On Thursday the DNS Root zone was signed by the Internet Systems Consortium (ISC). This is a critical step in the deployment of DNSSEC. This is also a political issue! What is DNSSEC and why does the root need to be signed? Read on.

http://tinyurl.com/27nrtrf
In review…

- ISA is very important abstraction layer
  - Contract between HW and SW
- Clocks control pulse of our circuits
- Voltages are analog, quantized to 0/1
- Circuit delays are fact of life
- Two types of circuits:
  - Stateless Combinational Logic (&, I, ~)
  - State circuits (e.g., registers)
- State elements are used to:
  - Build memories
  - Control the flow of information between other state elements and combinational logic
- D-flip-flops used to build registers
- Clocks tell us when D-flip-flops change
  - Setup and Hold times important
Maximum Clock Frequency

- What is the maximum frequency of this circuit?

Max Delay = Setup Time + CLK-to-Q Delay + CL Delay

Hint... Frequency = 1/Period
Maximum Clock Frequency

How does the clock period relate to the Max Delay?

If the clock period is shorter than the maximum circuit delay, the circuit will not function correctly!

Here the clock period is longer than the max delay. This is wasteful, as the clock period can be shortened.

Now the clock period is minimized. Minimum period means maximum frequency!
Pipelining to improve performance (1/2)

Extra Register are often added to help speed up the clock rate.

Note: delay of 1 clock cycle from input to output.
Clock period limited by propagation delay of adder/shifter.
Pipelining to improve performance (2/2)

- Insertion of register allows higher clock frequency.
- More outputs per second.

Timing...

Tradeoff: It now takes 2 clock cycles to produce a result!
Recap of Timing Terms

- **Clock (CLK)** - steady square wave that synchronizes system
- **Setup Time** - when the input must be stable before the rising edge of the CLK
- **Hold Time** - when the input must be stable after the rising edge of the CLK
- “**CLK-to-Q**” Delay - how long it takes the output to change, measured from the rising edge
- **Flip-flop** - one bit of state that samples every rising edge of the CLK
- **Register** - several bits of state that samples on rising edge of CLK or on LOAD
Finite State Machines (FSM) Introduction

- You have seen FSMs in other classes. (Have you?)

- Same basic idea.

- We represent the functionality of a FSM with a “state transition diagram”.

- With combinational logic and registers, any FSM can be implemented in hardware.
Finite State Machine Example: 3 ones…

 FSM to detect the occurrence of 3 consecutive 1’s in the input.

Draw the FSM…

Assume state transitions are controlled by the clock: on each clock cycle the machine checks the inputs and moves to a new state and produces a new output…
Hardware Implementation of FSM

… Therefore a register is needed to hold the representation of which state the machine is in. Use a unique bit pattern for each state.

Combinational logic circuit is used to implement a function maps from present state and input to next state and output.
Hardware for FSM: Combinational Logic

In the reference slides we discuss the detailed implementation, but for now can look at its functional specification, truth table form.

Truth table...

<table>
<thead>
<tr>
<th>PS</th>
<th>Input</th>
<th>NS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>00</td>
<td>1</td>
</tr>
</tbody>
</table>
General Model for Synchronous Systems

- Collection of CL blocks separated by registers.
- Registers may be back-to-back and CL blocks may be back-to-back.
- Feedback is optional.
- Clock signal(s) connects only to clock input of registers.
Peer Instruction

A. The period of a **usable synchronous circuit** is greater than the CLK-to-Q delay

B. You can build a FSM to signal when an equal number of 0s and 1s has appeared in the input.
Administrivia

• HW 6 due date has been moved to Wednesday at midnight
  • HW 7 moved back 1 day accordingly
  • HW 8 not moved!

• Midterm being handed back after lecture. Stick around.

• Discussion today will be covering common midterm problems. You should attend!

• Grades and solutions will be online today.

• Deadline for re-grades is next Monday. We will re-grade the entire test. This means grade could go down.

• Goal is to have HW1-5, Proj 1, all graded by next Monday.
  • We’ll do our best to meet the goal.
Combinational Logic

• FSMs had states and transitions
• How to we get from one state to the next?
• Answer: Combinational Logic
Truth Tables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(0,0,0,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(0,0,0,1)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(0,0,1,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(0,0,1,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(0,1,0,0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(0,1,0,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(0,1,1,0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(0,1,1,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(1,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(1,0,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(1,0,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(1,0,1,1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(1,1,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(1,1,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(1,1,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(1,1,1,1)</td>
</tr>
</tbody>
</table>
TT Example #1: 1 iff one (not both) a,b=1

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>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>0</td>
</tr>
</tbody>
</table>
TT Example #2: 2-bit adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>(a_0)</td>
<td>(b_1)</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

How Many Rows?
TT Example #3: 32-bit unsigned adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 ... 0</td>
<td>000 ... 0</td>
<td>000 ... 0</td>
</tr>
<tr>
<td>000 ... 0</td>
<td>000 ... 1</td>
<td>000 ... 01</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>111 ... 1</td>
<td>111 ... 1</td>
<td>111 ... 10</td>
</tr>
</tbody>
</table>

How Many Rows?
Logic Gates (1/2)

AND
\[ a \quad b \quad c \]
\[
\begin{array}{cc|c}
ab & c \\
00 & 0 \\
01 & 0 \\
10 & 0 \\
11 & 1 \\
\end{array}
\]

OR
\[ a \quad b \quad c \]
\[
\begin{array}{cc|c}
ab & c \\
00 & 0 \\
01 & 1 \\
10 & 1 \\
11 & 1 \\
\end{array}
\]

NOT
\[ a \quad b \]
\[
\begin{array}{c|c}
a & b \\
0 & 1 \\
1 & 0 \\
\end{array}
\]
And vs. Or review – Dan’s mnemonic

AND Gate

Symbol

A
B

AN

C

Definition

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Logic Gates (2/2)

<table>
<thead>
<tr>
<th>XOR</th>
<th>( ab )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAND</th>
<th>( ab )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOR</th>
<th>( ab )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>
2-input gates extend to n-inputs

- N-input XOR is the only one which isn’t so obvious
- It’s simple: XOR is a 1 iff the # of 1s at its input is odd

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>y</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>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Truth Table $\Rightarrow$ Gates (e.g., majority circ.)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>y</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>1</td>
</tr>
</tbody>
</table>
Boolean Algebra

• George Boole, 19\textsuperscript{th} Century mathematician

• Developed a mathematical system (algebra) involving logic
  • later known as “Boolean Algebra”

• Primitive functions: AND, OR and NOT

• The power of BA is there’s a one-to-one correspondence between circuits made up of AND, OR and NOT gates and equations in BA

$+$ means OR, $\cdot$ means AND, $\overline{x}$ means NOT
Boolean Algebra (e.g., for majority fun.)

\[ y = (a \cdot b) + (b \cdot c) + (a \cdot c) \]
\[ y = a \cdot b + b \cdot c + a \cdot c \]
\[ y = ab + bc + ac \]
BA: Circuit & Algebraic Simplification

original circuit

\[ y = ((ab) + a) + c \]

\[ = ab + a + c \]

\[ = a(b + 1) + c \]

\[ = a(1) + c \]

\[ = a + c \]

simplified circuit

BA also great for circuit verification

Circ X = Circ Y?
use BA to prove!
Laws of Boolean Algebra

\[ x \cdot \overline{x} = 0 \quad \text{complementarity} \]
\[ x \cdot 0 = 0 \quad \text{laws of 0’s and 1’s} \]
\[ x \cdot 1 = x \quad \text{identities} \]
\[ x \cdot x = x \quad \text{idempotent law} \]
\[ x \cdot y = y \cdot x \quad \text{commutativity} \]
\[ (xy)z = x(yz) \quad \text{associativity} \]
\[ x(y + z) = xy + xz \quad \text{distribution} \]
\[ xy + x = x \quad \text{uniting theorem} \]
\[ \overline{xy} + x = x + y \quad \text{uniting theorem v.2} \]
\[ x \cdot \overline{y} = \overline{x + y} \quad \text{DeMorgan’s Law} \]
\[ x + \overline{x} = 1 \]
\[ x + 1 = 1 \]
\[ x + 0 = x \]
Boolean Algebraic Simplification Example

\[ y = ab + a + c \]
\[ = a(b + 1) + c \quad \text{distribution, identity} \]
\[ = a(1) + c \quad \text{law of 1's} \]
\[ = a + c \quad \text{identity} \]
### Canonical forms (1/2)

#### Sum-of-products (ORs of ANDs)

<table>
<thead>
<tr>
<th>$\overline{a} \cdot \overline{b} \cdot \overline{c}$</th>
<th>$abc$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Canonical forms (2/2)

\[
y = \overline{a} \overline{b} \overline{c} + \overline{a} \overline{b} \overline{c} + \overline{a} \overline{b} \overline{c} + \overline{a} \overline{b} \overline{c} \\
= \overline{a} \overline{b} (\overline{c} + c) + a \overline{c} (\overline{b} + b) \quad \text{distribution} \\
= \overline{a} \overline{b} (1) + a \overline{c} (1) \quad \text{complementarity} \\
= \overline{a} \overline{b} + a \overline{c} \quad \text{identity}
\]
Peer Instruction

1. \((a+b) \cdot (\overline{a}+b) = b\)

2. You can use NOR(s) with clever wiring to simulate AND, OR, & NOT

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>FF</td>
</tr>
<tr>
<td>B</td>
<td>FT</td>
</tr>
<tr>
<td>C</td>
<td>TF</td>
</tr>
<tr>
<td>D</td>
<td>TT</td>
</tr>
</tbody>
</table>
“And In conclusion…”

- Max clock frequency is calculated based on max circuit delay
- We pipeline long-delay CL for faster clock
- Finite State Machines extremely useful
  - You’ll see them again 150, 152, 164, 172…
- Boolean algebra
  - Sum-of-products
- Use this table to transform from one format to another
You ARE responsible for the material on these slides (they’re just taken from the reading anyway) ; we’ve moved them to the end and off-stage to give more breathing room to lecture!
TT Example #4: 3-input majority circuit

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>y</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>1</td>
</tr>
</tbody>
</table>
Truth Table $\Rightarrow$ Gates (e.g., FSM circ.)

<table>
<thead>
<tr>
<th>PS</th>
<th>Input</th>
<th>NS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>00</td>
<td>1</td>
</tr>
</tbody>
</table>

or equivalently…
Boolean Algebra (e.g., for FSM)

<table>
<thead>
<tr>
<th>PS</th>
<th>Input</th>
<th>NS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>00</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ y = PS_1 \cdot PS_0 \cdot INPUT \]