Overview

- Last lectures
  - Finite-state machines
    - Example: A sequence detector FSM
    - Example: A vending machine FSM
- Today
  - A bigger example
  - Ant-brain FSM

Ant in a maze

- Electronic ant, electronic maze
- Design the ant
Example: ant brain (Ward, MIT)
- Sensors: L and R antennae, 1 if in touching wall
- Actuators: F - forward step, TL/TR - turn left/right slightly
- Goal: find way out of maze
- Strategy: keep the wall on the right

Example: ant brain (special case 1)
- Left (L) Antenna touching the wall

Example: ant brain (special case 2)
- Ant Lost

Example: ant brain (special case 2)
- Ant Lost (another example)
**Goal: Find a way out of maze**

- Sensors on L and R antennae
  - Sensor = “1” if touching wall; “0” if not touching wall
  - \( L'R' \equiv \) no wall
  - \( L'R \equiv \) wall on right
  - \( LR' \equiv \) wall on left
  - \( LR \equiv \) wall in front
  - \( *** \equiv \) exit

- Movement:
  - \( F \equiv \) forward one step
  - \( TL \equiv \) turn left 90 degrees
  - \( TR \equiv \) turn right 90 degrees

**Ant behavior**

- A: Following wall, touching
  - Go forward, turning left slightly

- B: Following wall, not touching
  - Go forward, turning right slightly

- C: Break in wall
  - Go forward, turning right slightly

- D: Hit wall again
  - Back to state A

- E: Wall in front
  - Turn left until...

- F: ...we are here, same as state B

- G: Turn left until...

- LOST: Forward until we touch something

**Notes & strategy**

- Notes
  - Maze has no islands
  - Corridors are wider than ant
  - Don’t worry about startup
  - Assume a Moore machine
  - Assume D flip-flops

- Strategy
  - Partition your design into datapath and control
  - Keep the wall on the right

**The ant’s behavior**

- S0: Lost
  - Go forward

- S1: Right antenna touching
  - Go forward

- S2: Break in wall
  - Turn right

- S3: Left antenna touching
  - Turn left

- S0: Lost
  - Go forward
The maze

- Virtual maze
  - 128 × 128 grid
  - Stored in memory
  - 16384 8-bit words
- YX is maze addresses
  - X is the ant’s horizontal position (7 bits)
  - Y is the ant’s vertical position (7 bits)
- Each memory location says
  - 00000001 ≡ No wall
  - 00000010 ≡ North wall
  - 00000100 ≡ West wall
  - 00001000 ≡ South wall
  - 00010000 ≡ East wall
  - 00100000 ≡ Exit

Can have multiple walls
Example: 00001100
⇒ Walls on South and East

Where do you start???

Don’t look ahead

What you need

- An FSM for the ant
  - 3 outputs
    - Go forward
    - Turn left
    - Turn right
- Two 7-bit registers for X and Y
  - With preload, increment, decrement
- A register to hold the ant’s heading
- Logic to convert memory data to antennae info

Recommendations

- 7-bit counters for X, Y
  - Move horizontally: Increment or decrement X
  - Move vertically: Increment or decrement Y
- Shift register for heading
  - N: 0001
  - W: 0010
  - S: 0100
  - E: 1000
  - Rotate right when ant turns right
  - Rotate left when ant turns left
- Combinational logic for antennae decoder
Partition the design

Design the ant-brain FSM

1. State diagram and state-transition table
2. State minimization
3. State assignment (or state encoding)
4. Minimize next-state logic
5. Implement the design

Step 1a: State diagram

Step 1b: State-transition table
Step 2: State minimization

- Two states are equivalent if they cannot be distinguished at the outputs of the FSM
  - The outputs are the same for any input sequence
- Two conditions for two states to be equivalent
  1) Outputs must be the same in both states
  2) Machine must transition to equivalent states for all inputs

- Any equivalent states in our state diagram?

Step 3: State encoding

<table>
<thead>
<tr>
<th>Exit</th>
<th>X</th>
<th>Y</th>
<th>L</th>
<th>R</th>
<th>X*</th>
<th>Y*</th>
<th>F</th>
<th>TL</th>
<th>TR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Step 4: Minimize the logic

Step 5: Implement the design
Antennae logic

- Each memory location says
  - 00000001 ≡ No wall
  - 00000010 ≡ North wall (NW)
  - 00000100 ≡ West wall (WW)
  - 00001000 ≡ South wall (SW)
  - 00010000 ≡ East wall (EW)
  - 00100000 ≡ Exit

- The ant can be heading
  - N: 0001
  - W: 0010
  - S: 0100
  - E: 1000

Logic for right antennae
\[
R = \text{NW}(N + W) + \text{WW}(W + S) + \text{SW}(S + E) + \text{EW}(E + N)
\]

Logic for left antennae
\[
L = \text{NW}(N + E) + \text{WW}(W + N) + \text{SW}(S + W) + \text{EW}(E + S)
\]

Gate count:
- 4 2-input ORs
- 8 2-input ANDs
- 2 4-input ORs

What we left out...

- Crumbs in cell
  - Ant eats crumbs in every cell it visits
  - Writes crumb file back to SRAM
  - Read crumb file, for future display on monitor

- Need a memory controller
  - A state machine to talk to the SRAM

- Need to deal with startup, exit states