Parity Checker Example

A string of bits has “even parity” if the number of 1’s in the string is even.

• Design a circuit that accepts a bit-serial stream of bits and outputs a 0 if the parity thus far is even and outputs a 1 if odd:

- Can you guess a circuit that performs this function?
Formal Design Process

- "State Transition Diagram"
  - circuit is in one of two states.
  - transition on each cycle with each new input over exactly one arc (edge).
  - Output depends on which state the circuit is in.

Formal Design Process

- State Transition Table:

<table>
<thead>
<tr>
<th>present state</th>
<th>OUT</th>
<th>IN</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVEN</td>
<td>0</td>
<td>0</td>
<td>EVEN</td>
</tr>
<tr>
<td>EVEN</td>
<td>0</td>
<td>1</td>
<td>ODD</td>
</tr>
<tr>
<td>ODD</td>
<td>1</td>
<td>0</td>
<td>ODD</td>
</tr>
<tr>
<td>ODD</td>
<td>1</td>
<td>1</td>
<td>EVEN</td>
</tr>
</tbody>
</table>

- Invent a code to represent states:
  Let 0 = EVEN state, 1 = ODD state

<table>
<thead>
<tr>
<th>present state (ps)</th>
<th>OUT</th>
<th>IN</th>
<th>next state (ns)</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>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>

Derive logic equations from table (how?):
OUT = PS
NS = PS xor IN
Formal Design Process

Logic equations from table:
\[
\begin{align*}
\text{OUT} &= \text{PS} \\
\text{NS} &= \text{PS} \oplus \text{IN}
\end{align*}
\]

- Circuit Diagram:

```
\begin{itemize}
  \item xor gate for ns calculation
  \item DFF to hold present state
  \item no logic needed for output
\end{itemize}
```

- Review of Design Steps:
  1. Circuit functional specification
  2. State Transition Diagram
  3. Symbolic State Transition Table
  4. Encoded State Transition Table
  5. Derive Logic Equations
  6. Circuit Diagram

\[
\begin{align*}
\text{FFs for state} \\
\text{CL for NS and OUT}
\end{align*}
\]

Finite State Machines (FSMs)

- Type of sequential circuit:
  - output depends on present and past inputs
    - effect of past inputs is represented by the current state

- Behavior is represented by State Transition Diagram:
  - traverse one edge per cycle.
FSM Implementation

- FFs form state register
- number of states \( \leq 2^{\text{number of flip-flops}} \)
- CL implements calculates next state and output

Combination Lock Example

- Used to allow entry to a locked room:
  2-bit serial combination. Example 01,11:
  1. Set switches to 01, press ENTER
  2. Set switches to 11, press ENTER
  3. OPEN is asserted (OPEN=1).
     If wrong code, ERROR is asserted (after second combo word entry).
     Press Reset at anytime to try again.
Combination Lock Example

• Used to allow entry to a locked room:
  2-bit serial combination. Example 01,11:
  1. Set switches to 01, press ENTER
  2. Set switches to 11, press ENTER
  3. OPEN is asserted (OPEN=1).
     If wrong code, ERROR is asserted (after second combo word entry).
     Press Reset at anytime to try again.
Combinational Lock STD

Symbolic State Transition Table

<table>
<thead>
<tr>
<th>RESET</th>
<th>ENTER</th>
<th>COM1</th>
<th>COM2</th>
<th>Preset State</th>
<th>Next State</th>
<th>OPEN</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>START</td>
<td>START</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>START</td>
<td>BAD1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td>START</td>
<td>OK1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>OK1</td>
<td>OK1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>*</td>
<td>0</td>
<td>OK1</td>
<td>BAD2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>OK1</td>
<td>OK2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>OK2</td>
<td>OK2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>BAD1</td>
<td>BAD1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>BAD1</td>
<td>BAD2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>BAD2</td>
<td>BAD2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>START</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Decoder logic for checking combination (01,11):

Decoder logic for checking combination (01,11):
Encoded ST Table

- Assign states:
  \[
  \text{START}=000, \text{OK1}=001, \text{OK2}=011
  \]
  \[
  \text{BAD1}=100, \text{BAD2}=101
  \]
- Omit reset. Assume that primitive flip-flops has reset input.
- Rows not shown have don't cares in output. Correspond to invalid PS values.

---

FSM Implementation Notes

- General FSM form:
  \[
  \begin{array}{c}
  \text{inputs} \\
  \hline
  \text{present state} \\
  \hline
  \text{next state} \\
  \hline
  \text{outputs}
  \end{array}
  \]

- All examples so far generate output based only on the present state:

- Commonly name **Moore Machine**
State Encoding

- **One-hot encoding of states.**
- One FF per state.

Ex: 3 States.

- **STATE1:** 001
- **STATE2:** 010
- **STATE3:** 100

- FF1
  - FF2
  - FF3

- In general: 
  - \# of possible state = 2\(^{\# \text{of FFs}}\)

- However, often more than \(\log_2(\# \text{of states})\) FFs are used, to simplify logic at the cost of more FFs.

- Extreme example is one-hot state encoding.

---

One-hot encoded FSM

- **Even Parity Checker Circuit:**

- **In General:**
  - FFs must be initialized for correct operation (only one 1)

  - To other state FF logic and/or output
One-hot encoded combination lock