**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:

```
<table>
<thead>
<tr>
<th>bit stream</th>
<th>IN</th>
<th>Parity Checker</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
```

example: 0 0 1 1 1 0 1
- even even odd even odd odd even

CLK
```
<table>
<thead>
<tr>
<th>time</th>
</tr>
</thead>
</table>
```

IN
```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
```

OUT
```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
```

- 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.

### Parity Checker

| IN   | OUT | bit stream | even | odd
|------|-----|------------|------|-----
| 0    | 0   | 0          | even | even
| 0    | 1   | 1          | odd  | odd
| 1    | 0   | 1          | even | odd
| 1    | 1   | 1          | odd  | even

### 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>

### Derive Logic Equations

- **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:**
  - XOR gate for ns calculation
  - DFF to hold present state
  - no logic needed for output

- **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

Finite State Machines (FSMs)

- FSM circuits are a 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 clock cycle.
**FSM Implementation**

- FFs form *state register*
- number of states \( \leq 2^{\text{number of flip-flops}} \)
- CL (combinational logic) calculates next state and output
- Remember: The FSM follows exactly one edge per cycle.

---

**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.
Announcements

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

![Combinational Lock STD Diagram]

### 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>0</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>0</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 Diagram]

Spring 2003 EECS150 - Lect07-FSM1 Page 11
### Encoded ST Table

- Assign states:
  - **START**=000, **OK1**=001, **OK2**=011
  - **BAD1**=100, **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.

- What are the output functions for OPEN and ERROR?

### FSM Implementation Notes

- General FSM form:

  ![ FSM Diagram ]

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

- Commonly name **Moore Machine**
  (If output functions include both present state and input then called a **Mealy Machine**)

---

*Spring 2003 EECS150 - Lec07-FSM1 Page 13*
State Encoding

- In general:
  \[ \text{# of possible states} = 2^\text{# of FFs} \]

- 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 encoding of states.
- One FF per state.

**Ex:** 3 States.

STATE1: 001
STATE2: 010
STATE3: 100

- Simple design procedure.
- Circuit matches state transition diagram.
- Can be costly for FSMs with large number of states

One-hot encoded FSM

- Even Parity Checker Circuit:
  - In General:
    - FFs must be initialized for correct operation (only one 1)
One-hot encoded combination lock