}$	$(\bar{x} \vee \bar{y})$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

3.3 Dual Functions

The dual of a function, f, is the function f^D
↳ derived from f

You replace: $\wedge$ with $\vee$ and $\vee$ with $\wedge$
1 with 0 and 0 with 1

example:

$$f(x,y) = (1 \wedge x) \vee (0 \vee \bar{y})$$

$$f^D(x,y) = (0 \vee x) \wedge (1 \wedge \bar{y})$$

3.4 Normal Form

To compare two logical expressions put into Normal Form.
↳ (to see if they represent same function).

Normal Form - sum of products term (A.K.A: conjuctive normal form).

example in normal form

$$f(a,b,c) = \underbrace{(\bar{a} \wedge b \wedge c)}_{\text{minterm}} \vee \underbrace{(a \wedge \bar{b} \wedge c)}_{\text{minterm}} \vee \underbrace{(a \wedge b \wedge \bar{c})}_{\text{minterm}} \vee \underbrace{(a \wedge b \wedge c)}_{\text{minterm}}$$

↳ each product term of logic expression in normal form corresponds to one row of truth table for the function.
↳ ANDed terms called minterms
↳ transform any logic expression into normal form by factoring it about each input using:

$$f(x_1, \ldots, x_i, \ldots x_n) = (x_i \wedge f(x_1 \ldots 1 \ldots x_n)) \vee (\bar{x}_i \wedge f(x_1, \ldots, 0 \ldots x_n))$$

Example

Truth table to normal form

	a	b	c	f(a,b,c)
	0	0	0	0
	0	0	1	0
	0	1	0	0
①	0	1	1	1
②	1	0	0	1
③	1	0	1	1
④	1	1	0	1
⑤	1	1	1	1

* when row is true (1), known as implicant. (minterm is implicant only if true row)

Create minterms for all rows that are 'true' (1) (any zeroes are NOTed)

$$f(a,b,c) = (\bar{a} \wedge b \wedge c) \vee (a \wedge \bar{b} \wedge \bar{c}) \vee (a \wedge \bar{b} \wedge c) \vee (a \wedge b \wedge \bar{c}) \vee (a \wedge b \wedge c)$$
$$\qquad\qquad ① \qquad\qquad ② \qquad\qquad ③ \qquad\qquad ④ \qquad\qquad ⑤$$

3.5 From equations to gates

- Logic functions are represented using logic diagrams (AKA schematic drawings)

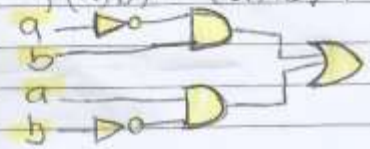
AND →  } can have many inputs.

OR →

Not → } always has single input

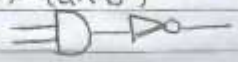
example

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

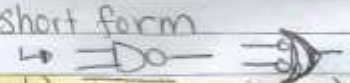


NAND Gate

$f(a,b) = \overline{(a \wedge b)}$



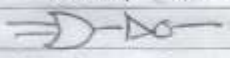
↳ short form



$f(a,b) = \overline{(a \wedge b)} = (\bar{a} \vee \bar{b})$

NOR Gate

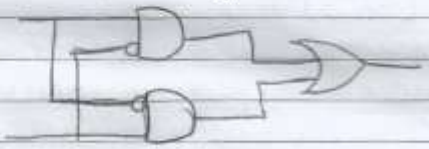
$f(a,b) = \overline{(a \vee b)}$



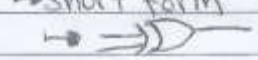
↳ short form

$f(a,b) = \overline{(a \vee b)} = (\bar{a} \wedge \bar{b})$

Exclusive-or gate (XOR)



↳ Short form



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

↳ true only if one is true.

3.6 Boolean expressions in VHDL

example:

```
library ieee;
use ieee.std_logic_1164.all;

entity majority is
   port (a, b, c : in std_logic;
         output : out std_logic);

end majority;

architecture impl of majority is
begin
   output <= (a and b) or (a and c) or (b and c);
end impl;
```

VHDL uses keywords and, or, xor and not.
↳ evaluated from left to right
↳ requires brackets.

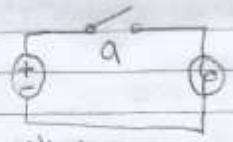
Chapter 4 CMOS

→ Build logic gates using Complementary-metal-oxide Semi-conductor (CMOS) transistors
↳ logic functions can be realized using switches
↳ series combination performs AND
↳ parallel combination performs OR

4.1 Switch Logic

Binary used to represent information
↳ switches controlled by binary (variables) to process information.

example



when a is false (a=0), switch open & light off

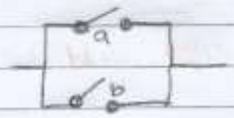
when a is true (a=1) switch closed and light on.

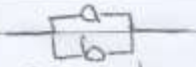
AND

a—b

$f = a \wedge b$

OR





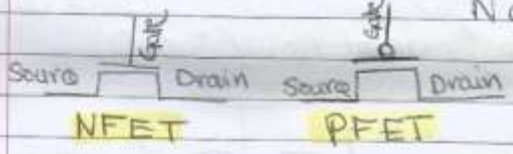
$f = a \vee b$

4.2 Switch model of MOS transistors

- most modern digital systems have CMOS field-effect transistors as switches
  - MOS field effect transistors (MOSFETs)
    (MOS - metal oxide silicon)
    - CMOS is a way of designing chips.

MOSFETs have 3 terminals: Source, gate, drain

2 types of MOSFETS: P channel (PFET)
N channel (NFET)



- MOSFETs are formed on semi-conductive substates
  - 3 terminals: Gate, Source, & Drain.

- The source & drain are identical in structure.
- Gate made of polysilicon.
  - dimensions effect performance
    - Length of gate is d that ~~electrons~~ charge carriers must travel to get from source to drain.
      - ∝ to speed of device.
      - refer to semiconductor process by gate L ex- 28nm CMOS processes.
    - Width determines strength of device, the wider the more ~~electrons~~ charge carriers, lower resistance and over all higher current.
      - Wider makes faster.

n-channel MOSFETs (NFETs)

↳ Source & drain are n-type semiconductor in a p-type substrate and charge carriers are electrons.

p-channel MOSFETs (PFETs)

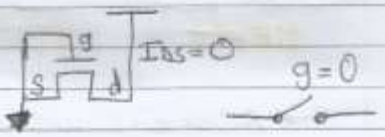
↳ Source & drain are p-type semiconductor in n-type substrate and charge carriers are holes.

NFET

↳ When gate of NFET is logic 0, the source & drain are isolated from each other by a pair of p-n junctions (back to back diodes)

↳ ∴ no current flow ($I_{DS}=0$)

↳ diagram:



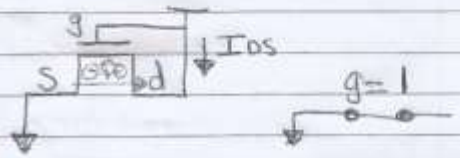
↳ When gate is logic 1, source terminal is 0, NFET is turned on.

↳ +ve voltage between gate & source induce -ve charge in channel beneath gate.

↳ makes channel n-type & form conductive region between source & drain

↳ causes current (I_{DS})

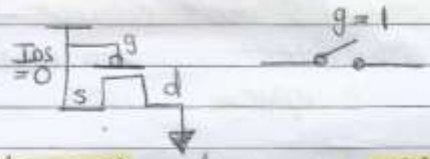
↳ diagram:



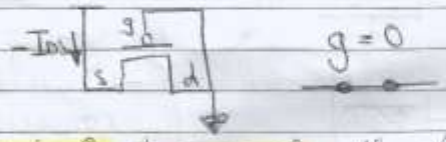
* if source = 1, gate can't be closed

↳ no voltage induced. (Because of this NFET can only pass logic 0)

To pass a logic 1, you need a PFET.

PFET
- ↳ when gate is 0 & source is 1, the device is on
- ↳ diagram: 

- ↳ when gate is 1, device is off.



- ↳ when g is 0 & s is 0, the device is in an undefined state.

Realistically there is delay in devices due to resistance & capacitance.
- ↳ we add resistance in series with drain & source
- ↳ we add capacitance ~~in series~~ from gate to ground.

@@ Capacitance ∝ area of device
∴ $C_g = W \cdot L \cdot K_C$

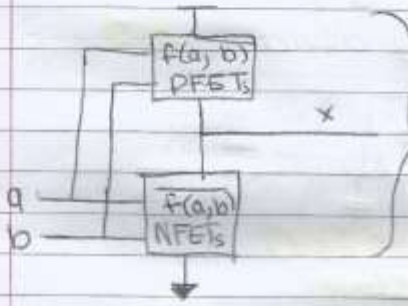
R ∝ ratio of device
∴ $R_s = (L/W) \cdot K_R$

For PFETs & NFETs, K_C is the same. But PFETs have higher resistance (K_R) than NFETs.

4.3 CMOS gate circuits

Basic CMOS gate circuit

Static CMOS gate circuit — realises logic function while restoring output compatible with input.



- when f is true, PFET connects output terminal x to positive supply (V_{DD})
- when f is false, output x connects negative supply by NFET switch network.

- only pass logic 1 (high) through PFET & logic 0 (low) through NFET. (complement each other)

↳ If both are true together, short circuit occurs, draws large amount of current & causes permanent damage.

↳ if path don't cover all (some inputs neither true/false) then output undefined.

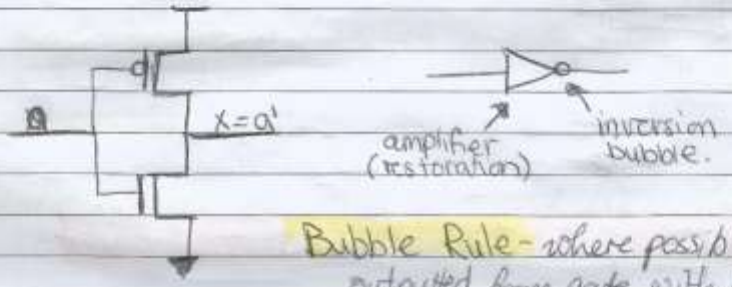
NFETs turn on with high input & generate low output.
PFET turn on with low input & generate high output.

↳ Due to this, we can only generate inverting logic functions with static CMOS gates.

∴ Positive transition (negative) on input of 1 CMOS gate circuit can cause negative transition (positive) on output or no change at all
↳ called monotonic logic functions

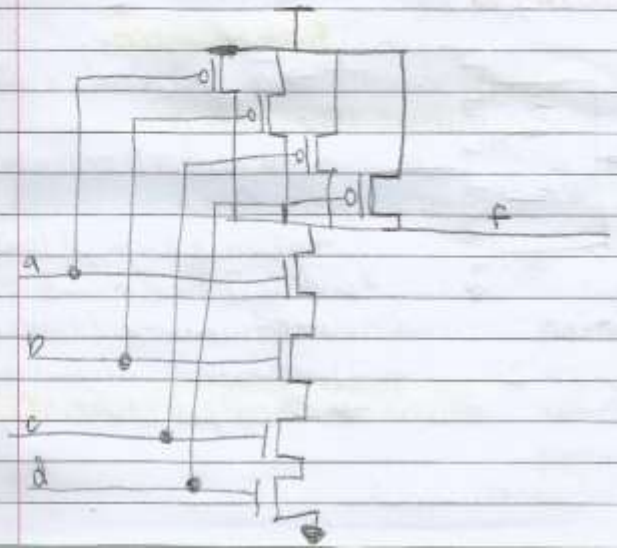
Inverters, NANDs and NORs

Simplest CMOS gate circuit: Inverter.

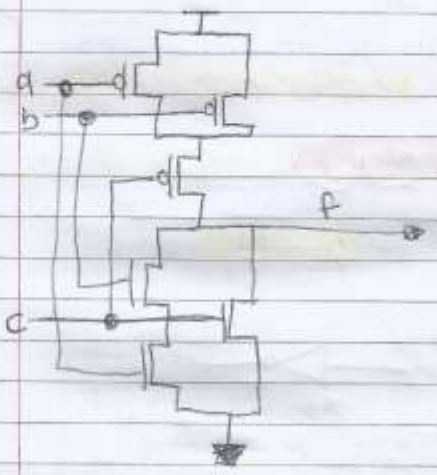


Bubble Rule - where possible, signals outputted from gate with inversion bubble on output shall be input into gate with inversion bubble on input.

Example - draw transister level impl of 4 input Nand gate.
$f = \overline{(a \wedge b \wedge c \wedge d)}$



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



AND OR Invert gate (AOI)

CMOS Gate Circuit / Tri state Circuit

↳ used to build distributed multiplier function where value driven from point A to signal node if logic signal a is true, and from point B when logic signal b is true.

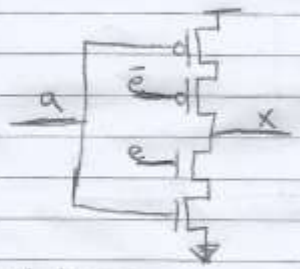
↳ tri state inverter.

↳ example

e	a	x
0	0	z
0	1	z
1	0	1
1	1	0

simplified: when e = 1, acts like normal inverter. when e = 0, output disconnected (z)

e means enable.



* note: not a gate.

tristate inverter.
- → $\bar{a}$ driven onto x when e is true.
- → neither 0 or 1 when e is false.
- → when e high ($\bar{e}$ low), middle 2 transistors on, outer transistors (controlled by a) function as normal inverters.
- → 3 output states: 0, 1, Z.

* ~~Where~~ wire not being driven denoted with 'z'

Chapter 6 Combinational logic design

6.1 Combinational Logic

A combinational logic circuit generates a set of outputs whose state depends only on current state of inputs.
↳ When input changes, time required for output to change.
↳ outputs don't reflect history of inputs.

A majority circuit (accepts n ~~outp~~ inputs and outputs 1 if more than half of inputs are 1) is a combinational logic circuits.

Static nature of combinational logic (CL) circuits makes them easy to design and analyze.

6.2 Closure

CL circuits are closed under acyclic composition.
↳ connect together a number of CL circuits (output of one to input of another) and avoid creating loops (cyclic), result would also be a CL circuit.
∴ We can create large CL circuits by connecting smaller CL circuits

6.3 Truth tables, minterms & normal form

If test question

example. build CL circuit that outputs 1 when 4 bit input represents prime number.

Start with truth table. for an n-bit input function, truth table has 2^n rows
4 bit has 2^4 row = 16 row.

Truth table of Prime numbers

Number	In	Out
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

Reduced truth table

Number	In	Out
1	0001	1
2	0010	1
3	0011	1
5	0101	1
7	0111	1
11	1011	1
13	1101	1

create function.

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

minterm: an input in each row
↳ derived from minimal number of input states.

function can be written as

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

normal form.

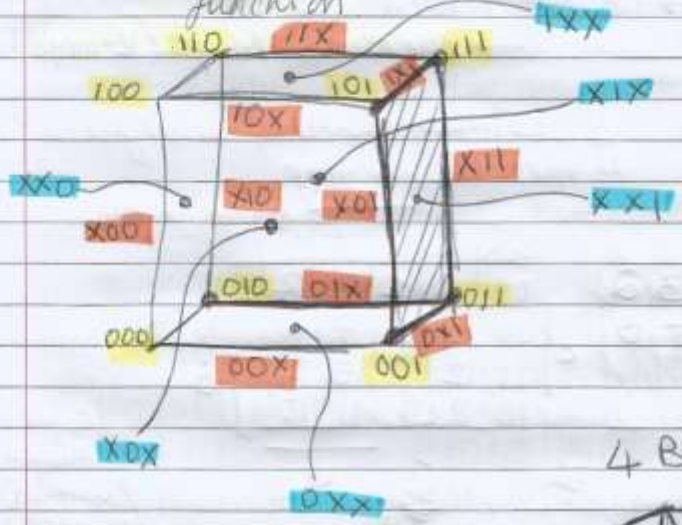
6.4 Implicant & Cubes

When examining a reduced truth table, it is apparent that most inputs differ in only one position.
↳ ∴ if we allow bits of input to be replaced with X, we can replace 2 rows with one.
↳ Example: 0010 and 0011 replaced with 001X

only possible with implicants?

$f_{001X} = (\bar{d} \wedge \bar{c} \wedge b \wedge \bar{a}) \vee (\bar{d} \wedge \bar{c} \wedge b \wedge a) \rightarrow = (\bar{d} \wedge \bar{c} \wedge b)$

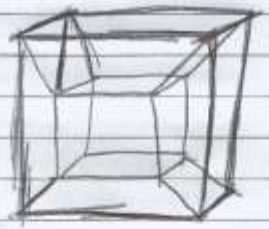
Cube visualization of 3 bit prime number function



Key
- Vertices
- Line general
- Plane general

Darkened verticies/lines are implicants.

4 Bit cube.



Join 3 bit to a bigger cube.

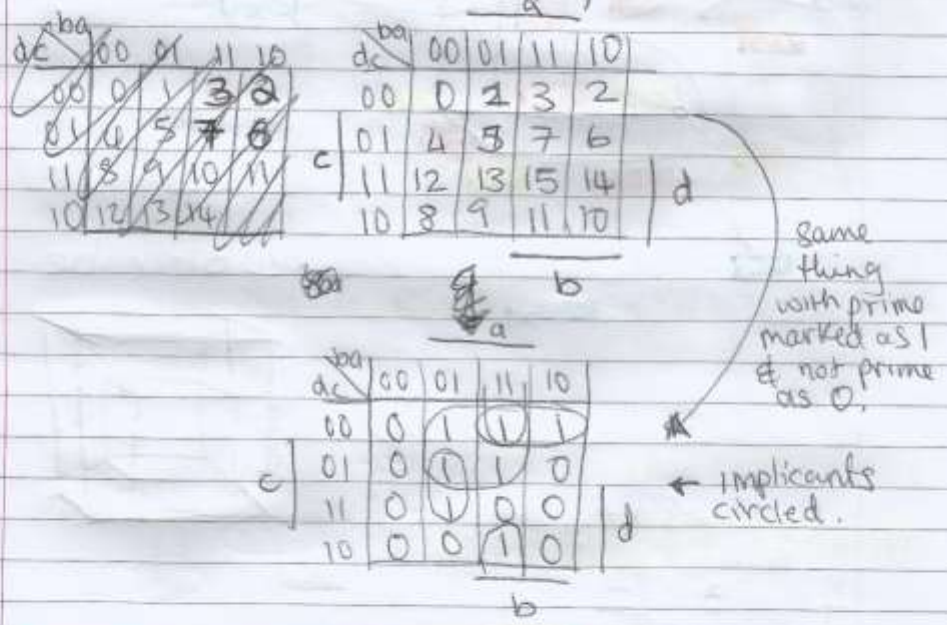
Implicants of 4 bit prime f.

Number of Variables			
4	3	2	1
0001	001X	0XX1	–
0010	00X1	–	–
0011	0X01	–	–
0101	0X11	–	–
0111	01X1	–	–
1011	X011	–	–
1101	X101	–	–

6.5 Karnaugh Maps

- inconvienient to draw cubes all the time.
  - harder as dimensions increase.
  - we replace with 2D cubes
    - Karnaugh Maps (K-maps).

example 4 variable minterms arranged in 4 variable K-map.



6.6 Covering a function.

Once we have list of implicants
- least expensive must be chosen that cover the function.
- Cost defined by # variables
  - 0011 (4 variables ∴ Cost 4)
  - 0X11 (3 variables ∴ Cost 3)
  - etc.

Procedure to select inexpensive implicant set:

1. Start with empty cover.
2. Add all essential prime implicants to cover.
3. For each remaining uncovered ~~implicant~~ minterm, add largest implicant that covers that minterm.

→ will alway give us good cover, not always least expensive.

Example. Derive minimal cover of 3 variable function:

a

$f = \sum_{1c} m(1, 3, 4, 5)$

c \ ba	00	01	11	10
0	0	1	3	2
1	4	5	7	6

b

c \ ba	00	01	11	10
0	0	1	1	0
1	1	1	0	0

0X1

10X

X01

only highlighted are essential.

$\therefore f = 10X \vee 0X1$

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

6.9 Product-of-sums implementation.

For sum of products we have looked at ~~the~~ input states where truth table is 1.

For product of sums we look at input states where truth table is 0.

Maxterm is a sum (OR) that includes every variable or its complement. Each 0 in truth table / K map ~~responds~~ corresponds to a maxterm.

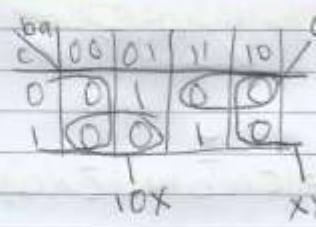
Once you have mastered sum-of-products design, the easiest way to generate product of sums logic circuit is to find the sum of products for complement of the logic function
↳ function that results by swapping 1 and 0 leaving f_x unchanged

↳ apply De Morgans to output (AND → OR)

example Product of Sums.

express 3 input $f = \Sigma_n m(1,7)$ as minimal product of sums.

① Draw K-map



② find implicants of the complement of this function (all zeros)

$f' = \Sigma_n m(0,2,3,4,5,6)$

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

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