Digital Circuits

CSC-173 Lecture
Tuesday, 11/27/2001

Athanasios E. Papathanasiou
Computer Science Department
University of Rochester

Circuit Design

- Gate:
  - Basic electronic device.
  - Computes a Boolean function.
  - AND, OR, NOT, NAND:
    - Easy to implement.
    - Any number of inputs.
    - Used in practice.

Circuit Design

- Circuit:
  - A combination of gates
  - Output of some gates are the input of others.
  - Has one or more inputs:
    - These are inputs to the gates in the circuit.
  - May have one or more outputs.

3-input NAND:
  - Two 2-input AND.
  - One Inverter.

Combinational & Sequential Circuits

- Combinational:
  - Output is a Boolean function of input values.
  - Are Acyclic:
    - No cycles between inputs of a gate and its outputs.
  - No memory:
    - Cannot remember previous inputs or outputs.
  - Example of use:
    - Decode instructions and perform arithmetic.

Combinational & Sequential Circuits (2)

- Sequential:
  - Output depends on the current input values and the previous sequence of input values.
  - Are Cyclic:
    - Output of a gate feeds its input at some future time.
  - Memory:
    - Remember results of previous operations
    - Use them as inputs.
  - Example of use:
    - Build registers and memory units.

Combinational Circuit: Encoder for a 7-Segment Display

- Goal: Design a circuit...
  - With 10 inputs: i_0, i_1, i_2, ..., i_9.
    - Each one corresponds to the decimal digits (0-9).
  - Lights up the display segments A, B, C, ..., G.
    - As needed to display the digit specified by the input.
    - Total: 7 outputs.

Number 2:
- Input i_2 = 1,
- i_0, i_1, i_3, ..., i_9 = 0.
- Outputs:
  - A=B=D=E=G=1
  - C=F=0
Encoder for a 7-Segment Display (2)

- Boolean expression for the outputs:
  \[ A = i_7 + i_6 + i_5 + i_4 + i_3 + i_2 + i_1 \]
  \[ B = i_6 + i_5 + i_4 + i_3 + i_2 + i_1 + i_0 \]
  \[ C = i_5 + i_4 + i_3 + i_2 + i_1 + i_0 + i_9 \]
  \[ D = i_4 + i_3 + i_2 + i_1 + i_0 + i_8 \]
  \[ E = i_3 + i_2 + i_1 + i_0 \]
  \[ F = i_2 + i_1 + i_0 + i_9 + i_8 \]
  \[ G = i_1 + i_0 + i_9 + i_8 + i_7 + i_6 \]

- Build the circuit with 7 OR gates:
  - One for each segment of the display

Constraints on Circuit Design

- Numerous constraints impact:
  - The speed and cost of a circuit.
- Speed:
  - Every gate in a circuit introduces a small delay.
  - Circuit delay depends on the number of gates between inputs and outputs.

Divide and Conquer Adder

- Already seen Ripple-Carry adder
- Need:
  - Adder with a smaller delay for larger words.
- Solution:
  - Use a divide and conquer strategy.
  - Use two N/2-bit adders and combine results.
  - Left and right halves added in parallel.

Constraints on Circuit Design

- Size limitations:
  - More gates lead to larger circuits.
  - Large circuits are more expensive
    - Higher failure rate.
    - And slower.
  - Signals must propagate from one end to the other.
- Fan-in and Fan-out:
  - Number of inputs and outputs of a gate.
  - Large fan-in makes a gate slower.

Divide and Conquer Adder (2)

- Carry: Not known in advance:
  - How can the adders operate in parallel?
  - Compute to sums for the upper half
    - One assuming there is a carry.
    - One assuming there is NO carry.
  - Use additional circuit to select the correct sum.
Design of an N-adder

- Assume two N-bit operands: \(x_1\ldots x_N\) & \(y_1\ldots y_N\).
- Design N-adder that computes:
  - Sum without carry-in: \(s_1\ldots s_N\).
  - Sum with carry-in: \(t_1\ldots t_N\).
  - The carry-propagate bit, \(p\):
    - It is 1 if there is a carry-out assuming there is carry-in.
  - The carry-generate bit, \(g\):
    - It is 1 if there is a carry-out even if there is NO carry-in.
    - NOTE: if \(g\) is one then \(p\) will be one too (implies \(p\)).
- First, build an 1-bit adder.

### A 1-bit Adder

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>s</th>
<th>t</th>
<th>p</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<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>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
  s &= \overline{x}y + xy \\
  t &= \overline{x}y' + xy' \\
  p &= x + y \\
  g &= xy
\end{align*}

### A 2-bit Adder from 1-bit Adders

``FIX'' Circuit

- Carry-propagate bit: \(p = p_H p_L + g_H\)
  - If there is a carry-in \(p\) is 1 if:
    - Both the low and high order part propagate a carry \((p_H p_L)\).
    - Or: The high order part generates a carry \((g_H)\).
- Carry-generate bit: \(g = g_H + g_L p_H\)
  - If there is NO carry-in \(g\) is 1 if:
    - If the high order part generates a carry \((g_H)\).
    - Or if there is a carry from the low order part and the high part propagate that carry \((g_L p_H)\).

``FIX'' Circuit (2)

- High order sum, NO carry-in:
  - It is:
    - \(s_H = \overline{s}_L p_H + t_H s_L\)
    - \(s_L\) if there is no carry from low order part \((\overline{g}_L)\).
    - \(t_H\) if there is carry from low order part \((g_L)\).
- High order sum, with carry-in:
  - It is:
    - \(t_H = \overline{s}_H p_L + t_L p_L\)
    - \(s_L\) if there is a carry from the low order part.
    - \(s_L\) otherwise.

Sequential Circuits for Memory Elements.

- Memory element:
  - A collection of gates capable of producing its last input as output.
  - They are sequential circuits.
    - Their behavior depends on current and past inputs.
- Flip-flop:
  - Typical flip-flop:
    - Takes two inputs (load and data-in).
    - Produces one output (data-out).
**Flip-Flops**

- **Load==0:**
  - The circuit produces the stored value as output.

- **Load==1:**
  - The circuit stores the value `data-in` and
  - Produces it as output.

**Flip-Flop Circuit**

- **Load=1⇒A1=0⇒data-out=A2=data-in.**
- **Load=0⇒A2=0⇒data-out=A1**
  - Which is the previously stored value.