ECE 550:
Fundamentals of Computer Systems and Engineering

Finite State Machines
Admin

- Homework #1 Due Friday!
  - One submission per group please
  - Clearly identify group members
Last time...

- Who can remind us what we did last time?
Last time...

- Who can remind us what we did last time?
  - Storage and Clocking
    - Latches
    - Flip-flops
  - Level vs Edge triggered
Finite Storage = Finite States

- Computers have finite storage (inside processor):
  - Design in fixed number of DFFs
  - Result: finite number of states ($N$ bits $\Rightarrow 2^N$ states)

- Useful to talk about finite state machines
  - Ubiquitous in processor design
    - Basically how the processor works out many multi-step processes
Finite State Machines

- Output = f(Input, Current State)
- New State = f(Input, Current State)

Example: Traffic Light

- Input: NS_turn, EW_turn
- Outputs: which lights are on
  - NS_green
  - NS_g_arrow
  - NS_yellow
  - NS_y_arrow
  - NS_red
  - EW_green
  - ...
State Diagrams

- Can draw state machine as a diagram
  - Circles for states
  - Arrows for transitions (possibly with a choice based on inputs)
State Diagrams

- Full diagram for our traffic light
  - Note start state: NSg0
- Note: real traffic lights have more states
  - Longer greens relative to yellows. All red in before next green...
State Diagrams

- Could make it smarter/fancier with more inputs
  - E.g., stay green unless opposing traffic present
  - Perfectly fine to have self-loops (stay in same state)
Transition function

- State diagrams describe the transition function pictorially.
  - next_state = f (inputs, current_state)
  - Easy to translate into VHDL:

```vhdl
state_d <= EWg0 when state_q = NSy and not NS_turn else NSa when state_q = NSy and NS_turn else Nsay when state_q = Nsa else EWg0 when state_q = NSAy else ...;
```
Transition function

Can define these as constants

- State diagrams describe the transition function pictorially
  - next_state = f (inputs, current_state)
  - Easy to translate into VHDL:

```vhdl
state_d <= EWg0 when state_q = NSy and not NS_turn else NSa when state_q = NSy and NS_turn else Nsay when state_q = Nsa else EWg0 when state_q = NSAy else ...
```

...
Transition function

Why state_d and state_q?

- State diagrams describe transition function pictorially
  - next_state = f (inputs, current_state)
  - Easy to translate into VHDL:

```vhdl
state_d <= EWg0 when state_q = NSy and not NS_turn else
          NSa  when state_q = NSy and NS_turn else
          Nsay when state_q = Nsa else
          EWg0 when state_q = NSAy else ... .
```

ECE 550 (Hilton): Finite State Machines
Transition function

Why state_d and state_q?  

Will latch state in DFFs from one cycle to next.

state_d = next one  
state_q = current one

• State diagrams describes transition function pictorially
  • next_state = f (inputs, current_state)
  • Easy to translate into VHDL:

\[
\begin{align*}
\text{state_d} & \leq \text{EWg0\ when\ state_q = NSy and not NS_turn} \ \text{else} \\
& \text{NSa\ when\ state_q = NSy and NS_turn} \ \text{else} \\
& \text{NSay\ when\ state_q = Nsa} \ \text{else} \\
& \text{EWg0\ when\ state_q = NSAy} \ \text{else} \ \ldots
\end{align*}
\]
Large number of similar states

- Sometimes have large # of similar states
  - E.g., instead of NSg0, NSg1, NSg2, may have 0 to 200
    - Example: VGA controller....
  - Painful:
    - Actually have NSg0, ...NSg200 states
  - Easier
    - NSg state, and a counter.
    - Transition to next state on counter\_q = 200
Output function

- Also need an output function:
  - For each output signal, compute as function of inputs and state
  - (or maybe just state, as in traffic lights)

<table>
<thead>
<tr>
<th>State</th>
<th>ns_gc</th>
<th>ns_ga</th>
<th>ns_yc</th>
<th>ns_ya</th>
<th>ns_r</th>
<th>ew_gc</th>
<th>ew_ga</th>
<th>ew_ya</th>
<th>ew_ya</th>
<th>ew_r</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSg_</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NSy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSay</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EWg_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EWy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EWa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EWay</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Output function

- Also need an output function:
  - For each output signal, compute as function of inputs and state
  - (or maybe just state, as in traffic lights)

<table>
<thead>
<tr>
<th>State</th>
<th>ns_gc</th>
<th>ns_ga</th>
<th>ns_yc</th>
<th>ns_ya</th>
<th>ns_r</th>
<th>ew_gc</th>
<th>ew_ga</th>
<th>ew_yc</th>
<th>ew_ya</th>
<th>ew_r</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSg_</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NSy</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NSa</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NSay</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EWg_</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EWy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EWa</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EWay</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Hardware implementation

<table>
<thead>
<tr>
<th>State</th>
<th>ns_gc</th>
<th>ns_ga</th>
<th>ns_ya</th>
<th>ns_r</th>
<th>ew_gc</th>
<th>ew_ga</th>
<th>ew_ya</th>
<th>ew_r</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSg_</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NSy</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NSa</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NSay</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EWg_</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EWy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>EWa</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>EWay</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Hardware implementation option 1:**
  - Logic from the truth table
  - (VHDL pretty straight forward)
Hardware implementation: ROM

- Can also use ROM
  - Read Only Memory
  - Address goes into decoder
  - One hot word line goes into memory array
  - Data comes out on bit lines
- More details soon (when we do RAMs)
Take a minute to draw an FSM for a combination lock

- Combination: 12345

“So the combination is... one, two, three, four, five? That's the stupidest combination I've ever heard in my life! That's the kind of thing an idiot would have on his luggage!” —Dark Helmet (Spaceballs, the movie)

- Inputs:
  - One hot is_0, is_1, is_2, ...

- Outputs:
  - Unlock

- Draw transitions as state diagram, note which states have unlock on.
  - Feel free to abbreviate “all other cases” by leaving arrow label blank
Combination Lock

- is_1 always takes us to S1
- Correct input moves us “right”
- Other: back to start
- S5 unlocks
Hwk2 will have FSM to implement in VHDL

- VGA controller
- Scan row from left to right, sending out data pixel by pixel
  - One pixel per cycle
Hwk2 will have FSM to implement in VHDL
- VGA controller
- Scan row from left to right, sending out data pixel by pixel
  - One pixel per cycle
- Then period of black (all 0 pixel) with some control signals
  - “Past” the right edge
- Actually three different states here.
VGA controller: FSM

- Hwk2 will have FSM to implement in VHDL
  - VGA controller
  - Scan row from left to right, sending out data pixel by pixel
    - One pixel per cycle
  - Then period of black (all 0 pixel) with some control signals
    - “Past” the right edge
  - Then restart on next row
VGA controller: FSM

• VGA controller
  • After last row, similar behavior to horizontal
VGA controller: FSM

- VGA controller
  - After last row, similar behavior to horizontal
  - Trace blank rows
    - All black, goes through same horizontal states as real rows
  - Also three different states.
VGA controller: FSM

- VGA controller
  - After last row, similar behavior to horizontal
  - Trace blank rows
    - All black, goes through same horizontal states as real rows
  - Also three different states.
  - Then reset to top left corner
VGA on hwk2

- More details in hwk2 assignment
  - Can think of as one big state machine
  - Or two working together (one horizontal, one vertical)
Division: math with an FSM

- We have talked about add, sub
  - Pretty easy math to implement in hardware
- What about divide?
  - Much more complicated
  - Multi-step process
  - Well suited to FSM
Division: Binary

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
Division: Binary

• Binary long division similar to decimal
  • But a little simpler, because it goes in 1 or 0 times
Division: Binary

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
Division: Binary

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
Division: Binary

001
11 101101

10 101 - 11 = 10

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
Division: Binary

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
Binary long division similar to decimal
  • But a little simpler, because it goes in 1 or 0 times
Division: Binary

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
Division: Binary

00111
11 101101

1 100 – 11 = 1

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
Division: Binary

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
Division: Binary

\[
\begin{array}{c|c}
 & 001111 \\
\hline
11 & 101101 \\
\end{array}
\]

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

\[
\begin{array}{c}
0 \\
11 - 11 = 0 \\
\end{array}
\]
Division: Binary

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

45 / 3 = 15 remainder 0

Remainder = 0
Done
Division FSM/Circuit

- 32 bit division: 32 states (5 bits)
  - Decrement state # each cycle (count down which bit)
• Use State # to pick out which bit of Dividend
- Shift remainder left 1, concatenate dividend bit at right
Division FSM/Circuit

- Check if divisor is < result... used for two things
  - Mux selector on remainder_d
  - Lowest bit of answer_d
Division FSM/Circuit

- For answer, shift old answer $<<1$, concatenate in $<$ result
• For remainder, pick from two things (based on < result)
  • Result of shifting old remainder and concatenating dividend bit
  • That minus the divisor
Summary

• Finite State Machine
  • Finite states (encoded in some way: binary nums, one-hot...)
  • Transition function: (state * inputs) -> state
    • Helpful to draw as diagram
  • Output function: (state * inputs) -> outputs

• Examples:
  • Traffic Light
  • VGA controller (hwk2)
  • Division
    • Plus learned division algorithm