

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 (\bullet), OR ($+$) $A \bullet B = AB$ $B + C$
 - unary: NOT ($\bar{\quad}$) \bar{A}
- Composite operations, equations
 - precedence: NOT, AND, OR
 - parenthesis $D = A + \bar{B} \bullet C = A + (\bar{B}C)$

(ja, tai, ei)

Boolean Algebra

- Other operations
 - NAND $A \text{ NAND } B = \text{NOT}(A \text{ AND } B) = \overline{AB}$
 - NOR $A \text{ NOR } B = \text{NOT}(A \text{ OR } B) = \overline{A + B}$
- Truth tables
 - What is the result of the operation?

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

Table A.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 (liitantalait)
 - DeMorgan's theorem (DeMorganin laki)

Gates

(portit)


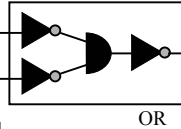
- Fundamental building blocks
 - easy to build <http://tech-www.informatik.uni-hamburg.de/applets/cmos/cmosdemo.html>
 - implement basic Boolean algebra operations
- Combine to build more complex circuits
 - memory, adder, multiplier (yhteenlaskupiiri, kertolaskupiiri)
- 1-3 inputs, 1 output 
 - AND, OR, NOT, NAND, NOR
- Gate delay (viive)
 - change inputs, after gate delay new output available
 - 1 ns? 10 ns? 0.1 ns?

Fig. A.1

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”? $A + B = \overline{\overline{A} \cdot \overline{B}}$
 - With “or”, “not”
 - With “nand” alone Fig A.2
 - With “nor” Fig A.3



OR

Combinational Circuits (3)

(yhdistelmäpiirit)

- 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 function with *Boolean equations* (one for each output) $F = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C$
 - describe function with *truth table* Table A.3
 - describe implementation with graphical symbols for gates and wires Fig A.4

Simplify Presentation (and Implementation) (3)

- Boolean equations
 - Sum of products form (SOP) $F = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C$ Fig A.4
 - Product of sums form (POS) $F = (A + B + C) \cdot (A + B + \overline{C}) \cdot (\overline{A} + B + C) \cdot (\overline{A} + B + \overline{C}) \cdot (\overline{A} + B + \overline{C})$ Fig A.5

Boolean algebra

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
 - $F = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C$ Fig A.4
 - $= \overline{A}B + \overline{A}C = B(\overline{A} + C)$ Fig A.6
- May be difficult to do
- How to do it automatically?
- Build a program to do it “best”?

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

Karnaugh Map Squares

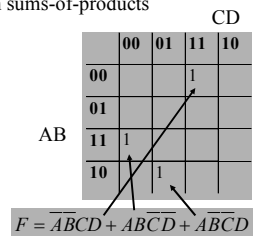
- Each square represents complete input value combination
 - canonical form: each term has each variable once
 - adjacent squares differ only in one input value (wrap around)

CD\AB	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 input value combination
 $ABCD = 1001$

Karnaugh Maps

- Represent Boolean function (I.e., circuit) truth table in another way Fig A.7
 - 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



Karnaugh Map Simplification (3)

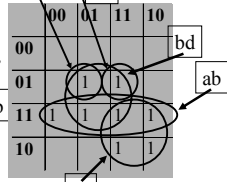
- Starting point:
 - Adjacent squares differ only in one input variable value
- 
- Adjacent squares have value 1
- Input values differ only in one variable
- Value of that variable is irrelevant (when all other input variables are fixed to corresponding values for those squares)
- Can ignore that variable for those expressions

22.9.2003

Copyright Teemu Kerola 2003

13

Using Karnaugh Maps to Minimize Boolean Functions (8)

- Original function $f = \overline{a}bcd + a\overline{b}cd + ab\overline{c}d + abc\overline{d} + \overline{a}bc\overline{d} + \overline{a}b\overline{c}d + \overline{a}b\overline{c}\overline{d} + \overline{a}b\overline{c}d$
- 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
- Get reduced function $f = bd + ac + ab$

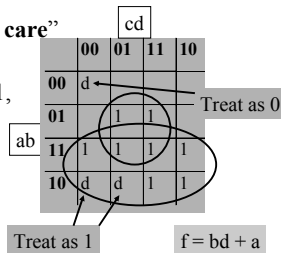
22.9.2003

Copyright Teemu Kerola 2003

14

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

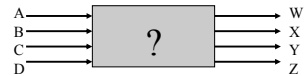
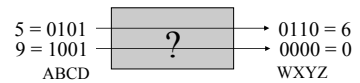


22.9.2003

Copyright Teemu Kerola 2003

15

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

22.9.2003

Copyright Teemu Kerola 2003

16

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

22.9.2003

Copyright Teemu Kerola 2003

17

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)

22.9.2003

Copyright Teemu Kerola 2003

18

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 Fig A.15
 - decode 3 wires back to 8 wires at target

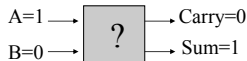


Read-Only-Memory (ROM) (4)

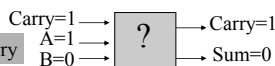
- Given input values, get output value
 - Like multiplexer, but with fixed data
 - Consider input as address, output as contents of memory location
 - Truth tables for a ROM Table A.8
 - 64 bit ROM
 - 16 words, each 4 bits wide
- Mem (7) = 4 Mem (11) = 14
- Implementation with decoder & or gates

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

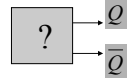
(sarjalliset piirit)

- 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

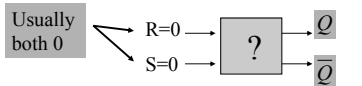
(kiikku)

- 2 states for Q (0 or 1, true or false)
- 2 outputs Q and \bar{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⁾

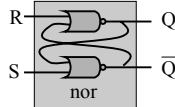
(laituri)



“Write 1” = “set S=1 for a short time” = “set”

“Write 0” = “set R=1 for a short time” = “reset”

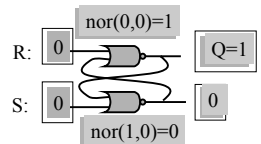
nor (0, 0) = 1
 nor (0, 1) = 0
 nor (1, 0) = 0
 nor (1, 1) = 0



S-R Latch Stable States ⁽³⁾

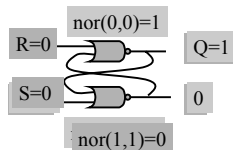
- 1 bit memory (value = value of Q)
- bi-stable, when R=S=0
 - Q=0?
 - Q=1?

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

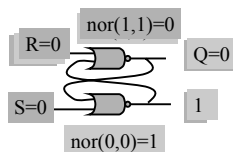


S-R Latch Set (1) and Reset (0) ⁽ⁿ⁾

Write 1: S=0 → 1 → 0



Write 0: R=0 → 1 → 0



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

22.9.2003

Copyright Teemu Kerola 2003

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

22.9.2003

Copyright Teemu Kerola 2003

32

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

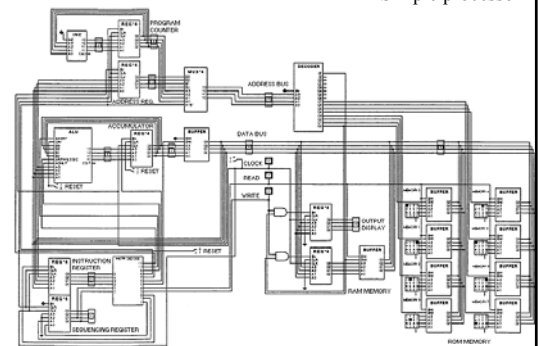
22.9.2003

Copyright Teemu Kerola 2003

33

-- End of Appendix A: Digital Logic --

Simple processor



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

22.9.2003

Copyright Teemu Kerola 2003

34