250P: Computer Systems Architecture

Lecture 2: Basics of Digital Design

Anton Burtsev
October, 2019
Digital Design Basics

• Two voltage levels – high and low (1 and 0, true and false)
  Hence, the use of binary arithmetic/logic in all computers

• A transistor is a 3-terminal device that acts as a switch
Logic Blocks

• A logic block has a number of binary inputs and produces a number of binary outputs – the simplest logic block is composed of a few transistors

• A logic block is termed **combinational** if the output is only a function of the inputs

• A logic block is termed **sequential** if the block has some internal memory (state) that also influences the output

• A basic logic block is termed a **gate** (AND, OR, NOT, etc.)

We will only deal with combinational circuits today
Truth Table

• A truth table defines the outputs of a logic block for each set of inputs

• Consider a block with 3 inputs A, B, C and an output E that is true only if *exactly* 2 inputs are true

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</th>
</tr>
</thead>
</table>
Truth Table

- A truth table defines the outputs of a logic block for each set of inputs.

- Consider a block with 3 inputs A, B, C and an output E that is true only if *exactly* 2 inputs are true.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</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>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>0</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>

Can be compressed by only representing cases that have an output of 1.
Boolean Algebra

• Equations involving two values and three primary operators:
  
  ▪ **OR**: symbol + , \( X = A + B \) \( \Rightarrow \) \( X \) is true if at least one of \( A \) or \( B \) is true

  ▪ **AND**: symbol \( \cdot \) , \( X = A \cdot B \) \( \Rightarrow \) \( X \) is true if both \( A \) and \( B \) are true

  ▪ **NOT**: symbol \( \bar{\ } \) , \( X = \bar{A} \) \( \Rightarrow \) \( X \) is the inverted value of \( A \)
Boolean Algebra Rules

• Identity law : \( A + 0 = A \) ; \( A . 1 = A \)

• Zero and One laws : \( A + 1 = 1 \) ; \( A . 0 = 0 \)

• Inverse laws : \( A \cdot \bar{A} = 0 \) ; \( A + \bar{A} = 1 \)

• Commutative laws : \( A + B = B + A \) ; \( A \cdot B = B \cdot A \)

• Associative laws : \( A + (B + C) = (A + B) + C \)
  \( A \cdot (B \cdot C) = (A \cdot B) \cdot C \)

• Distributive laws : \( A \cdot (B + C) = (A \cdot B) + (A \cdot C) \)
  \( A + (B \cdot C) = (A + B) \cdot (A + C) \)
DeMorgan’s Laws

- $A + B = A \cdot B$

- $A \cdot B = A + B$

- Confirm that these are indeed true
Logic for common arithmetic operations
Simple ALU
Pictorial Representations

AND

OR

NOT

What logic function is this?

Source: H&P textbook
Boolean Equation

• Consider the logic block that has an output \( E \) that is true only if exactly two of the three inputs \( A, B, C \) are true

Multiple correct equations:

Two must be true, but all three cannot be true:
\[
E = (A \cdot B) + (B \cdot C) + (A \cdot C) \cdot (A \cdot B \cdot C)
\]

Identify the three cases where it is true:
\[
E = (A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (C \cdot B \cdot \overline{A})
\]
Sum of Products

- Can represent any logic block with the AND, OR, NOT operators
  - Draw the truth table
  - For each true output, represent the corresponding inputs as a product
  - The final equation is a sum of these products

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>E</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>0</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>

\[(A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (C \cdot B \cdot \overline{A})\]

- Can also use “product of sums”
- Any equation can be implemented with an array of ANDs, followed by an array of ORs
NAND and NOR

- NAND: NOT of AND: A nand B = A \cdot B

- NOR: NOT of OR: A nor B = A + B

- NAND and NOR are *universal gates*, i.e., they can be used to construct any complex logical function
Common Logic Blocks – Decoder

Takes in N inputs and activates one of $2^N$ outputs

<table>
<thead>
<tr>
<th>$I_0$</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$O_0$</th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
<th>$O_4$</th>
<th>$O_5$</th>
<th>$O_6$</th>
<th>$O_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td></td>
<td></td>
<td>1 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td></td>
<td></td>
<td>0 1 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td></td>
<td></td>
<td>0 0 1 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td></td>
<td></td>
<td>0 0 0 1 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td></td>
<td></td>
<td>0 0 0 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td></td>
<td></td>
<td>0 0 0 0 0 1 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td></td>
<td></td>
<td>0 0 0 0 0 0 0 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
<td></td>
<td>0 0 0 0 0 0 0 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Common Logic Blocks – Multiplexor

- Multiplexor or selector: one of N inputs is reflected on the output depending on the value of the \( \log_2 N \) selector bits

2-input mux

Source: H&P textbook
Adder Algorithm

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

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

Truth Table for the above operations:

<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></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Adder Algorithm

<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>

Truth Table for the above operations:

Truth Table for the above operations:

Equations:

Sum = Cin \cdot \overline{A} \cdot \overline{B} +
B \cdot \overline{Cin} \cdot \overline{A} +
A \cdot \overline{Cin} \cdot \overline{B} +
A \cdot B \cdot Cin

Cout = A \cdot B \cdot Cin +
A \cdot B \cdot \overline{Cin} +
A \cdot Cin \cdot \overline{B} +
B \cdot Cin \cdot \overline{A}

= A \cdot B +
A \cdot Cin +
B \cdot Cin
Carry Out Logic

Equations:
Sum = Cin \cdot \overline{A} \cdot \overline{B} + B \cdot Cin \cdot A + A \cdot \overline{Cin} \cdot \overline{B} + A \cdot B \cdot Cin

Cout = A \cdot B \cdot Cin + A \cdot B \cdot Cin + A \cdot Cin \cdot B + B \cdot Cin \cdot \overline{A} \\
= A \cdot B + A \cdot Cin + B \cdot Cin

Source: H&P textbook
1-Bit ALU with Add, Or, And

- Multiplexor selects between Add, Or, And operations
32-bit Ripple Carry Adder

1-bit ALUs are connected “in series” with the carry-out of 1 box going into the carry-in of the next box.
Incorporating Subtraction

Must invert bits of B and add a 1
• Include an inverter
• CarryIn for the first bit is 1
• The CarryIn signal (for the first bit) can be the same as the Binvert signal

Source: H&P textbook
Incorporating NOR and NAND

Source: H&P textbook
Incorporating cmp (set bit when less than)

- Perform $a - b$ and check the sign

- New signal (Less) that is zero for ALU boxes 1-31

- The 31st box has a unit to detect overflow and sign – the sign bit serves as the Less signal for the 0th box

Source: H&P textbook
Incorporating cmp (set bit when less than)

- Perform $a - b$ and check the sign

- New signal (Less) that is zero for ALU boxes 1-31

- The 31\textsuperscript{st} box has a unit to detect overflow and sign – the sign bit serves as the Less signal for the 0\textsuperscript{th} box
Incorporating jeq (jump when equal)

- Perform $a - b$ and confirm that the result is all zero’s.
Control Lines

What are the values of the control lines and what operations do they correspond to?
Control Lines

What are the values of the control lines and what operations do they correspond to?

<table>
<thead>
<tr>
<th></th>
<th>Ai</th>
<th>Bn</th>
<th>Op</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>OR</td>
<td>0</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>Add</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Sub</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>SLT</td>
<td>0</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>NOR</td>
<td>1</td>
<td>1</td>
<td>00</td>
</tr>
</tbody>
</table>

Source: H&P textbook
Speed of Ripple Carry

• The carry propagates thru every 1-bit box: each 1-bit box sequentially implements AND and OR – total delay is the time to go through 64 gates!

• We’ve already seen that any logic equation can be expressed as the sum of products – so it should be possible to compute the result by going through only 2 gates!

• Caveat: need many parallel gates and each gate may have a very large number of inputs – it is difficult to efficiently build such large gates, so we’ll find a compromise:
  ▪ moderate number of gates
  ▪ moderate number of inputs to each gate
  ▪ moderate number of sequential gates traversed
Clocks

• A microprocessor is composed of many different circuits that are operating simultaneously – if each circuit $X$ takes inputs at time $T_{I_X}$, takes time $T_{E_X}$ to execute the logic, and produces outputs at time $T_{O_X}$, imagine the complications in coordinating the tasks of every circuit

• A major school of thought (used in most processors built today): all circuits on the chip share a clock signal (a square wave) that tells every circuit when to accept inputs, how much time they have to execute the logic, and when they must produce outputs
Clock Terminology

Rising clock edge

Falling clock edge

Cycle time

4 GHz = clock speed = \( \frac{1}{\text{cycle time}} = \frac{1}{250 \text{ ps}} \).
Sequential Circuits

• Until now, circuits were combinational – when inputs change, the outputs change after a while (time = logic delay thru circuit)

• We want the clock to act like a start and stop signal – a “latch” is a storage device that separates these circuits – it ensures that the inputs to the circuit do not change during a clock cycle
Sequential Circuits

- Sequential circuit: consists of combinational circuit and a storage element

- At the start of the clock cycle, the rising edge causes the “state” storage to store some input values

- This state will not change for an entire cycle (until next rising edge)

- The combinational circuit has some time to accept the value of “state” and “inputs” and produce “outputs”

- Some of the outputs (for example, the value of next “state”) may feed back (but through the latch so they’re only seen in the next cycle)
Sequential circuits
Designing a Latch

- An S-R latch: set-reset latch
  - When Set is high, a 1 is stored
  - When Reset is high, a 0 is stored
  - When both are low, the previous state is preserved (hence, known as a storage or memory element)
  - Both are high – this set of inputs is not allowed

Verify the above behavior!
D Latch

- Incorporates a clock
- The value of the input D signal (data) is stored only when the clock is high – the previous state is preserved when the clock is low

Source: H&P textbook
D Flip Flop

- Terminology:
  Latch: outputs can change any time the clock is high (asserted)
  Flip flop: outputs can change only on a clock edge

- Two D latches in series – ensures that a value is stored only on the falling edge of the clock

Source: H&P textbook
Sequential Circuits

- Until now, circuits were combinational – when inputs change, the outputs change after a while (time = logic delay thru circuit)

- We want the clock to act like a start and stop signal – a “latch” is a storage device that separates these circuits – it ensures that the inputs to the circuit do not change during a clock cycle
Finite State Machines
Finite State Machine

- A sequential circuit is described by a variation of a truth table – a finite state diagram (hence, the circuit is also called a finite state machine)

- Note that state is updated only on a clock edge
State Diagrams

• Each state is shown with a circle, labeled with the state value – the contents of the circle are the outputs

• An arc represents a transition to a different state, with the inputs indicated on the label

This is a state diagram for ___?
3-Bit Counter

- Consider a circuit that stores a number and increments the value on every clock edge – on reaching the largest value, it starts again from 0

Draw the state diagram:
- How many states?
- How many inputs?
3-Bit Counter

- Consider a circuit that stores a number and increments the value on every clock edge – on reaching the largest value, it starts again from 0.

Draw the state diagram:
- How many states?
- How many inputs?