Topics

- Basics of sequential machines.
- Sequential machine specification.
- Sequential machine design processes.

D_latch Level Trigger

module D_latch(D, Clk, Q);
    input D, Clk;
    output Q;
    reg Q;

    always @(D or Clk)
    if (Clk)
        Q = D;
endmodule

D_latch Positive Edge Trigger

module flipflop(D, Clock, Q);
    input D, Clock;
    output Q;
    reg Q;

    always @(posedge Clock)
    Q = D;
endmodule

D_flipflop with asynchronous reset

module flipflop(D, Clock, Resetn, Q);
    input D, Clock, Resetn;
    output Q;
    reg Q;

    always @(negedge Resetn or posedge Clock)
    if (!Resetn)
        Q <= 0;
    else
        Q <= D;
endmodule
D_flipflop with synchronous reset

```verilog
module flipflop(D, Clock, Resetn, Q);
    input D, Clock, Resetn;
    output Q;
    reg Q;

    always @(posedge Clock)
        if (!Resetn)
            Q <= 0;
        else
            Q <= D;

endmodule
```

Sequential machines

- Use registers to make primary output values depend on state + primary inputs.
- Varieties:
  - Mealy—outputs function of present state, inputs;
  - Moore—outputs depend only on state.

FSM structure

Constraints on structure

- No combinational cycles.
- All components must have bounded delay.
**Synchronous design**

- Controlled by clock(s).
  - State changes at time determined by the clock.
  - Inputs to registers settle in time for state change.
  - Primary inputs settle in time for combinational delay through logic.

- Machine state is determined solely by registers.
  - Don’t have to worry about timing constraints, events outside the registers.

**Non-functional requirements and optimization**

- Performance:
  - Clock period is determined by combinational logic delay.

- Area:
  - Combinational logic size usually dominates area.

- Energy/power:
  - Often dominated by combinational logic.
  - May be improved by latching values.

**Models of state machines**

- Register-transfer:
  - Combinational equations for inputs to registers.

- State transition graph/table:
  - Next-state, output functions described piecewise.

**State transition graph**

- Each transition describes part of the next-state, output functions:

```
S1  0/010  S2
    / 
   /    
S1   S3
    / 
   / 0/101

S1  1/1-0  S3
```
Register-transfer structure

- Registers fed by combinational logic:

D Q D Q

D Q D Q

D Q D Q

Block diagram

- Purely structural description:

A

B1

B2

Symbolic values

- A sequential machine description may use symbolic, not binary values.
  - Symbolic values must be encoded during implementation.
- Encoding may optimize implementation characteristics:
  - Area.
  - Performance.
  - Energy.

STG vs. register-transfer

- Each representation is easier for some types of machines.
- Example: counter.
Counter state transition graph

- Cyclic structure:

![Diagram of counter state transition graph]

Counter register-transfer function

- Specify using addition:
  - Next_count = count + 1.
- Regular structure of logic.

Example: 01 string recognizer

- Recognize 01 sequence in input string:

![Diagram of recognizer state transition graph]

Recognizer state transition graph

- Bit 1
  - 1/0 → 0/0
  - 0/0 → 1/1
- Bit 2
  - 1/0 → 0/0
  - 0/0 → 1/1
Mealy vs. Moore machine

- Moore machine:
  - Output a function of state.
- Mealy machine:
  - Output a function of primary inputs + state.

Sequential machine definition

- Machine computes next state $N$, primary outputs $O$ from current state $S$, primary inputs $I$.
- Next-state function:
  - $N = \delta(I,S)$.
- Output function (Mealy):
  - $O = \lambda(I,S)$.

Reachability

- State is reachable if there is a path from given state.
- May be created by state encoding:

Homing sequence

- Sequence of inputs that drives the machine to a given state.
Equivalent states

States are equivalent if they cannot be distinguished by any input sequence:

Networks of FSMs

Functions can be built up from interconnected FSMs:

Illegal composition of Mealy machines

Communicating FSM states
Two connected machines:

- Component STGs

Behavior of connected machines

Forming product machine

- Form Cartesian product of states:
  - R1S1, R1S2, R2S1, R2S2, R3S1, R3S2.
- For each product state, determine the combined behavior of each product transition:
  - Required inputs.
  - Produced output.
  - Next product state.
State assignment

- Find a binary code for symbolic values in machine.
  - Optimize area, performance.
  - May be done on inputs, outputs as well.

Optimizing state assignments

- Codes affect the next-state, output logic.
  - Compute conditions based on state.
- Best code depends on the input, output logic and its interaction with state computations.

Encoding a shift register

- Symbolic state transition table for shift register:

<table>
<thead>
<tr>
<th>0</th>
<th>S00</th>
<th>S00</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S00</td>
<td>S10</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S01</td>
<td>S00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>S01</td>
<td>S10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>S10</td>
<td>S01</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S10</td>
<td>S11</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S11</td>
<td>S01</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>S11</td>
<td>S11</td>
<td>1</td>
</tr>
</tbody>
</table>

Bad encoding

- Let S00 = 00, S01 = 01, S10 = 10, S11 = 10.
- Logic:
  - Output = S1 S0' + S1' S0
  - N1 = I
  - N0 = I S1' + I S1
Good encoding

- Let $S_{00} = 00$, $S_{01} = 01$, $S_{10} = 10$, $S_{11} = 11$.
- Logic:
  - Output = $S_0$
  - $N_1 = 1$
  - $N_0 = S_1$

One-hot code

- N-state machine has n-bit encoding.
- Ith bit is 1 if machine is in state i.
- Comparison:
  - Easy to tell what state the machine is in.
  - Easy to get the machine into an illegal state (0000, 1111, etc.).
  - Uses a lot of registers.

Common factors in state coding

- Consider this set of transitions:
  - 0, $s_1$ OR $s_2$ -> $s_3$, 1
- Want to choose a code that easily produces $s_1$ OR $s_2$.
  - $S_1 = 00$, $S_2 = 01$.

State codes in n-space

- $s_2$ code = 110
- $s_1$ code = 111
State codes and delay

input

state

output

D Q

next state logic

D Q

next state

output

state in (s2, s4)