Chapter # 1: Introduction

Contemporary Logic Design

Randy H. Katz
University of California, Berkeley
May 1994

The Process Of Design

Design
- Initial concept: what is the function performed by the object?
- Constraints: How fast? How much area? How much cost?
- Refine abstract functional blocks into more concrete realizations

Implementation
- Assemble primitives into more complex building blocks
- Composition via wiring
- Choose among alternatives to improve the design

Debug
- Faulty systems: design flaws, composition flaws, component flaws
- Design to make debugging easier
- Hypothesis formation and troubleshooting skills
• Complex systems can be described from three independent viewpoints:

  – **Functional**: describes the behavior in terms of its inputs and outputs
  
  – **Structural**: describes how the system is broken down into ever more primitive components that form its implementation
  
  – **Physical**: describes the detailed placement and interconnection of the primitive building blocks that make up the implementation

---

**The Art Of Design: Refinement of Representations**

1. **Functional Specification/What the System Does**

   *Ex: Traffic Light Controller*

   - Lights point in the directions N, S, E, W
   - Illuminates the same lights N as S and E as W
   - Cycles thru the sequence GREEN-YELLOW-RED
   - N-S and E-W never GREEN or YELLOW at the same time
   - Stay GREEN for 45 seconds, yellow for 15, red for 60

2. **Performance Constraints/Requirements to be Met**

   - speed: compute changes in under 100 ms
   - power: consume less than 20 watts
   - area: implementation in less than 20 square cm
   - cost: less than $20 in manufacturing costs
The Art of Design: "To Design Is To Represent"

1. **English language specification**
   - easy to write, but not precise and subject to ambiguity

2. **Functional description**
   - more precise specification
   - flow charts, program fragments

3. **Structural description**
   - complex components decomposed into compositions of less complex components

4. **Physical description**
   - the design in terms of most primitive building blocks, e.g., logic gates or transistors

---

The Process of Design

*Implementation as Assembly*

**Top Down Design:**
- Complex functions replaced by more primitive functions

**Bottom Up Design:**
- Primitives composed to build more and more complex assemblies

**Rules of Composition and Correctness by Construction:**
- Electrical Rules: how many components can be cascaded?
- Timing Rules: how does the system change in conjunction with periodic triggering events?
The Process of Design

Top Down Decomposition

Structural Representation

To decomposition of high level functions into more primitive functions

The Process of Design

Bottom Up Assembly

Primitives composed to build more and more complex assemblies

e.g., a group of rooms form a floor

e.g., a group of floors form a bldg.

a group of transistors form a gate

a group of gates form an addition circuit

addition circuits plus storage circuits form a processor datapath
The Process of Design

Physical Representation:

• In general, there is more than one way to realize a particular hardware function.

• Design optimization often involves selecting among alternative components and assemblies, choosing the best for the task at hand within the speed, power, area, and cost constraints on the design.

The Process of Design: Debugging the System

What Can Go Wrong

• **Design Flaws**
  - Implementation does not meet functional specification
  - Logic design is incorrect (wrong function implemented)
  - Misinterpretation or corner cases ignored

• **Implementation Flaws**
  - Individual modules function correctly but their compositions do not
  - Misunderstanding of interface and timing behavior
  - Wiring mistakes, Electrical mistakes

• **Component Flaws**
  - Logically correct and correctly wired
  - Not all hardware components are guaranteed to work!
  - E.g., burnt out component
The Process of Design

Debugging via Simulation Before Construction

Debugging Skills:

• Improving the testability of the design

• Formulating a testing plan and choosing test cases

• Hypothesizing about the cause of the problem

• Isolating portions of the implementation for testing

• Effective use of laboratory instruments for troubleshooting

Digital Hardware Systems

Digital Systems
Digital vs. Analog Waveforms

Digital: only assumes discrete values

Analog: values vary over a broad range continuously
**Advantages of Digital Systems**

Analog systems: slight error in input yields large error in output

Digital systems more accurate and reliable
   Readily available as self-contained, easy to cascade building blocks

Computers use digital circuits internally

Interface circuits (i.e., sensors & actuators) often analog

*This course is about logic design, not system design (processor architecture), not circuit design (transistor level)*

---

**Digital Binary Systems**

- **Two discrete values:**
  yes, on, 5 volts, current flowing, magnetized North, "1"
  no, off, 0 volts, no current flowing, magnetized South, "0"

- **Advantage of binary systems:**
  rigorous mathematical foundation based on logic

- **Logic gates examples:**
  IF the garage door is open
  AND the car is running
  THEN the car can be backed out of the garage

- **Logic gates examples:**
  IF N-S is green
  AND E-W is red
  AND 45 seconds has expired since the last light change
  THEN we can advance to the next light configuration

  *the three preconditions must be true to imply the conclusion*
Digital Hardware Systems

Boolean Algebra and Logical Operators

**Algebra:** variables, values, operations

In Boolean algebra, the values are the symbols 0 and 1
If a logic statement is false, it has value 0
If a logic statement is true, it has value 1

Operations: AND, OR, NOT

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X AND Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X OR Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>NOT X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Digital Hardware Systems

Hardware Systems and Logical Operators

**IF** the garage door is open
AND the car is running
**THEN** the car can be backed out of the garage

<table>
<thead>
<tr>
<th>door open?</th>
<th>car running?</th>
<th>back out car?</th>
</tr>
</thead>
<tbody>
<tr>
<td>false/0</td>
<td>false/0</td>
<td>false/0</td>
</tr>
<tr>
<td>false/0</td>
<td>true/1</td>
<td>false/0</td>
</tr>
<tr>
<td>true/1</td>
<td>false/0</td>
<td>false/0</td>
</tr>
<tr>
<td>true/1</td>
<td>true/1</td>
<td>TRUE/1</td>
</tr>
</tbody>
</table>
Digital Hardware Systems

The Real World

Physical electronic components are continuous, not discrete!
These are the building blocks of all digital components!

Transition from logic 1 to logic 0 does not take place instantaneously in real digital systems
Intermediate values may be visible for an instant

Boolean algebra useful for describing the steady state behavior of digital systems
Be aware of the dynamic, time varying behavior too!

Digital Hardware Systems

Digital Circuit Technologies

Integrated circuit technology
choice of conducting, non-conducting, sometimes conducting ("semiconductor") materials
whether or not their interaction allows electrons to flow forms the basis for electrically controlled switches

Main technologies
MOS: Metal-Oxide-Silicon

Bipolar
Transistor-Transistor Logic
Emitter Coupled Logic
**Digital Hardware Systems**

**Combinational vs. Sequential Logic**

Network implemented from switching elements or logic gates. The presence of feedback distinguishes between **sequential** and **combinational** networks.

**Combinational logic**
- no feedback among inputs and outputs
- outputs are a pure function of the inputs
- e.g., full adder circuit:
  - \((A, B, \text{Carry In})\) mapped into \((\text{Sum}, \text{Carry Out})\)

**Sequential logic**
- inputs and outputs overlap
- outputs depend on inputs **and** the entire history of execution!

- network typically has a limited number of unique configurations
- these are called **states**
- e.g., traffic light controller sequences infinitely through four states

- new component in sequential logic networks: **storage elements** to remember the current state

- output and new state is a function of the inputs and the old state
  - i.e., the fed back inputs are the state!

**Synchronous systems**
- period reference signal, the clock, causes the storage elements to accept new values and to change state

**Asynchronous systems**
- no single indication of when to change state
Digital Hardware Systems

Combination vs Sequential Logic

Traffic Light Example

Next State Logic
Maps current state and alarm events into the next state

Current State
Storage elements replaced by next state when the clock signal arrives

Output Logic
Current state mapped into control signals to change the lights and to start the event timers

IF controller in state N-S green, E-W red AND the 45 second timer alarm is asserted THEN the next state becomes N-S yellow, E-W red when the clk signal is next asserted

Representations of a Digital Design

Switches

A switch connects two points under control signal.

Normally Open when the control signal is 0 (false), the switch is open
when it is 1 (true), the switch is closed

Normally Closed when control is 1 (true), switch is open
when control is 0 (false), switch is closed
**Examples:** routing inputs to outputs through a maze

**EXAMPLE:**

IF car in garage
AND garage door open
AND car running
THEN back out car

**EXAMPLE:**

IF car in driveway
OR (car in garage
AND NOT garage door closed)
AND car running
THEN can back out car

Floating nodes:
what happens if the car is not running?
outputs are floating rather than forced to be false

Under all possible control signal settings
(1) all outputs must be connected to some input through a path
(2) no output is connected to more than one input through any path

**Representations of a Digital Design: Switches**

**Implementation of AND and OR Functions with Switches**

**AND function**
Series connection to TRUE

**OR function**
Parallel connection to TRUE
Representations of a Digital Design

Truth Tables

tabulate all possible input combinations and their associated output values

Example: half adder
adds two binary digits to form Sum and Carry

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Example: full adder
adds two binary digits and Carry in to form Sum and Carry Out

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

NOTE: 1 plus 1 is 0 with a carry of 1 in binary

Representations of a Digital Design

Boolean Algebra

values: 0, 1
variables: A, B, C, ..., X, Y, Z
operations: NOT, AND, OR, ...

NOT X is written as \( \overline{X} \)
X AND Y is written as \( X \land Y \), or sometimes \( X \cdot Y \)
X OR Y is written as \( X \lor Y \)

Deriving Boolean equations from truth tables:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{Sum} = \overline{A} \overline{B} + A \overline{B} \\
\text{Carry} = A B
\]

OR'd together product terms for each truth table row where the function is 1
if input variable is 0, it appears in complemented form;
if 1, it appears uncomplemented
Another example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Sum = A B Cin + A B Cin + A B Cin + A B Cin

Cout = A B Cin + A B Cin + A B Cin + A B Cin

Reducing the complexity of Boolean equations

Laws of Boolean algebra can be applied to full adder’s carry out function to derive the following simplified expression:

Cout = A Cin + B Cin + A B

Verify equivalence with the original Carry Out truth table:

place a 1 in each truth table row where the product term is true

each product term in the above equation covers exactly two rows in the truth table; several rows are “covered” by more than one term
Representations of a Digital Design

Gates

most widely used primitive building block in digital system design

Standard Logic Gate Representation

Half Adder Schematic

---

Representations of a Digital Design: Gates

Full Adder Schematic

Fan-in: number of inputs to a gate
Fan-out: number of gate inputs an output is connected to

Technology "Rules of Composition" place limits on fan-in/fan-out
Representations of a Digital Design

Waveforms

dynamic behavior of a circuit
real circuits have non-zero delays

Timing Diagram of the Half Adder

Output changes are delayed from input changes
The propagation delay is sensitive to paths in the circuit
Outputs may temporarily change from the correct value to the wrong value back again to the correct value: this is called a glitch or hazard

Representations of a Digital Design: Waveforms

Tracing the Delays: A=0, B=0 to A=0, B=1

(i) Initial conditions
(ii) Y changes from 0 to 1
(iii) Output of top AND gate changes after 10 time units
(iv) Output of OR gate changes after 10 time units

10 time units of delay

电路故障: 1+0不是1,而是0!
Representations of a Digital Design

Blocks

structural organization of the design
black boxes with input and output connections
 corresponds to well defined functions
concentrates on how the components are composed by wiring

Full Adder realized in terms of composition of half adder blocks

Block diagram representation of the Full Adder

Representations of a Digital Design

Waveform Verification

Does the composed full adder behave the same as the full gate implementation?

Sum, Cout waveforms lag input changes in time
How many time units after input change is it safe to examine the outputs?
Representation of a Digital Design: Behaviors

ABEL Hardware Description Language

MODULE half_adder;
  a, b, sum, carry PIN 1, 2, 3, 4;
  TRUTH_TABLE [(a, b) -> (sum, carry)]
  [0, 0] -> [0, 0];
  [0, 1] -> [1, 0];
  [1, 0] -> [1, 0];
  [1, 1] -> [0, 1];
END half_adder;

MODULE half_adder;
  a, b, sum, carry PIN 1, 2, 3, 4;
  EQUATIONS
  SUM = (A & !B) # (!A & B);
  CARRY = A & B;
END half_adder;

Truth Table
Specification

Equation
Specification

AND OR NOT

Representations of a Digital Design

Behaviors

Hardware description languages
structure and function of the digital design

Example: Half Adder in VHDL

--- ***** inverter gate model *****
--- external ports
ENTITY inverter_gate;
  PORT (a: IN BIT;  z: OUT BIT);
END inverter_gate;

--- internal behavior
ARCHITECTURE behavioral OF inverter_gate IS
BEGIN
  z <= NOT a AFTER 10 ns;
END behavioral;

--- ***** and gate model *****
--- external ports
ENTITY and_gate;
  PORT (a, b: IN BIT;  z: OUT BIT);
END and_gate;

--- internal behavior
ARCHITECTURE behavioral OF and_gate IS
BEGIN
  z <= a AND b AFTER 10 ns;
END behavioral;

Black Box View
as seen by outside world

Internal Behavior
Note delay statement

No. 1-35
No. 1-36
Representation of a Digital Design: Behaviors

-- ***** or gate model *****
-- external ports
ENTITY or_gate;
PORT (a, b: IN BIT; z: OUT BIT);
END or_gate;
-- internal behavior
ARCHITECTURE behavioral OF or_gate IS
BEGIN
z <= a OR b AFTER 10 ns;
END behavioral;

-- ***** half adder model *****
-- external ports
ENTITY half_adder;
PORT (a_in, b_in: INPUT; sum, c_out: OUTPUT);
END half_adder;
-- internal structure
ARCHITECTURE structural of half_adder IS
-- component types to use
COMPONENT inverter_gate
PORT (a: IN BIT; z: OUT BIT); END COMPONENT;
COMPONENT and_gate
PORT (a, b: IN BIT; z: OUT BIT); END COMPONENT;
COMPONENT or_gate
PORT (a, b: IN BIT; z: OUT BIT); END COMPONENT;
-- internal signal wires
SIGNAL s1, s2, s3, s4: BIT;
--- Textual description of the netlist
BEGIN
--- one line for each gate, describing its type and connections
i1: inverter_gate PORT MAP (a_in, s1);
i2: inverter_gate PORT MAP (b_in, s2);
a1: and_gate PORT MAP (b_in, s1, s3);
a2: and_gate PORT MAP (a_in, s2, s4);
o1: or_gate PORT MAP (s3, s4, sum);
END structural;

This VHDL specification corresponds to the following labeled schematic
Rapid Electronic System Prototyping:

Computer-Aided Design

Synthesis tools

- create a portion of the design from other portions
- map more abstract representation to more physical representation

VHDL → Behavioral Synthesis → Boolean Equations → Logic Synthesis → Schematics

ABEL → Gate Libraries

map a representation into a more optimized form of that representation, e.g., espresso

Rapid Electronic System Prototyping

Simulation

- program which dynamically executes an abstract design description
- obtain verification of functional correctness and some timing information before the design is physically constructed
- easier to probe and debug a simulation than an implemented design
- simulation cannot guarantee that a design will work only as good as the test cases attempted
- does not check electrical errors
- abstracts away some of the realities of a real system

Logic Simulation

- design described in terms of logic gates
- values are 0, 1 (plus others to be introduced)
- good for truth table verification

Timing Simulation

- waveform inputs and outputs
- model of gate delays
- are the waveform shapes what was expected?
- identification of performance bottlenecks