Computer Systems Overview
Digital Logic Combination Circuits
CO-I Ch 1-8 [Sta10]
Some material from
Comp. Org I
Digital Logic, Ch 20.1-3
John von Neumann
and EDVAC, 1949

Overview Content
- Structure
- OS view point
- Buses
- I/O-controller and memory-mapped I/O
- Memory hierarchy

Goal:
- Remember what has already been covered on Comp. Org I

Structure of a computer
Hardware vs Software
Control, Processing, Storage, Data movement

Operating System’s view point
Applications
Shell
System programs

Resource management
File system
Memory management
I/O management
Interrupt handling

I/O module (Device controller)
Devices and device drivers

Buses
- Local (Sisäinen), System, I/O expansion
- Device controllers (Laiteohjaimet), NOTE: Sta10: “I/O module”

I/O controller and memory-mapped I/O
- Device driver (ajuri) controls the device via controller’s registers
- Driver refers to these registers as regular memory locations
- Common memory references, like in load/store-instructions
- Controller (ohjain) detects its own memory addresses on the bus
- Device controller – “intelligent” memory location
Memory hierarchy

- Access time (saantiaika) (not?) dependent of the location
- Registers, cache, main memory
- Block buffering (lohkopuskurointi) (OS functionality)
- Magnetic and optical storage devices
- File servers (tiedostopalvelu)
- Network Attached Storage (NAS) - files
- Storage Area Network (SAN) - blocks

Which new common technology is missing from picture?

CPU execution modes

- Instruction privileges
  - Privileged (etuoikeutettu) and normal
  - Privileged, kernel
  - User, normal

- Memory protection
  - Memory area marked for a user and controlled access

- User mode (käyttäjätila)
  - May use only normal instructions
  - Can refer only to its own memory area

- Kernel mode (etuoikeutettu tila)
  - Can use all instructions, including the privileged ones
  - May refer to all memory locations, including the kernel data structures of the operating system

Mode change

- User mode, normal mode → kernel mode, privileged mode
- Interrupt or special SVC instructions (service request)
- Kernel mode → User mode
- Privileged instruction, for example IRET (return from interrupt, interrupt return)
- Returns the control and mode as they were before the mode change
  - Very similar with return from a subroutine

Layers of the I/O system

- Direct I/O (Suora I/O)
- Indirect I/O (Epäsuora I/O)
- DMA I/O

Teemu’s cheese cake

- Register, on-chip cache, memory, disk, and tape speeds relative to times locating cheese for the cheese cake you are baking...

Which now common technology is missing from picture?
Review Questions

- Main parts of a computing system?
- DMA: principles and functionalities?
- Obligatory hardware and its features?
- How to make CPU to execute normal user program? Operating system?

Digital logic

Stallings:
- Online Chapters 20.1-3
- Boolean Algebra
- Simplification
- Gates
- Combination Circuits

Boolean Algebra

- George Boole
  - ideas 1854
  - Claude Shannon (MSc thesis, "Maths")
  - apply to circuit design, 1938
  - "father of information theory"

Topics:
- Describe digital circuitry function
- programming language?
- Optimise given circuitry
- Use algebra (Boolean algebra) to manipulate (Boolean) expressions into simpler expressions

Boolean Algebra

- Variables: A, B, C
- Values: TRUE (1), FALSE (0)
- Basic logical operations:
  - binary: AND (·) $A \cdot B = AB$
  - OR (+) $B + C = B + C$
  - unary: NOT (') $A$
- Composite operations, equations
  - precedence: NOT, AND, OR
  - parenthesis

\[ D = A + \overline{B} \cdot C = A + (\overline{B}C) \]
**Boolean Algebra**

- Other operations
  - XOR (exclusive-or)
  - NAND
  - NOR
- Truth tables
  - What is the result of the operation?

### Truth tables

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>NOT P</th>
<th>P AND Q</th>
<th>P OR Q</th>
<th>P NAND Q</th>
<th>P NOR Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Function

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>NOT P</th>
<th>P AND Q</th>
<th>P OR Q</th>
<th>P NAND Q</th>
<th>P NOR Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Operators

\[
A \land B = \overline{A} \lor \overline{B}
\]

### Truth tables

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>NOT P</th>
<th>P AND Q</th>
<th>P OR Q</th>
<th>P NAND Q</th>
<th>P NOR Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Simplification

\[
F = \overline{A} \lor \overline{B}C + A \lor \overline{B}C
\]

### Sum-of-Products

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Product-of-sums

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Gates, circuits, combination circuits

- Implement basic Boolean algebra operations
- Gates - fundamental building blocks
  - 1 or 2 inputs, 1 output
- Combine to build more complex circuits
  - memory, adder, multiplier, …
- Gate delay in combination circuits
- Change inputs; after (combined) gate delay new output available
  - 1 ns? 10 ns? 0.1 ns?

### Graphical Symbols

- Sum-of-Products
- Product-of-sums

### Describing the Circuit

a) Boolean equations

\[
F = \overline{A} \lor \overline{B}C + A \lor \overline{B}C
\]

b) Truth table

### Simplification of Circuits

- Algebraic Simplification

\[
F = \overline{A} \lor \overline{B}C + A \lor \overline{B}C = \overline{A}B + \overline{C}B = B(\overline{A} + \overline{C})
\]

- Simplification with Karnaugh Maps
  - E.g., see Kerola slides 2003/appa
Using Karnaugh Maps to Minimize Boolean Functions

Original function:
\[ f = \overline{abcd} \overline{abc}d + \overline{abcd} \overline{abc}d \overline{abcd} + \overline{abcd} \overline{abc}d + \overline{abc}d \]

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:
\[ f = bd + ac + ab \]

Discussion?

Multiplexers

- Select one of many possible inputs to output
- Black box
- Truth table
- Implementation
- Each input/output “line” can be many parallel lines
  - Select one of 16 1-bit values
  - \( R_{data}, R_{IR}, ALU_{CS} \)
  - Simple extension to one line selection
  - Lots of wires, plenty of gates
- Used to control signal and data routing
  - Example: loading the value of PC

Encoders/Decoders

- Exactly one of many Encoder input or Decoder output lines is 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

Decoder

Read-Only-Memory (ROM)

- Given input values (address), get output value (contents)
  - Like multiplexer, but with fixed data

4-bit deep data

Mem (7) = 4
Mem (11) = 14

ROM truth table

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Select lines "address"
Adders

1-bit adder
A=1  \rightarrow  ?  \rightarrow  Carry=0  \rightarrow  Sum=1
B=0  \rightarrow  ?  \rightarrow  Carry=1  \rightarrow  Sum=0

1-bit adder with carry
Carry=1  \rightarrow  ?  \rightarrow  Carry=1
A=1  \rightarrow  ?  \rightarrow  Sum=0

Implementation

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

Simple processor

http://www.gamezero.com/team-0/articles/math_magic/microstage4.htm