LECTURE 6: State machines
VHDL Component, Entity, and Architecture

- for-generate | if generate
  - Component Instance
    - Component Declaration
      - Entity
        - Architecture
          - Concurrent Boolean Equations
          - Concurrent With-Select-When
          - When-Else
          - Other Concurrent Components
VHDL Components

Component Declaration

COMPONENT component_entity_name
  [ GENERIC ( { identifier: type [:= initial_value]; } ) ]
  [ PORT       ( { identifier: mode type; } ) ]
END;

Component Instance

identifier : component_entity_name
  [ GENERIC MAP ( identifier { ,identifier } ) ]
  [ PORT MAP ( identifier { ,identifier } ) ]

mode := IN | OUT | INOUT

CWRU EECS 318
VHDL Concurrent Statements

Boolean Equations
relation ::= relation LOGIC relation | NOT relation | ( relation )

LOGIC ::= AND | OR | XOR | NAND | NOR | XNOR

Example: \( y \leq \text{NOT} \left( \text{NOT} \left( a \right) \text{AND} \text{NOT} \left( b \right) \right) \)

Multiplexor case statement
WITH select_signal SELECT
signal <= signal_value_1 WHEN select_compare_1,
    . . .
    WHEN select_compare_n;

Example: 2 to 1 multiplexor
WITH s SELECT y <= a WHEN \text{‘0’}, b WHEN OTHERS;
VHDL Concurrent Statements

Conditional signal assignment

```vhdl
signal <= signal_value_1 WHEN condition_1 ELSE
       . . .
       signal_value_n WHEN condition_n; ELSE
       signal_value_{n+1}
```

Example: Priority Encoder
```
y <= a WHEN s='0' ELSE b;
```
SR Flip-Flop (Latch)

### NOR

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>$Q_{n+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$Q_n$</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>U</td>
</tr>
</tbody>
</table>

$Q \ <= R \ \text{NOR} \ \overline{NQ}$;

$\overline{NQ} \ <= S \ \text{NOR} \ Q$;

### NAND

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>$Q_{n+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>U</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>$Q_n$</td>
</tr>
</tbody>
</table>

$Q \ <= R \ \text{NAND} \ \overline{NQ}$;

$\overline{NQ} \ <= S \ \text{NAND} \ Q$;
SR Flip-Flop (Latch)

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>$Q_{n+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>U</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>$Q_n$</td>
</tr>
</tbody>
</table>

### NAND Inputs

- **R**: 1, 0, 1, 1
- **S**: 1, 1, 1, 1

### Truth Table

<table>
<thead>
<tr>
<th>t</th>
<th>0</th>
<th>5ns</th>
<th>10ns</th>
<th>15ns</th>
<th>20ns</th>
<th>25ns</th>
<th>30ns</th>
<th>35ns</th>
<th>40ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Example:

- $R \leq '1'$, '0' after 10ns, '1' after 30ns; $S \leq '1'$;

### Timing Diagram

- $R(t)$, $Q(t)$, $\overline{Q(t)}$ with delay 5ns.
- $Q(t+5\text{ns})$ with delay 5ns.

### Output

- $Q(t+5\text{ns})$ with delay 5ns.

---

CWRU EECS 318
Gated-Clock SR Flip-Flop (Latch Enable)

\[ Q \leftarrow (S \text{ NAND } LE) \text{ NAND } NQ; \]
\[ NQ \leftarrow (R \text{ NAND } LE) \text{ NAND } Q; \]

Synchronous:
Set and Reset
Asynchronous:
Preset and Clear

Latches require that during the gated-clock the data must also be stable (i.e. S and R) at the same time.

Suppose each gate was 5ns: how long does the clock have to be enabled to latch the data?

Answer: 15ns
Rising-Edge Flip-flop

D-Type Flip Flop

\[ Q \leq D \text{ when } Clk \text{ changes from '0' to } \]

Dashed vertical lines indicate when an input to the flip flop has changed.

\[ Clk \]

\[ D \]

\[ Q \]

\[ t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \]
Rising-Edge Flip-flop logic diagram

Figure 6.24 Negative edge-triggered D flip-flop when clock is
Synchronous Sequential Circuit

Issues: Specification, design, clocking and timing
Abstraction: Finite State Machine

- A Finite State Machine (FSM) has:
  - $K$ states, $S = \{s_1, s_2, \ldots, s_K\}$, initial state $s_1$
  - $N$ inputs, $I = \{i_1, i_2, \ldots, i_N\}$
  - $M$ outputs, $O = \{o_1, o_2, \ldots, o_M\}$
  - Transition function $T(S, I)$ mapping each current state and input to a next state
  - Output function $O(S)$ mapping each current state to an output

- Given a sequence of inputs the FSM produces a sequence of outputs which is dependent on $s_1$, $T(S, I)$ and $O(S)$
FSM Representations

State Transition Graph

Initial state

State Transition Table

\[ T(S, I) \]

\[ O(S) \]

\[
\begin{array}{ccc}
0 & s_1 & s_2 \\
1 & s_1 & s_3 \\
0 & s_2 & s_1 \\
1 & s_2 & s_2 \\
0 & s_3 & s_1 \\
1 & s_3 & s_3 \\
\end{array}
\]

\[
\begin{array}{c}
s_1 \\
s_2 \\
s_3 \\
\end{array}
\]

\[
\begin{array}{c}
00 \\
10 \\
11 \\
\end{array}
\]

Inputs: 0 1 0
Outputs: 00
Simple Design Example

- Design a FSM that outputs a 1 if and only if the number of 1’s in the input sequence is odd
State Encoding: Choose a unique binary code for each $s_i$ so the combinational logic can be specified

- Choose $s_1 = 0$ and $s_2 = 1$
- Choose $s_1 = 1$ and $s_2 = 0$
Logic Implementations

Choose \( s_1 = 0 \) and \( s_2 = 1 \)

Choose \( s_1 = 1 \) and \( s_2 = 0 \)
Observations

- Number of bits required to encode $K$ states is $\lceil \log_2 K \rceil$

- Encoding states results in combinational logic specifications for $T(S, I)$ and $O(S)$

- Choice of encoding affects complexity of logic implementation
  - How does one find the optimum state encoding?
Coke Machine Example

- Coke costs $.10
- Only nickels and dimes accepted
- FSM inputs:
  - 5: Nickel
  - 10: Dime
  - Coke: Give me a coke
  - Return: Give me my money back

- FSM outputs:
  - Drop: Drop a coke
  - Ret5: Return $.05
  - Ret10: Return $.10
Assumption: At most one input among Coke, 5, 10, and Return is asserted

* represents all unspecified transitions from state

Does this work? _________________
After Return input, any input in the next cycle is ignored!
Moore Machines

- So far we considered Moore machines where the output $O$ is a function of only the current state $Q$

- Moore FSM State Transition Graph
Mealy Machines

- In Mealy machines the output $O$ is a function of the current state $Q$ and input $I$.

- Mealy FSM State Transition Diagram