One Flip-Flop per State

Alternative Design

- A flip-flop is assigned to each state
- Only one flip-flop may be true
- Each flip-flop represents a state

The next four slides give:

- Symbol substitution rules that:
  - Change an ASM chart into:
    - A sequential circuit with one flip-flop per state.
State Box Transformation

$\rightarrow$ D flip-flop
Decision Box Transformation

\[ \rightarrow \text{Demultiplexer} \]
The previous three transformations may be used to transform the sequencing part of a ASM chart into a circuit with one flip-flop per state.
CS2022 Sequencing Part of ASM Chart
Conditional Output Box Transformation

- Control output is generated by:
  - Attaching Control line in the right location
  - Adding output logic
  - The Original ASM is used for the control
Transformation

Replace:

1. State boxes with D flip-flops
2. Decision boxes with Demultiplexers
3. Junctions with OR gates
4. Add output signals

Use table on the following slide
<table>
<thead>
<tr>
<th>Block Diagram Module</th>
<th>Microoperation</th>
<th>Control Signal Name</th>
<th>Control Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register A:</td>
<td>A ← 0</td>
<td>Initialize</td>
<td>IDLE \cdot G</td>
</tr>
<tr>
<td></td>
<td>A ← A + B</td>
<td>Load</td>
<td>MUL0 \cdot Q_0</td>
</tr>
<tr>
<td></td>
<td>C∥A∥Q ← sr C∥A∥Q</td>
<td>Shift_dec</td>
<td>MUL1</td>
</tr>
<tr>
<td>Register B:</td>
<td>B ← IN</td>
<td>Load_B</td>
<td>LOADB</td>
</tr>
<tr>
<td>Flip-Flop C:</td>
<td>C ← 0</td>
<td>Clear_C</td>
<td>IDLE \cdot G + MUL1</td>
</tr>
<tr>
<td></td>
<td>C ← C_{out}</td>
<td>Load</td>
<td>—</td>
</tr>
<tr>
<td>Register Q:</td>
<td>Q ← IN</td>
<td>Load_Q</td>
<td>LOADQ</td>
</tr>
<tr>
<td></td>
<td>C∥A∥Q ← sr C∥A∥Q</td>
<td>Shift_dec</td>
<td>—</td>
</tr>
<tr>
<td>Counter P:</td>
<td>P ← n - 1</td>
<td>Initialize</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>P ← P - 1</td>
<td>Shift_dec</td>
<td>—</td>
</tr>
</tbody>
</table>
Binary Multiplier Control Unit
One Flip-Flop per State
-- Binary Multiplier with n = 4: VHDL Description
-- See Figures 8-6 and 8-7 for block diagram and ASM Chart
-- in Mano and Kime

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity binary_multiplier is
  port(CLK, RESET, G, LOADB, LOADQ: in std_logic;
       MULT_IN: in std_logic_vector(3 downto 0);
       MULT_OUT: out std_logic_vector(7 downto 0));
end binary_multiplier;
architecture behavior_4 of binary_multiplier is
  type state_type is (IDLE, MUL0, MUL1);
  signal state, next_state : state_type;
  signal A, B, Q: std_logic_vector(3 downto 0);
  signal P: std_logic_vector(1 downto 0);
  signal C, Z: std_logic;
begin
  Z <= P(1) NOR P(0);
  MULT_OUT <= A & Q;
state_register: process (CLK, RESET)
begin
    if (RESET = '1') then
        state <= IDLE;
    elsif (CLK'event and CLK = '1') then
        state <= next_state;
    end if;
end process;
next_state_func: process (G, Z, state)
begin
  case state is
  when IDLE =>
    if G = '1' then
      next_state <= MUL0;
    else
      next_state <= IDLE;
    end if;
  when MUL0 =>
    next_state <= MUL1;
  when MUL1 =>
    if Z = '1' then
      next_state <= IDLE;
    else
      next_state <= MUL0;
    end if;
  end case;
end process;
datapath_func: process (CLK)
variable CA: std_logic_vector(4 downto 0);
begin
  if (CLK’event and CLK = '1') then
    if LOADB = '1' then
      B <= MULT_IN;
    end if;
    if LOADQ = '1' then
      Q <= MULT_IN;
    end if;
  end if;
end process;
directed_func: process (CLK) Part 2

case state is
  when IDLE =>
    if G = '1' then
      C <= '0';
      A <= "0000";
      P <= "11";
    end if;
  when MUL0 =>
    if Q(0) = '1' then
      CA := ('0' & A) + ('0' & B);
    else
      CA := C & A;
    end if;
    C <= CA(4);
    A <= CA(3 downto 0);
  when MUL1 =>
    C <= '0';
    A <= C & A(3 downto 1);
    Q <= A(0) & Q(3 downto 1);
    P <= P - "01";
end case;
end if;
end process;
end behavior_4;