State Machines

Module 6
State Machines

• A common form of sequential logic circuit
  – relatively easy to design and analyze
• “Clocked Synchronous State Machine”
  – clocked - storage elements (flip-flops) have clock input
  – synchronous - flip-flops have common clock signal
  – only changes state on clock transition (edge)
• Use a standard coding convention
  – enumerated type for states
  – processes (state memory and next state logic)
• Need to understand state encoding (optimal design)
  – one-hot
  – binary
  – other
State Machines

• Block Diagram - Moore Machine
  – Outputs determined by current state
State Machine (cont’d)

• Current state determined by state memory (flip-flops)
• Outputs are decoded from the value of the current state
  – combinational logic
• Next state is determined by current state and inputs at time of next triggering clock edge.
Simple State Machine Example

RESET=1

EN=0

S0
C=0
EN=1

S3
C=1
EN=0

S2
C=0
EN=1

S1
C=0
EN=0
STATE MACHINE CODING STYLE

ENTITY sm1 IS
  PORT(clk, reset, en : IN std_logic;
       c : OUT std_logic);
END sm1;

ARCHITECTURE behav OF sm1 IS
  TYPE state_type IS (s0, s1, s2, s3); -- enumerated type
  SIGNAL current_state, next_state : state_type;
BEGIN
  state_memory: PROCESS(clk, reset) -- see next slides for detail

    END PROCESS state_memory;

    next_state_logic: PROCESS(en, current_state)

    END PROCESS next_state_logic;
END behav;
BEGIN
state_memory:
PROCESS(clk, reset)
BEGIN
IF reset = '1' THEN
    current_state <= s0;
ELSIF clk'EVENT AND clk = '1' THEN -- triggering edge
    current_state <= next_state;
END IF;
END PROCESS state_memory;

next_state Process

next_state_logic:
PROCESS(en, current_state)
BEGIN
    CASE current_state IS
        WHEN s0 =>
            IF en = '1' THEN
                next_state <= s1;
            ELSE
                next_state <= s0;
            END IF;
            c <= '0';
        WHEN s1 =>
            IF en = '1' THEN
                next_state <= s2;
            ELSE
                next_state <= s1;
            END IF;
            c <= '0';
        WHEN s2 =>
            IF en = '1' THEN
                next_state <= s3;
            ELSE
                next_state <= s2;
            END IF;
            c <= '0';
        WHEN s3 =>
            IF en = '1' THEN
                next_state <= s0;
            ELSE
                next_state <= s3;
            END IF;
            c <= '1';
    END CASE;
END PROCESS next_state_logic;
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sm1 is
  port ( clk : in std_logic;
          reset : in std_logic);
  signal current_state, next_state : state_type;
  c : out std_logic;
end sm1;

architecture Behavioral of sm1 is
  type state_type is (s0, s1, s2, s3); -- enumerated type
  signal current_state, next_state : state_type;
begin
  state_memory_process : process(clk, reset)
  begin
    if reset = '1' then
      current_state <= s0;
      else
        if clk'event and clk = '1' then
          current_state <= next_state;
    end if;
  end process state_memory_process;

  next_state_logic_process : process(en, current_state)
  begin
    case current_state is
      when s0 =>
        if en = '1' then
          current_state <= s1;
        else
          next_state <= s0;
        end if;
      when s1 =>
        if en = '1' then
          current_state <= s2;
        else
          next_state <= s1;
        end if;
      when s2 =>
        if en = '1' then
          current_state <= s3;
        else
          next_state <= s2;
        end if;
      when s3 =>
        if en = '1' then
          current_state <= s0;
        else
          next_state <= s3;
        end if;
    end case;
  end process next_state_logic_process;
end Behavioral;
```
SM1 - Synthesis Results

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity sm1 is
  port ( clk : in std_logic;
          reset : in std_logic;
          in : in std_logic;
          out : out std_logic);
end sm1;

architecture Behavioral of sm1 is
  TYPE state_type IS (s0, s1, s2, s3): -- enumerated type
  SIGNAL current_state, next_state : state_type;
begin
  state_memory_process: PROCESS(clk, reset)
  BEGIN
    IF reset = '1' THEN
      current_state <= s0;
    ELSE
      IF clk'EVENT AND clk = '1' THEN -- triggering edge
        current_state <= next_state;
      END IF;
    END IF;
  END PROCESS state_memory_process;

  next_state_logic_process: PROCESS(in, current_state)
  BEGIN
    CASE current_state IS
      ...
    END CASE;
  END PROCESS next_state_logic_process;
end Behavioral;
```

Jim Duckworth, WPI 11 State Machines - Module 6
Synthesis Report

--- Synthesis Report ---

* HDL Synthesis *

--- HDL Synthesis Report ---

Synthesizing Unit <sm1>.
Related source file is C:/ee574/sm1/sm1.vhd.
Found finite state machine <FSM_0> for signal <current_state>.

| States    | 4 |
| Transitions | 8 |
| Inputs    | 1 |
| Outputs   | 1 |
| Clock     | clk (rising_edge) |
| Reset     | reset (positive) |
| Reset type | asynchronous |
| Reset State | s0 |
| Power Up State | s0 |
| Encoding  | automatic |
| Implementation | LUT |

Summary:
inferred 1 Finite State Machine(s).
Unit <sm1> synthesized.

--- Advanced HDL Synthesis ---

Advanced RAM inference ...
Advanced multiplier inference ...
Advanced Registered AddSub inference ...
Selecting encoding for FSM_0 ...
Optimizing FSM <FSM_0> on signal <current_state> with one-hot encoding.
Dynamic shift register inference ...

--- HDL Synthesis Report ---

Macro Statistics
# FSMs : 1
# Registers : 4
1-bit register : 4
SM1 - Schematic
Behavioral Simulation
Encoding Style

• One-hot
  – use one flip-flop per state
  – only one flip-flop active (hot)
  – best for flip-flop rich technology
    • use more flip-flops but simpler next state logic (faster)
  – e.g. Xilinx FPGAs (Spartan 3, Virtex, etc)

• Sequential (binary) encoding
  – generates sequential values for enumerated states
    • 00, 01, 10, 11
  – less flip-flops but more complex next-state logic
  – e.g. Altera CPLDs
HDL Options for FSM Encoding

![HDL Options for FSM Encoding Diagram]
### Synthesis Results with Sequential Encoding

**Process Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSM Encoding Algorithm</td>
<td>Auto</td>
</tr>
<tr>
<td>Case Implementation Style</td>
<td>Auto</td>
</tr>
<tr>
<td>RAM Style</td>
<td>One-hot</td>
</tr>
<tr>
<td>ROM Extraction</td>
<td>Sequential</td>
</tr>
<tr>
<td>ROM Style</td>
<td>Johnson</td>
</tr>
<tr>
<td>Mix Extraction</td>
<td>User</td>
</tr>
<tr>
<td>Mix Style</td>
<td>None</td>
</tr>
<tr>
<td>Decoder Extraction</td>
<td>Yes</td>
</tr>
<tr>
<td>Priority Encoder Extension</td>
<td>Yes</td>
</tr>
<tr>
<td>Shift Register Extension</td>
<td>Yes</td>
</tr>
<tr>
<td>Logical Shifter Extension</td>
<td>Yes</td>
</tr>
<tr>
<td>XOR Collapsing</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource Sharing</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiplier Style</td>
<td>Auto</td>
</tr>
</tbody>
</table>

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

type sm is
  record
    clk : in std_logic;
    reset : in std_logic;
    en : in std_logic;
    c : out std_logic;
  end record;

architecture Behavioral of sm is
begin
  process
    type state_type is (s0, s1, s2, s3);
    -- enumerated type
    variable current_state, next_state : state_type;
    variable state_memory_process : PROCESS(clk, reset);  
    variable i : integer;
    variable start : state_type := s0;
  begin
    if reset = '1' then
      current_state <= s0;
      if clk'EVENT AND clk = '1' then  -- triggering edge
        current_state <= next_state;
      end if;
    end if;
    state_memory_process := PROCESS(clk, reset);
    process
      variable new_state : state_type;
    begin
      case current_state is  
        when s0 =>
        start <= start;
        when s1 =>
        start <= start;
        when s2 =>
        start <= start;
        when s3 =>
        start <= start;
      end case;
    end process;
    process
      variable i : natural;
    begin
      if reset = '1' then
        state_memory_process <= state_memory_process;
      else
        state_memory_process <= state_memory_process;
      end if;
    end process;
  end process;
end Behavioral;
```

**Jim Duckworth, WPI**  
17  
**State Machines - Module 6**
Same VHDL but selecting Sequential
One-Hot Encoding Style (forcing with VHDL)

ENTITY sm1_onehot IS
  PORT(clk, reset, en : IN std_logic;
       c : OUT std_logic);
END sm1_onehot;

ARCHITECTURE behav OF sm1_onehot IS
  SUBTYPE state_vec IS std_logic_vector(0 TO 3);
  SIGNAL current_state, next_state : state_vec;
BEGIN
  state_memory: PROCESS(clk, reset)
  BEGIN
    IF reset = '1' THEN
      current_state <= state_vec'(OTHERS => '0');
      current_state(0) <= '1';
    ELSIF clk'EVENT AND clk = '1' THEN -- triggering edge
      current_state <= next_state;
    END IF;
  END PROCESS state_memory;

One-Hot Encoding Style (cont’d)

next_state_logic:
    PROCESS(en, current_state)
    BEGIN
        next_state <= state_vec'(OTHERS => '0');  -- initialize to all 0’s
        c <= '0';
        IF current_state(0) = '1' THEN  -- in state 0
            IF en = '1' THEN
                next_state(1) <= '1';
            ELSE
                next_state(0) <= '1';
            END IF;
        END IF;
        IF current_state(1) = '1' THEN  -- in state 1 etc...
            IF en = '1' THEN
                next_state(0) <= '1';
            ELSE
                next_state(3) <= '1';
            END IF;
            c <= '1';
        END IF;
        IF current_state(3) = '1' THEN  -- in state 3
            IF en = '1' THEN
                next_state(0) <= '1';
            ELSE
                next_state(3) <= '1';
            END IF;
            c <= '1';
        END IF;
    END PROCESS next_state_logic;
END behav;
VHDL Description – forcing one-hot

```vhdl
architecture Behavioral of sm1_one_hot is
    type state_vec is std_logic_vector(0 to 3);
    signal current_state, next_state : state_vec;
begin
    process(clk, reset)
    begin
        if reset = '1' then
            current_state <= state_vec(OTHERS => '0');
            current_state(0) <= '1';
            x Spartan 4 Event and clk = '1' then -- triggering edge
                current_state <= next_state;
        end if;
    end process state_memory;

    process, current_state
    begin
        next_state <= state_vec(OTHERS => '0'); -- initialize to all 0's
        current_state(0) <= '0';
        if current_state(0) = '1' then
            next_state(1) <= '1';
        else
            next_state(0) <= '1';
        end if;
    end process;

    process, current_state
    begin
        next_state <= state_vec(OTHERS => '0'); -- initialize to all 0's
        current_state(0) <= '0';
        if current_state(0) = '1' then
            next_state(1) <= '1';
        else
            next_state(0) <= '1';
        end if;
    end process;

    process, current_state
    begin
        next_state <= state_vec(OTHERS => '0'); -- initialize to all 0's
        current_state(0) <= '0';
        if current_state(0) = '1' then
            next_state(1) <= '1';
        else
            next_state(0) <= '1';
        end if;
    end process;

    process, current_state
    begin
        next_state <= state_vec(OTHERS => '0'); -- initialize to all 0's
        current_state(0) <= '0';
        if current_state(0) = '1' then
            next_state(1) <= '1';
        else
            next_state(0) <= '1';
        end if;
    end process;

    end Behavioral;
end sm1_one_hot;
```
SM1_onehot: Synthesis summary

Jim Duckworth, WPI

State Machines - Module 6
SM1_onehot - Schematic
Functional Simulation
Monitor internal Signals

- Select **Structure Window**

- Select **Signals Window**

- Right mouse click
  - select **Add to Wave** => **Selected Signals**
Displaying internal signals

- Type **Restart** at Modelsim prompt
- Type **Run 1uS**
Timing Analysis (Xilinx Timing Report)

- **One Hot**
  - Minimum clock period : 18.9 ns
  - Estimated Maximum Clock Speed : 52.9 MHz

- **Sequential**
  - Minimum clock period : 25.8 ns
  - Estimated Maximum Clock Speed : 38.7 MHz
-- one process with one state signal
ARCHITECTURE behav OF sm2 IS
    TYPE state_type IS (s0, s1, s2, s3); -- enumerated type
    SIGNAL state : state_type;
BEGIN
    PROCESS(clk, reset)
    BEGIN
        IF reset = '1' THEN
            state <= s0;
        ELSIF clk'EVENT AND clk = '1' THEN -- triggering edge
            CASE state IS
                WHEN s0 =>
                    IF en = '1' THEN
                        state <= s1;
                    ELSE
                        state <= s0;
                    END IF;
                    c <= '0';
                WHEN s1 =>
                    IF en = '1' THEN
                        state <= s2;
                    END IF;
                    c <= '0';
ELSE
  state <= s1;
END IF;
c <= '0';
WHEN s2 =>
  IF en = '1' THEN
    state <= s3;
  ELSE
    state <= s2;
  END IF;
c <= '0';
WHEN s3 =>
  IF en = '1' THEN
    state <= s0;
  ELSE
    state <= s3;
  END IF;
c <= '1';
END CASE;
END IF;
END PROCESS;
END behav;
SM2 - Synthesis Results
Registered Outputs

- Output signal assignments within state machine process
- No glitches on registered outputs
- Outputs are registered in parallel with state registers
- Register introduces a one cycle delay
Block Diagram (registered outputs)

- Inputs
- Clock
- Next State Logic
- Current State
- State Memory
- Output Logic
- Output Register
- Outputs
Functional Simulation
Block Diagram (using next state logic)

- Inputs
- Clock
- Next State Logic
- State Memory
- Current State
- Output Logic
- Output Register
- Outputs
Registered Outputs (no delay)

• Three processes:
  – state_memory
    ```
    PROCESS (clk, reset)
      ELSIF clk'EVENT AND clk = '1' THEN
        current_state <= next_state;
    ```
  – next_state_logic
    ```
    PROCESS (en, current_state)
      CASE current_state IS
        WHEN s0 =>
          next_state <= s1;
    ```
  – registered_output_logic (uses next state to determine outputs)
    ```
    PROCESS (clk, reset)
      ELSIF clk'EVENT AND clk = '1' THEN
        CASE next_state IS
          WHEN s3 =>
            c <= '1';
          WHEN OTHERS =>
            c <= '0';
    ```