Class Overview

- Cover hardware operation of digital computers.
- First, consider the various digital components used in the organization and design.
- Second, go through the necessary steps to design a basic computer.
- Third, look in detail at the organization and architecture of the CPU.
- Finally, if time permits, go over the organization of I/O and memory.

Chapter 1 – Digital Logic Circuits

Section 1.1 – Digital Computers

The word *digital* implies information in computers is represented by variables that take a limited number of discrete values.

Decimal digits 0, 1, …, 9 provide ten discrete values.

The physical restrictions of the components that are used restrict us to two states:

true/false  on/off  yes/no  high/low  0/1

Binary digits, 0 or 1, are called bits.

Groups of bits can represent binary numbers, decimal digits, or letters.

01000001b = 65d = 41h = ‘A’ = some control word

Computer system is divided into two functional entities: hardware and software.

Hardware: Lowest level in a computer are all the electronic circuits and physical devices from which it is built.

Software: Sequences of instructions and data that make computers do useful work.

- Program: sequence of instructions for a particular task
- Operating system:
  - programs included in system software package.
  - link between hardware and user needs.

3 components of hardware:

- CPU:
  - Controls operation of system by issuing timing and control signals
  - Contains ALU which does all computation on data
  - Contains registers, control circuits for fetching and executing instructions
• Memory: Storage for instructions and data (no distinction). RAM because CPU can access any location in memory at random and retrieve the binary info within a fixed interval of time.

• Input/Output: interface to transfer info between computer and outside world, e.g. keyboard, printer, scanner, mass storage, etc.

3 viewpoints for hardware:

• Computer organization: interconnection of h/w to form the computer system
• Computer design: the determination of how to interconnect the components and which components to used based upon some specs
• Computer architecture: the structure and behavior of the computer perceived by the user. Includes instruction format, instruction set, and techniques for addressing memory.

Section 1.2 – Logic Gates

Binary info (0/1) is represented in digital computers by voltage signals.

\[
\begin{align*}
3 \text{ V} &= \text{binary 1 = high} \\
0.5 \text{ V} &= \text{binary 0 = low}
\end{align*}
\]

The manipulation of binary information is done by logic circuits called gates. Gates are blocks of h/w that produce signals of 1 or 0 when input logic requirements are satisfied.

Each logic gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression. The input/output relationship is represented by a truth table.
<table>
<thead>
<tr>
<th>Name</th>
<th>Graphic symbol</th>
<th>Algebraic expression</th>
<th>Truth table</th>
</tr>
</thead>
</table>
| AND      | ![AND symbol](image1) | \( x = A \cdot B \) or \( x = A + B \) | \( \begin{array}{c|c|c|c} A & B & x \\ 
0 & 0 & 0 \\ 
0 & 1 & 0 \\ 
1 & 0 & 0 \\ 
1 & 1 & 1 \\ 
\end{array} \) |
| OR       | ![OR symbol](image2) | \( x = A + B \) | \( \begin{array}{c|c|c|c} A & B & x \\ 
0 & 0 & 0 \\ 
0 & 1 & 1 \\ 
1 & 0 & 1 \\ 
1 & 1 & 1 \\ 
\end{array} \) |
| Inverter | ![Inverter symbol](image3) | \( x = \overline{A} \) | \( \begin{array}{c|c} A & x \\ 
0 & 1 \\ 
1 & 0 \\ 
\end{array} \) |
| Buffer   | ![Buffer symbol](image4) | \( x = A \) | \( \begin{array}{c|c} A & x \\ 
0 & 0 \\ 
1 & 1 \\ 
\end{array} \) |
| NAND     | ![NAND symbol](image5) | \( x = \overline{A \cdot B} \) | \( \begin{array}{c|c|c|c} A & B & x \\ 
0 & 0 & 1 \\ 
0 & 1 & 1 \\ 
1 & 0 & 1 \\ 
1 & 1 & 0 \\ 
\end{array} \) |
| NOR      | ![NOR symbol](image6) | \( x = \overline{A + B} \) | \( \begin{array}{c|c|c|c} A & B & x \\ 
0 & 0 & 1 \\ 
0 & 1 & 0 \\ 
1 & 0 & 0 \\ 
1 & 1 & 0 \\ 
\end{array} \) |
| Exclusive-OR | ![Exclusive-OR symbol](image7) | \( x = A \oplus B \) or \( x = \overline{A \cdot B} \) | \( \begin{array}{c|c|c|c} A & B & x \\ 
0 & 0 & 0 \\ 
0 & 1 & 1 \\ 
1 & 0 & 1 \\ 
1 & 1 & 0 \\ 
\end{array} \) |
| Exclusive-NOR (or equivalence) | ![Exclusive-NOR symbol](image8) | \( x = A \oplus B \) or \( x = \overline{A \cdot B} \) | \( \begin{array}{c|c|c|c} A & B & x \\ 
0 & 0 & 1 \\ 
0 & 1 & 0 \\ 
1 & 0 & 0 \\ 
1 & 1 & 1 \\ 
\end{array} \) |

Figure 1-2 Digital logic gates.
Section 1.3 – Boolean Algebra

Boolean algebra uses binary variables and logic operations. The three basic logic operations are AND, OR, and complement.

The value of a Boolean function can be either 1 or 0. Consider the following:

\[ F = x + y'z \]

The relationship between a function and its binary variables can be represented in a truth table – \( 2^n \) combinations of the \( n \) binary variables.

The function can also be represented by a logic diagram.

![Truth table and logic diagram for \( F = x + y'z \).](image)

Boolean algebra allows us to:

- Analyze and design digital circuits
- Express in algebraic form a truth table relationship between binary variables
- Express in algebraic form the i/o relationship of logic diagrams
- Find simpler circuits for the same function

Different identities of Boolean algebra allow us to manipulate expressions and simplify.
F = AB' + C'D + AB' + C'D
= AB' + C'D (by identity 5)

F = ABC + ABC' + A'C
= AB(C + C') + A'C (by identity 13)
= AB(1) + A'C (by identity 7)
= AB + A'C (by identity 4)
DeMorgan’s theorem states:
- OR-invert = invert-AND    (NOR)    identity 15
- AND-invert = invert-OR    (NAND)   identity 16

Figure 1-4  Two graphic symbols for NOR gate.

Figure 1-5  Two graphic symbols for NAND gate.
Methods to complement a function:
- Interchange 1’s and 0’s for the values of $F$ in the truth table
- Use DeMorgan’s theorem on algebraic function
  - Change OR to AND
  - Change AND to OR
  - Complement each individual variable

\[ F = AB + C'D + B'D \]
\[ F' = (A' + B')(C + D')(B + D') \]

Section 1.4 – Map Simplification

- Truth table representation of a function is unique
- Algebraic representation of a function is not unique
- Sometimes difficult to simplify algebraic function
- A Karnaugh map is a pictorial arrangement of the truth table which allows an easy interpretation for choosing the minimum number of terms needed to express the function algebraically
- Each combination of variables in a truth table is called a minterm
- A truth table of $n$ variables has $2^n$ minterms
- A Boolean function is equal to 1 for some minterms and 0 for others
- Another representation of the function is to list the decimal equivalent of those minterms that produce a 1 for the function

\[ F = x + y'z \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ F(x,y,z) = \Sigma (1, 4, 5, 6, 7) \]

- A Karnaugh map is a diagram made up of squares, with each square representing one minterm.
- The value entered in a square corresponds to the output result of the minterm
- Simplification of the function is achieved by recognizing patterns and combining squares marked by 1’s
The minterm numbers are assigned such that adjacent squares represent minterms that differ by only one variable.
Group adjacent squares containing 1 -- the number of squares is a power of 2.
Each group of squares represents an algebraic term, and the OR of those terms gives the simplified algebraic expression for the expression:

\[ F(A,B,C) = \Sigma (3, 4, 6, 7) \]

**Figure 1.7** Maps for two-, three-, and four-variable functions.

\[ F = BC + AC' \]
\[ F(A,B,C) = \sum(0,2,4,5,6) \]

\[ \Rightarrow F = C' + AB' \]

\[ F(A,B,C,D) = \sum(0,1,2,6,8,9,10) \]

\[ \Rightarrow F = B'D' + B'C' + A'CD' \]

- Expressions obtained in previous examples are *sum-of-products*
- Procedure to obtain *product-of-sums*
  - Mark empty squares in Karnaugh map with 0’s
  - Obtain the complement \( F' \) by combining adjacent squares of 0’s
  - Take the complement of \( F' \) to obtain \( F \) in product-of-sums form
\[ F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10) \]

Figure 1-11  Map for \( F(A, B, C, D) = \Sigma(0,1,2,5,8,9,10) \).

\[ \Rightarrow F = B'D' + B'C' + A'C'D \]

\[ \Rightarrow F' = AB + CD + BD' \]  

(pick off 0's)

\[ \Rightarrow F = (A' + B')(C' + D')(B' + D) \]  

(DeMorgan)

Figure 1-12  Logic diagrams with AND and OR gates.

- Sum-of-products can be implemented with NAND gates
- Product-of-sums can be implemented with NOR gates
Minterms that may produce either 0 or 1 for the function are *don’t care conditions.*

- Mark don’t cares with an *x* in the map.
- When combining squares to simplify the expression, don’t cares can be taken to be either 1 or 0 – whichever gives the simplest expression.

\[
F(A,B,C) = \sum (0,2,6) \\
d(A,B,C) = \sum (1,3,5)
\]

\[
\Rightarrow F = A'C' + BC' \quad \text{(without don’t cares)}
\]

\[
\Rightarrow F = A' + BC' \quad \text{(includes don’t cares)}
\]

\[
\Rightarrow F(A,B,C) = \sum (0,1,2,3,6) \quad \text{(includes don’t cares)}
\]
Section 1.5 – Combinational Circuits

- A *combinational circuit* is a connected arrangement of logic gates with a set of inputs and outputs
- The binary values of the outputs are a function of the binary combination of the inputs

![Block diagram of a combinational circuit](image)

- A truth table showing the binary relationship between the \( n \) input variables and the \( m \) output variables can describe the combinational circuit
- A list of \( m \) Boolean functions expressed in terms of the \( n \) input variables can also describe the combinational circuit
- When analyzing, the result should be either a set of Boolean functions or a truth table
- The design process of a combinational circuit is as follows:
  - State the problem
  - Assign letter symbols to the input and output
  - Derive the truth table
  - Obtain simplified Boolean functions for each output
  - Draw the logic diagram

- A *half-adder* is a combinational circuit that performs the arithmetic addition of two bits
- There are two input variables – \( x \) and \( y \)
- There are two output variables – \( S \) and \( C \) (sum and carry)
\[ S = x'y + xy' = x \oplus y \]
\[ C = xy \]

- A full-adder performs the addition of three bits – two significant bits and a previous carry
- There are three input variables – \( x, y, z \)
- There are two output variables – \( S \) and \( C \) (sum and carry)
- Two half-adders are used to design a full-adder

### TABLE 1-2 Truth Table for Full-Adder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y )</td>
</tr>
<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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Section 1.6 – Flip-Flops

- *Synchronous sequential circuits* use signals that affect storage elements only at discrete instants of time
- The signals are a periodic train of clock pulses
- The storage elements employed in clocked sequential circuits are called *flip-flops*
- A flip-flop is a binary cell capable of storing one bit of information
- It has two outputs
• A flip-flop maintains a binary state until directed by a clock pulse to switch states

• **SR flip-flops:**
  o three inputs: \( S \) (set), \( R \) (reset), and \( C \) (clock)
  o two outputs: \( Q \) and \( Q' \)
  o responds to a positive transition of the input clock signal
  o \( Q(t) \) – present state
  o \( Q(t+1) \) – next state

![SR flip-flop diagram](image)

**Figure 1-19** SR flip-flop.

• **D flip-flops:**
  o two inputs: \( D \) (data), and \( C \) (clock)
  o two outputs: \( Q \) and \( Q' \)
  o responds to a positive clock transition
  o no “no change” condition
  o accomplish “no change” by disabling clock or by feeding output back to input

![D flip-flop diagram](image)

**Figure 1-20** D flip-flop.

• **JK flip-flops:**
  o three inputs: \( J, K, C \)
  o two outputs: \( Q \) and \( Q' \)
  o responds to a positive clock transition
  o similar to SR flip-flop, but has a complement condition
• **T-flip-flops:**
  - two inputs: \( T \) (toggle) and \( C \)
  - two outputs: \( Q \) and \( Q' \)
  - responds to a positive clock transition
  - only two conditions
  - \( Q(t+1) = Q(t) \oplus T \)

• **In an edge-triggered flip-flop, output transitions occur at a specific level of the clock pulse**
• When the pulse input level exceeds this threshold level, the inputs are locked out so that the flip-flop is unresponsive to further changes in inputs until the clock pulse returns to 0 and another pulse occurs
• **Positive-edge transition:** transition on the rising edge of the clock signal
• **Falling-edge transition:** transition on the falling edge
• **Positive clock transition** includes setup time and hold time
• **Setup time:** minimum time in which the input must remain at a constant value before the transition
• **Hold time:** a definite time in which the input must not change after the positive transition
• Special inputs called *preset* and *clear* are sometimes provided to set or clear the flip-flop asynchronously.
• These inputs are useful for bringing the flip-flops to an initial state prior to its clocked operation.

- Characteristic tables of flip-flops specify the next state when the inputs and the present state are known.
- During design, we usually know the required transition from present state to next state and wish to find the flip-flop input conditions that will cause the required transition.
- An *excitation table* lists the required input combinations for a given change of state.
- Each table consists of two columns for present state and next state, and a column for each input to show how the input is achieved.
- There are four possible transitions from present state to next state.
The information for the excitation tables are derived from the characteristic tables.

Don’t care conditions exist when there are two ways of achieving the required transition.

### Section 1.7 – Sequential Circuits

- **A sequential circuit** is an interconnection of flip-flops and gates.
- Any number or type of flip-flops may be used.
- The combinational circuit receives binary signals from external inputs and from the outputs of the flip-flops.
- The outputs of the combinational circuit go to the external outputs and to inputs of flip-flops.
- The external outputs of a sequential circuit are functions of both external inputs and the present state of the flip-flops.
- Also, the next state of the flip-flops is also a function of their present state and external inputs.
- Thus, a sequential circuit is specified by a time sequence of external inputs, external outputs, and internal flip-flop binary states.
The interconnections among the gates in the combinational circuit can be specified by a set of Boolean expressions.

Flip-flop equations are a set of Boolean expressions that describe the part of the combinational circuit that generates the inputs to the flip-flops.

Set convention is used to denote input variables.

Consider example in figure 1-25:

\[ D_A = Ax + Bx \]
\[ D_B = A'x \]
\[ y = Ax' + Bx' \]

- A sequential circuit is specified by a state table that relates outputs and next states as a function of inputs and present states.
- A clock signal activates the transition from present state to next state in clocked sequential circuits.
- A state table consists of four sections:
  - present state – states of flip-flops at any given time \( t \)
  - input – all combinations of external inputs for each possible present state
  - next state – states of the flip-flops one clock period later at time \( t + 1 \)
  - output – value of external outputs for each present state in and input condition
- A sequential circuit with \( m \) flip-flops, \( n \) inputs, and \( p \) outputs will contain \( m \) columns for present state, \( n \) columns for inputs, \( m \) columns for next state, and \( p \) columns for outputs.
- The present state and input columns are combined and the \( 2^{m+n} \) binary combinations are listed under them.

**TABLE 1-4 State Table for Circuit of Fig. 1-25**

<table>
<thead>
<tr>
<th>Present state</th>
<th>Input</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A ) ( B )</td>
<td>( x )</td>
<td>( A ) ( B )</td>
<td>( y )</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0</td>
<td>1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>1 0</td>
<td>0</td>
</tr>
</tbody>
</table>

- A state diagram is a graphical representation of a state table.
- Circles represent each state.
- Directed lines connecting the circles represent state transitions.
- Input and output values are noted above the directed lines.
The procedure for designing sequential circuits consists of first translating the circuit specifications into a state diagram.

Next, convert the state diagram into a state table.

Finally, from the state table, obtain the information for determining the logic circuit diagram.

Problem statement: Design a clocked sequential circuit that goes through a sequence of repeated binary states 00, 01, 10, and 11 when an external input \( x \) is equal to 1. The state of the circuit remains unchanged when \( x = 0 \). (2-bit binary counter)
• Two flip-flops are required to represent the two bits

![State diagram for binary counter.](image)

• Fill in the columns for present state, input, and next state
• The excitation table is an extension of the state table that consists of a list of flip-flop excitations that will cause the required state transitions. (This depends upon the type of flip-flop used.)

<table>
<thead>
<tr>
<th>Present state</th>
<th>Input</th>
<th>Next state</th>
<th>Flip-flop inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>x</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 1-28  Maps for combinational circuit of counter.

Figure 1-29  Logic diagram of a 2-bit binary counter.