Combinational Circuits

Jason Filippou

CMSC250 @ UMCP

06-02-2016
Hardware design levels
Levels of abstraction for ICs

- **Small Scale Integration**: \( \approx 10 \) boolean gates
- **Medium Scale Integration**: \( > 10, \leq 100 \)
- **Large Scale Integration**: Anywhere between 100 and 30,000.
- **Very Large Scale Integration**: Up to 150,000.
- **Very Very Large Scale Integration**: \( > 150,000 \).
Figure 1: VLSI schematic for a professional USB interface for Macintosh PCs.
SSI

- Consists of circuits that contain about 10 gates.
  - 16-bit adders.
  - Encoders / Decoders.
  - MUX/DEMUX.
  - ...

- All our examples will be at an SSI level.

- **Logic Design** is the branch of Computer Science that essentially analyzes the kinds of circuits and optimizations done at an SSI/MSI level.
  - We do not have such a course in the curriculum, but you can expect to be exposed to it if you ever take 411.
Combinational Circuit Design
A combinational circuit is one where the output of a gate is never used as an input into it.
Combinational vs Sequential circuits

- A **combinational** circuit is one where the output of a gate is never used as an input into it.
- A **sequential** circuit is one where this constraint can be violated.
Combinational vs Sequential circuits

- A **combinational** circuit is one where the output of a gate is never used as an input into it.
- A **sequential** circuit is one where this constraint can be violated.
  - Example: Flip-flops (yes, seriously).
- In this course, we will only touch upon combinational circuits.
Boolean Gates
Analogy between logic and hardware

- Every binary or unary boolean operator ($\land$, $\lor$, $\sim$) is mapped to a gate.
- This includes the binary connectives ($\Rightarrow$, $\Leftrightarrow$) (why?)
- So, every propositional logic construct ("compound" or otherwise) can be mapped to a logically equivalent boolean circuit!
AND gate

Figure 2: The ANSI symbol of an AND gate.

Truth table corresponds to $\land$ binary operator. (board)
OR gate

Figure 3: The ANSI symbol of an OR gate.

Truth table corresponds to ∨ binary operator.
NOT gate (inverter)

Figure 4: The ANSI symbol of a NOT gate.

Truth table corresponds to \( \sim \) unary operator.
**XOR gate**

![XOR Gate Diagram](image)

*Figure 5: ANSI symbol for a XOR gate.*

Truth table? (whiteboard)
Can I implement a XOR gate using gates I know?
XOR gate

- Can I implement a XOR gate using gates I know?
- Homework 😊
NAND/NOR gates

Figure 6: ANSI symbol for a NAND gate

Figure 7: ANSI symbol for a NOR gate

What’s the truth table for both of those?
NAND/NOR gates

Figure 6: ANSI symbol for a NAND gate

Figure 7: ANSI symbol for a NOR gate

- What’s the truth table for both of those?
- Sheffer stroke: ↑ (used for NAND operation)
- Quince arrow: ↓ (used for NOR operation)
NAND/NOR gates

Figure 6: ANSI symbol for a NAND gate

Figure 7: ANSI symbol for a NOR gate

- What’s the truth table for both of those?
- **Sheffer stroke**: ↑ (used for NAND operation)
- **Quince arrow**: ↓ (used for NOR operation)
- NAND gates are the **cheapest** gates to implement in modern hardware.
Complex circuits
Examples

- Convert the following boolean expressions to their corresponding circuits:
  - \((p \land q \land z) \lor (\sim r)\)
  - \((p \lor q) \land q\)

- Now, convert them into circuits that only use NAND gates!
We can often use the axioms of boolean logic to simplify a circuit into one that uses a smaller number of gates.

Then, we can convert that circuit into one that only uses NAND gates!

Pipeline (Important to remember!):

1. Identify boolean expression implemented by the circuit (tip: scan the circuit from output to inputs).
2. Use axioms of boolean algebra to simplify expression in terms of boolean connectives used.
3. Use axioms of boolean algebra to translate the expression into one that only uses (possibly negated) conjunctions ($\neg(p \land \neg q \land z \land \ldots)$).
4. Draw the corresponding circuit.
Examples

Whiteboard...
K-maps

- A much more efficient way to quickly simplify circuits is the **Karnaugh map**, or K-map for short.\(^1\)
- Automatically converts any boolean expression to either a *sum-of-product* form (ORs of ANDs) or *product-of-sums* form (ANDs of ORs).
- We will not analyze them in this course, but you’re directed to the Wikipedia article for information.

\(^1\)Named such after Maurice Karnaugh.