

## Digital Logic Appendix A

Boolean Algebra  
Gates  
Combinatorial Circuits  
Sequential Circuits

1.11.1999 Copyright Teemu Kerola 1999 1

## 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

1.11.1999 Copyright Teemu Kerola 1999 2

## Boolean Algebra

- Variables: A, B, C
- Values: TRUE (1), FALSE (0)
- Basic logical operations:
  - binary: AND (•), OR (+)  $A \bullet B = AB$   $B + C$
  - unary: NOT ( )  $\bar{A}$  (ja, tai, ei)
- Composite operations, equations
  - precedence: NOT, AND, OR
  - parenthesis  $D = A + \bar{B} \bullet C = A + ((\bar{B})C)$

1.11.1999 Copyright Teemu Kerola 1999 3

## 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?

Table A.1

		Q	
	AND	0	1
P	0	0	0
	1	0	1

1.11.1999 Copyright Teemu Kerola 1999 4

## 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 (liitántälait)
  - DeMorgan's theorem (DeMorganin laki)

1.11.1999 Copyright Teemu Kerola 1999 5

## Gates (portit)

- Fundamental building blocks
  - easy to build
  - 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
- Gate delay (viive)
  - change inputs, new output available after gate delay

1.11.1999 Copyright Teemu Kerola 1999 6

### Functionally Complete Set

- Can build all basic gates (and, or, not) from a smaller set of gates
  - and, or, not (trivial!)
  - and, not
    - OR?  $A + B = \overline{\overline{A} \cdot \overline{B}}$
  - or, not
  - nand
  - nor

Fig A.2  
Fig A.3

1.11.1999 Copyright Teemu Kerola 1999 7

### Combinational Circuits (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
  - Boolean equations (one for each output)
    - $F = \overline{A}BC + A\overline{B}C + ABC$
  - truth table
  - graphical symbols describe implementation with gates and wires

Table A.3  
Fig A.4

1.11.1999 Copyright Teemu Kerola 1999 8

### Simplify Presentation (and Implementation) (3)

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

Boolean algebra

Which presentation is better?  
Fewer gates? Smaller area on chip?  
Smaller circuit delay? Faster?

Fig A.4  
Fig A.5

1.11.1999 Copyright Teemu Kerola 1999 9

### Algebraic Simplification

- Circuits become too large to handle?
- Use basic identities to simplify Boolean expressions
  - $F = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C$
  - $= \overline{A}B + B\overline{C} = B(\overline{A} + \overline{C})$

Fig A.5  
Fig A.6

1.11.1999 Copyright Teemu Kerola 1999 10

### Karnaugh Map Squares

- Each square represents complete input value combination (canonical form)
  - adjacent squares differ only in one input value (wrap around)

CD:	00	01	11	10	
AB:	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  $ABCD = 1001$

order!!

1.11.1999 Copyright Teemu Kerola 1999 11

### Karnaugh Maps


- Represent Boolean function (I.e., circuit) truth table 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
  - value is function value for those input values

Fig A.7

1.11.1999 Copyright Teemu Kerola 1999 12

### Karnaugh Map Simplification

- 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

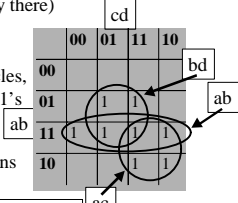
1.11.1999 Copyright Teemu Kerola 1999 13

### Using Karnaugh Maps to Minimize Boolean Functions <sup>(6)</sup>

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

Canonical form (now already there)

Karnaugh Map



Find smallest number of circles, each with largest number of 1's

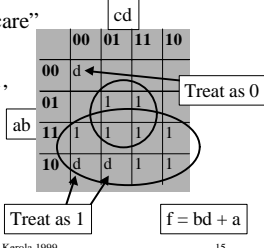
Select parameter combinations corresponding to the circles

Get reduced function  $f = bd + ac + ab$

1.11.1999 Copyright Teemu Kerola 1999 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

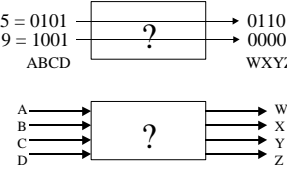


1.11.1999 Copyright Teemu Kerola 1999 15

### Example: Circuit to add 1 (mod 10) to 4-bit BCD decimal number <sup>(4)</sup>

5 = 0101  
9 = 1001  
ABCD

0110 = 6  
0000 = 0  
WXYZ



Truth table? Table A.4

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

1.11.1999 Copyright Teemu Kerola 1999 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

1.11.1999 Copyright Teemu Kerola 1999 17

### Multiplexers

- 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 "line" can be many parallel lines
  - select one of three 15 bit values Fig A.14
  - simple extension

1.11.1999 Copyright Teemu Kerola 1999 18

### Encoders/Decoders

- Only one of many input lines can be 1
- Encode the line number as output
  - hopefully less pins needed this way
  - 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

1.11.1999 Copyright Teemu Kerola 1999 19

### Read-Only-Memory (ROM) <sup>(4)</sup>

- Given input values, get output value
- 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

1.11.1999 Copyright Teemu Kerola 1999 20

### Adders <sup>(4)</sup>

1-bit adder

A=1 →

B=0 →

?

→ Carry=0

→ Sum=1

1-bit adder with carry

Carry=1 →

A=1 →

B=0 →

?

→ Carry=1

→ Sum=0

Implementation Table A.9, Fig A.22

Build 4-bit adder from 4 1-bit adders Fig A.21

1.11.1999 Copyright Teemu Kerola 1999 21

### Sequential Circuit

(sarjalliset piirit)

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

1.11.1999 Copyright Teemu Kerola 1999 22

### 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)

1.11.1999 Copyright Teemu Kerola 1999 23

### S-R Flip-Flop or S-R Latch <sup>(4)</sup>

(laituri)

Usually both 0

R=0 →

S=0 →

?

→ Q

→  $\bar{Q}$

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

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

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

1.11.1999 Copyright Teemu Kerola 1999 24

### S-R Latch Stable States <sup>(3)</sup>

- 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

1.11.1999 Copyright Teemu Kerola 1999 25

### S-R Latch Set (1) and Reset (0) <sup>(n)</sup>

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

1.11.1999 Copyright Teemu Kerola 1999 26

### 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

1.11.1999 Copyright Teemu Kerola 1999 27

### Registers <sup>(2)</sup>

- Parallel registers Fig. A.30
  - read/write
  - CPU user registers
  - additional internal registers
- Shift Registers Fig. A.31
  - shifts data 1 bit to the right
  - serial to parallel?
  - ALU operation?
  - rotate?

1.11.1999 Copyright Teemu Kerola 1999 28

### Counters <sup>(4)</sup>

- 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

1.11.1999 Copyright Teemu Kerola 1999 29

### Summary

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

1.11.1999 Copyright Teemu Kerola 1999 30

