Digital Logic
Appendix A

Boolean Algebra

Gates

Combinatorial Circuits

Sequential Circuits

Boolean Algebra

- George Boole
  - ideas 1854
- Claude Shannon,
  - apply to circuit design, 1938 (piirisuunnittelu)
- Describe digital circuitry function
  - programming language?
- Optimise given circuitry
  - use algebra (Boolean algebra) to manipulate (Boolean) expressions into simpler expressions

Boolean Algebra

(tulo,
yhteenlasku
negaatio)

- Variables: A, B, C
- Values: TRUE (1), FALSE (0)
- Basic logical operations:
  - binary: AND (●), OR (+)
  - unary: NOT (‾)
- Composite operations, equations
  - precedence: NOT, AND, OR
  - parenthesis

(ja,
tai,
ei)

$A \bullet B = AB$ $B + C$

$\overline{A}$

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

Boolean Algebra

- Other operations
  - NAND
  - NOR
- Truth tables
  - What is the result of the operation?

$A \text{ NAND } B = \text{NOT}\,(A \text{ AND } B) = \overline{AB}$

$A \text{ NOR } B = \text{NOR}\,(A \text{ OR } B) = \overline{A+B}$

Q

AND	0	1
0	0	0
1	0	1

P

Table A.1

P	Q	P AND Q
0	0	0
0	1	0
1	0	0
1	1	1

Postulates, Identities in Boolean Algebra

- How can I manipulate expressions?
  - Simple set of rules?
- Basic identities

Table A.2

  - commutative laws (vaihdantalait)
  - distributive laws (osittelulait)
  - identity elements (identiteetit)
  - inverse elements (vasta-alkiot)
  - associative laws (liitäntälait)
  - DeMorgan’s theorem (DeMorganin laki)

Gates

(portit)

- Fundamental building blocks
  - easy to build

http://tech-www.informatik.uni-hamburg.de/applets/cmos/cmosdemo.html

  - implement basic Boolean algebra operations
- Combine to build more complex circuits
  - memory, adder, multiplier

(yhteenlaskupiiri, kertolaskupiiri)

- 1-3 inputs, 1 output

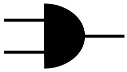


Fig. A.1

AND

AND, OR, NOT, NAND, NOR

- Gate delay

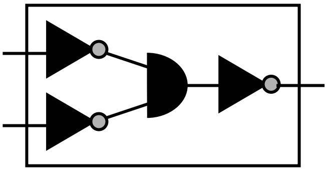
(viive)

  - change inputs, after gate delay new output available

1 ns? 10 ns? 0.1 ns?

Functionally Complete Set

- Can build all basic gates (“and”, “or”, “not”) from a smaller set of gates
  - – With “and”, “or”, “not” (trivial!)
  - – With “and”, “not”
    - “or”? $A+B=\overline{\overline{A}\bullet\overline{B}}$



OR

  - – With “or”, “not”
  - – With “nand” alone Fig A.2
  - – With “nor” Fig A.3

Combinational Circuits (3)

(yhdistelmä- piirit)

- Interconnected set of gates
  - – change input, wait for gate delays, new output
- Output is Boolean function of input signals
  - – m (binary, Boolean) inputs
  - – n (binary, Boolean) outputs
- Described in three ways
  - – describe <u>function</u> with *Boolean equations* (<u>one for each output</u>) $F=\overline{A}B\overline{C}+\overline{A}BC+AB\overline{C}$
  - – describe <u>function</u> with *truth table* Table A.3
  - – describe <u>implementation</u> with graphical symbols for gates and wires Fig A.4

Simplify Presentation (and Implementation) (3)

- Boolean equations
  - Sum of products form (SOP)

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

  - Product of sums form (POS)

$$F = (A+B+C)\bullet(A+B+\overline{C})\bullet(\overline{A}+B+C)\bullet(\overline{A}+B+\overline{C})\bullet(\overline{A}+\overline{B}+\overline{C})$$

Fig A.4

Boolean algebra

Fig A.5

Which presentation is better?

Fewer gates? Smaller area on chip?

Smaller circuit delay? Faster?

Algebraic Simplification

- Circuits become too large to handle?
- Use basic identities to simplify Boolean expressions

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$
$$= \overline{A}B + B\overline{C} = B(\overline{A}+\overline{C})$$

Fig A.4

Fig A.6

- May be difficult to do
- How to do it automatically?
- Build a program to do it “best”?

$$f = \overline{a}b\overline{c}d + \overline{a}bcd + ab\overline{c}\overline{d} + ab\overline{c}d$$
$$+ abcd + abc\overline{d} + a\overline{b}cd + a\overline{b}c\overline{d}$$

Karnaugh Map Squares

- Each square represents complete input value combination
  - canonical form: each term has each variable once
  - adjacent squares differ only in <u>one</u> input value (wrap around)

order!!

CD: <u>AB</u>	**00**	**01**	**11**	**10**
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Square for <u>input value combination</u>

$\boxed{A\overline{B}\overline{C}D = 1001}$

Karnaugh Maps

- Represent Boolean function (I.e., circuit) <u>truth table</u> in another way
  - each square is **one product** in sums-of-products (SOP) presentation
  - value is **one** (1) iff corresponding input values give value 1, o/w value is "**empty**"
  - I.e., value is 1 iff function value for those input values is one

Fig A.7

CD

AB

	00	**01**	**11**	**10**
00			1	
01				
11	1			
10		1		

$F = \overline{A}\overline{B}CD + AB\overline{C}\overline{D} + A\overline{B}\overline{C}D$

Karnaugh Map Simplification (3)

- Starting point:
  - Adjacent squares differ only in one input variable value

1	1
C	$\bar{C}$

Adjacent squares have value 1

↓

Input values differ only in one variable

↓

Value of <u>that variable</u> is irrelevant
(when all other input variables are fixed
to corresponding values for those squares)

↓

Can ignore that variable for those expressions

Using Karnaugh Maps to Minimize Boolean Functions (8)

Original function

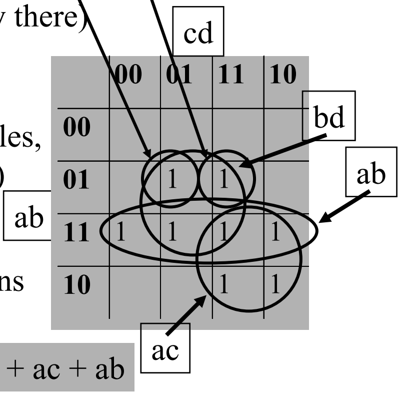
$$f = \bar{a}b\bar{c}d + \bar{a}bcd + ab\bar{c}\bar{d} + ab\bar{c}d + abcd + abc\bar{d} + a\bar{b}cd + a\bar{b}c\bar{d}$$

Canonical form (now already there)

Karnaugh Map

Find smallest number of circles, each with largest number (2^i) of 1's

Select parameter combinations corresponding to the circles

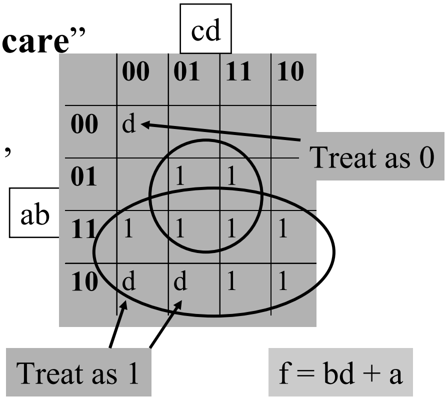


ab \ cd	00	01	11	10
00				
01		1	1	
11	1	1	1	1
10			1	1

Get reduced function f = bd + ac + ab

Impossible Input Variable Combinations

- What if some input combinations can never occur?
  - Mark them ”**don’t care**” or “d”
  - treat them as 0 or 1, whichever is best
  - more room to optimize



cd

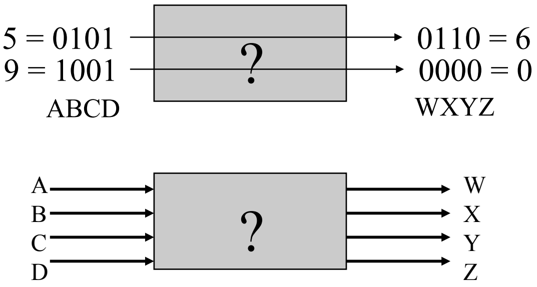
ab	00	01	11	10
00	d			
01		1	1	
11	1	1	1	1
10	d	d	1	1

Treat as 0

Treat as 1

f = bd + a

Example: Circuit to add 1 (mod 10) to 4-bit BCD decimal number (4)



Truth table? Table A.4

Karnaugh maps for W, X, Y and Z? Fig. A.10

Quine-McKluskey Method

- Another method to minimise Boolean expressions
- Why?
  - – Karnaugh maps become complex with 6 input variables
- Quine-McKluskey method
  - – tabular method
  - – automatically suitable for programming
  - – details skipped

Luque Method

- Another method to minimise Boolean expressions
- Based on dividing circle in different ways
- Can be fractally expanded to infinitely many variables
- Interesting, but not part of this course
- Details skipped

See http://www.cs.helsinki.fi/Teemu.Kerola/tikra/artikkelit/d_luque_1.pdf
(September 2003)

Basic Combinational Circuits

- Building blocks for more complex circuits
  - Multiplexer
  - Encoders/decoder
  - Read-Only-Memory
  - Adder

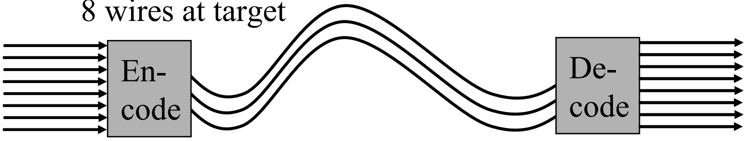
Multiplexers (2)

- Select one of many possible inputs to output
  - black box Fig A.12
  - simple truth table Tbl A.7
  - implementation Fig A.13
- Each input/output "line" can be many parallel lines
  - select one of three 16 bit values Fig A.14
    - $C_{0..15}$, $IR_{0..15}$, $ALU_{0..15}$
  - simple extension to one line selection
    - lots of wires, plenty of gates …

Encoders/Decoders

- Only one of many Encoder input or Decoder output lines can be 1
- Encode that line number as output
  - hopefully less pins (wires) needed this way
  - optimise for space, not for time
  - Example:
    - encode 8 input wires with 3 output pins
    - route 3 wires around the board
    - decode 3 wires back to 8 wires at target

Fig A.15



Read-Only-Memory (ROM) (4)

- Given input values, get output value
  - Like multiplexer, but with fixed data
- Consider input as address, output as contents of memory location
- Truth tables for a ROM
  - 64 bit ROM
  - 16 words, each 4 bits wide

Table A.8

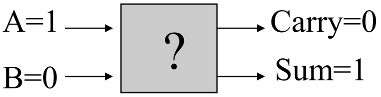
Mem (7) = 4

Mem (11) = 14

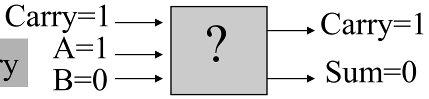
- Implementation with decoder & or gates

Adders (4)

1-bit adder



1-bit adder with carry



Implementation

Table A.9, Fig A.22

Build a 4-bit adder from four 1-bit adders

Fig A.21

Sequential Circuit

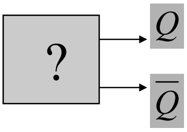
(sarjalliset piirit)

- Circuit has (modifiable) internal state
- Output of circuit depends (also) on internal state
  - not only from current inputs
  - output = f_o (input, state)
  - new state = f_s (input, state)
- Processor control, registers, memory

Flip-Flop

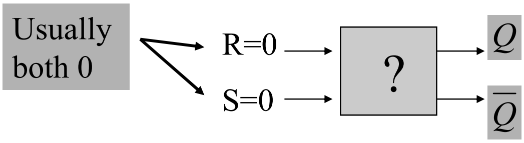
(kiikku)

- 2 states for Q (0 or 1, true or false)
- 2 outputs $Q \text{ and } \overline{Q}$
  - complement values
  - both always available on different pins
- Need to be able to change the state (Q)

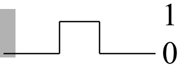


S-R Flip-Flop or S-R Latch (3)

(laituri)

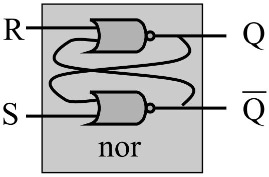

Usually both 0

"Write 1" = "set S=1 for a short time" = "set"



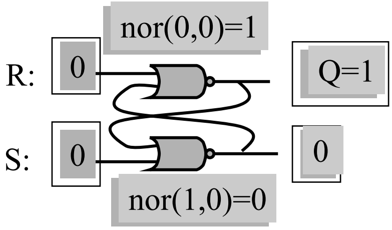
"Write 0" = "set R=1 for a short time" = "reset"

nor (0, 0) = 1
nor (0, 1) = 0
nor (1, 0) = 0
nor (1, 1) = 0



S-R Latch Stable States (3)

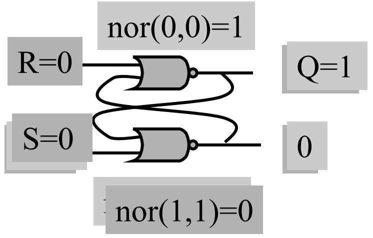
- 1 bit memory (value = value of Q)
- bi-stable, when R=S=0
  - Q=0?
  - Q=1?



nor (0, 0) = 1
nor (0, 1) = 0
nor (1, 0) = 0
nor (1, 1) = 0

S-R Latch Set (1) and Reset (0) (n)

Write 1: S=0 → 1 → 0



R=0

S=0

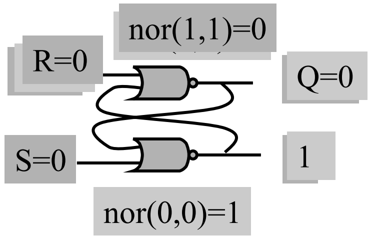
nor(0,0)=1

nor(1,1)=0

Q=1

0

Write 0: R=0 → 1 → 0



nor(1,1)=0

R=0

S=0

Q=0

1

nor(0,0)=1

nor (0, 0) = 1
nor (0, 1) = 0
nor (1, 0) = 0
nor (1, 1) = 0

Clocked Flip-Flops

- State change can happen only when clock is 1
  - more control on state changes
- Clocked S-R Flip-Flop Fig. A.26
- D Flip-Flop Fig. A.27
  - only one input D
    - D = 1 and CLOCK → write 1
    - D = 0 and CLOCK → write 0
- J-K Flip-Flop Fig. A.28
  - Toggle Q when J=K=1 Fig. A.29

Registers (2)

- Parallel registers
  - – read/write
  - – CPU user registers
  - – additional internal registers
- Shift Registers
  - – shifts data 1 bit to the right
  - – serial to parallel?
  - – ALU operation?
  - – rotate?

Fig. A.30

Fig. A.31

Counters (4)

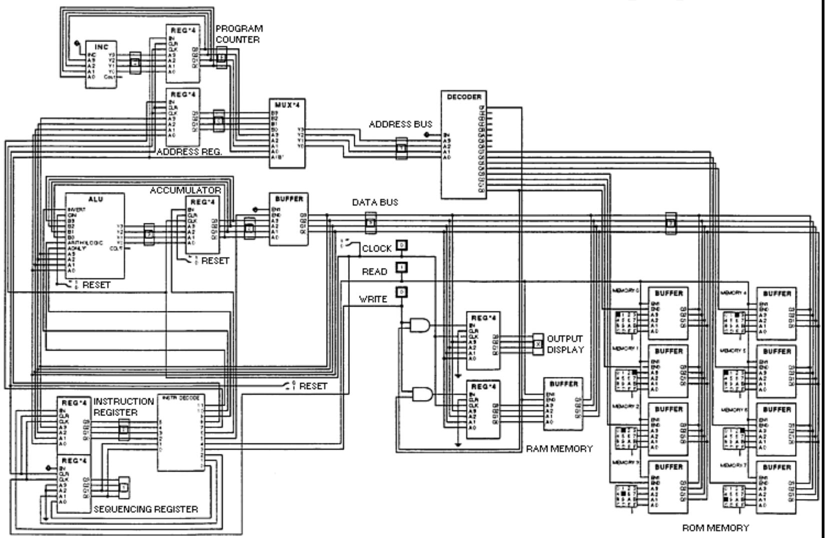
- Add 1 to stored counter value
- Counter
  - – parallel register plus increment circuits
- Ripple counter
  - – asynchronous
  - – increment least significant bit, and handle “carry” bit as far as needed
- Synchronous counter
  - – modify all counter flip-flops simultaneously
  - – faster, more complex, more expensive

Summary

- Boolean Algebra → Gates → Circuits
  - can implement all with NANDs or NORs
  - simplify circuits: Karnaugh,
    Quine-McKluskey, Luque, …
- Components for CPU design
  - ROM, adder
  - multiplexer, encoder/decoder
  - flip-flop, register, shift register, counter

-- End of Appendix A: Digital Logic --

Simple processor



http://www.gamezero.com/team-0/articles/math_magic/micro/stage4.html