ECE 448
Lecture 6

Finite State Machines
State Diagrams,
State Tables,
Algorithmic State Machine (ASM) Charts,
and VHDL Code
Required reading

• P. Chu, *FPGA Prototyping by VHDL Examples*
  
  *Chapter 5, FSM*
Recommended reading

• S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*
  
  Chapter 8, *Synchronous Sequential Circuits*
  
  Sections 8.1-8.5
  
  Section 8.10, *Algorithmic State Machine (ASM)*
  
  Charts
Datapath vs. Controller
Structure of a Typical Digital System

Datapath (Execution Unit) → Controller (Control Unit)

Data Inputs → Control Signals → Status Signals → Data Outputs

Control & Status Inputs → Data Outputs

Control & Status Outputs
Datapath (Execution Unit)

- Manipulates and processes data
- Performs arithmetic and logic operations, shifting/rotating, and other data-processing tasks
- Is composed of registers, multiplexers, adders, decoders, comparators, ALUs, gates, etc.
- Provides all necessary resources and interconnects among them to perform specified task
- Interprets control signals from the Controller and generates status signals for the Controller
Controller (Control Unit)

- Controls data movement in the Datapath by switching multiplexers and enabling or disabling resources
  - Example: enable signals for registers
  - Example: select signals for muxes
- Provides signals to activate various processing tasks in the Datapath
- Determines the sequence of operations performed by the Datapath
- Follows Some ‘Program’ or Schedule
Programmable vs. Non-Programmable Controller

- Controller can be programmable or non-programmable
- Programmable
  - Has a program counter which points to next instruction
  - Instructions are held in a RAM or ROM
  - Microprocessor is an example of programmable controller
- Non-Programmable
  - Once designed, implements the same functionality
  - Another term is a “hardwired state machine,” or “hardwired FSM,” or “hardwired instructions”
  - In this course we will be focusing on non-programmable controllers.
Finite State Machines

- Controllers can be described as Finite State Machines (FSMs)
- Finite State Machines can be represented using
  - *State Diagrams and State Tables* - suitable for simple controllers with a relatively few inputs and outputs
  - *Algorithmic State Machine (ASM) Charts* - suitable for complex controllers with a large number of inputs and outputs
- All of these descriptions can be easily translated to the corresponding synthesizable VHDL code
Hardware Design with RTL VHDL

Interface

Pseudocode

Datapath

Block diagram

Controller

Block diagram

State diagram or ASM chart

VHDL code

VHDL code

VHDL code
Steps of the Design Process

1. Text description
2. Interface
3. Pseudocode
4. Block diagram of the Datapath
5. Interface divided into Datapath and Controller
6. State diagram or ASM chart of the Controller
7. RTL VHDL code of the Datapath, Controller, and Top-Level Unit
8. Testbench for the Datapath, Controller, and Top-Level Unit
9. Functional simulation and debugging
10. Synthesis and post-synthesis simulation
11. Implementation and timing simulation
12. Experimental testing using FPGA board
Steps of the Design Process
Introduced in Class Today

1. Text description
2. Interface
3. Pseudocode
4. Block diagram of the Datapath
5. Interface divided into Datapath and Controller
6. State diagram or ASM chart of the Controller
7. RTL VHDL code of the Datapath, Controller, and Top-level Unit
8. Testbench for the Datapath, Controller, and Top-Level Unit
9. Functional simulation and debugging
10. Synthesis and post-synthesis simulation
11. Implementation and timing simulation
12. Experimental testing using FPGA board
Finite State Machines Refresher
Finite State Machines (FSMs)

- An FSM is used to model a system that transits among a finite number of internal states. The transitions depend on the current state and external input.
- The main application of an FSM is to act as the controller of a medium to large digital system.
- Design of FSMs involves:
  - Defining states
  - Defining next state and output functions
  - Optimization / minimization
- Manual optimization/minimization is practical for small FSMs only.
Moore FSM

- Output is a Function of the Present State Only
Mealy FSM

- Output is a Function of the Present State and the Inputs
State Diagrams
Moore Machine

state 1 / output 1

transition condition 1

transition condition 2

state 2 / output 2
Mealy Machine

transition condition 1 / output 1

state 1

transition condition 2 / output 2

state 2
Moore FSM - Example 1

- Moore FSM that Recognizes Sequence “10”

Meaning of states:

S0: No elements of the sequence observed
S1: “1” observed
S2: “10” observed
Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence “10”

Meaning of states:

- S0: No elements of the sequence observed
- S1: “1” observed
Moore & Mealy FSMs without delays

Moore

Mealy

<table>
<thead>
<tr>
<th>clock</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S0</td>
</tr>
<tr>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>state</th>
<th>S0</th>
<th>S0</th>
<th>S1</th>
<th>S0</th>
<th>S0</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Moore & Mealy FSMs with delays

Moore

<table>
<thead>
<tr>
<th>state</th>
<th>S0</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S0</th>
<th>S0</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mealy

<table>
<thead>
<tr>
<th>state</th>
<th>S0</th>
<th>S0</th>
<th>S1</th>
<th>S0</th>
<th>S0</th>
<th>S0</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Moore vs. Mealy FSM (1)

• Moore and Mealy FSMs Can Be Functionally Equivalent
  • Equivalent Mealy FSM can be derived from Moore FSM and vice versa

• Mealy FSM Has Richer Description and Usually Requires Smaller Number of States
  • Smaller circuit area
Moore vs. Mealy FSM (2)

- Mealy FSM Computes Outputs as soon as Inputs Change
  - Mealy FSM responds one clock cycle sooner than equivalent Moore FSM
- Moore FSM Has No Combinational Path Between Inputs and Outputs
  - Moore FSM is less likely to affect the critical path of the entire circuit
Moore vs. Mealy FSM (3)

- Types of control signal
  - Edge sensitive
    - E.g., enable signal of a counter
    - Both can be used but Mealy is faster
  - Level sensitive
    - E.g., write enable signal of SRAM
    - Moore is preferred
Which Way to Go?

Mealy FSM
- Fewer states
- Lower Area
- Responds one clock cycle earlier

Moore FSM
- Safer.
- Less likely to affect the critical path.
Problem 1

Assuming state diagram given on the next slide, supplement timing waveforms given in the answer sheet with the correct values of signals \textbf{State} and \textbf{c}, in the interval from 0 to 575 ns.
Finite State Machines in VHDL
FSMs in VHDL

• Finite State Machines Can Be Easily Described With Processes
• Synthesis Tools Understand FSM Description if Certain Rules Are Followed
  • **State transitions** should be described in a **process** sensitive to **clock** and **asynchronous reset** signals only
  • **Output function** described using rules for combinational logic, i.e. as **concurrent statements** or a **process** with all inputs in the sensitivity list
Moore FSM

process(clock, reset)

Inputs → Next State function

Present State Register

Output function → Outputs

clock reset

concurrent statements
Mealy FSM

process(clock, reset)

Next State

Present State Register

Output function

Inputs

concurrent statements

clock
reset

Output

Present State

Next State function

Inputs

Mealy FSM

process(clock, reset)
Moore FSM - Example 1

• Moore FSM that Recognizes Sequence “10”
Moore FSM in VHDL (1)

TYPE state IS (S0, S1, S2);
SIGNAL Moore_state: state;

U_Moore: PROCESS (clock, reset)
BEGIN
  IF(reset = '1') THEN
    Moore_state <= S0;
  ELSIF rising_edge(clock) THEN
    CASE Moore_state IS
    WHEN S0 =>
      IF input = '1' THEN
        Moore_state <= S1;
      ELSE
        Moore_state <= S0;
      END IF;
    END CASE;
  END IF;
END PROCESS;
Moore FSM in VHDL (2)

WHEN S1 =>
    IF input = '0' THEN
        Moore_state <= S2;
    ELSE
        Moore_state <= S1;
    END IF;
WHEN S2 =>
    IF input = '0' THEN
        Moore_state <= S0;
    ELSE
        Moore_state <= S1;
    END IF;
END CASE;
END IF;
END PROCESS;

Output <= '1' WHEN Moore_state = S2 ELSE '0';
Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence “10”
Mealy FSM in VHDL (1)

TYPE state IS (S0, S1);
SIGNAL Mealy_state: state;

U_Mealy: PROCESS(clock, reset)
BEGIN
  IF(reset = '1') THEN
    Mealy_state <= S0;
  ELSIF rising_edge(clock) THEN
    CASE Mealy_state IS
      WHEN S0 =&gt;
        IF input = '1' THEN
          Mealy_state <= S1;
        ELSE
          Mealy_state <= S0;
        END IF;
    END CASE;
  END IF;
END U_Mealy;
Mealy FSM in VHDL (2)

WHEN S1 =>
  IF input = ‘0’ THEN
    Mealy_state <= S0;
  ELSE
    Mealy_state <= S1;
  END IF;
END CASE;
END IF;
END PROCESS;

Output <= ‘1’ WHEN (Mealy_state = S1 AND input = ‘0’) ELSE ‘0’;
Algorithmic State Machine (ASM) Charts
Algorithmic State Machine

Algorithmic State Machine – representation of a Finite State Machine suitable for FSMs with a larger number of inputs and outputs compared to FSMs expressed using state diagrams and state tables.
Elements used in ASM charts (1)

State name

Output signals or actions (Moore type)

Condition expression

0 (False)  1 (True)

(a) State box  (b) Decision box

Conditional outputs or actions (Mealy type)

(c) Conditional output box
State Box

- **State box** – represents a state.
- Equivalent to a node in a state diagram or a row in a state table.
- Contains register transfer actions or output signals.
- **Moore-type outputs are listed inside of the box.**
- It is customary to write only the name of the signal that has to be asserted in the given state, e.g., \( z \) instead of \( z \leq 1 \).
- Also, it might be useful to write an action to be taken, e.g., \( \text{count} \leq \text{count} + 1 \), and only later translate it to asserting a control signal that causes a given action to take place (e.g., enable signal of a counter).
Decision Box

• **Decision box** – indicates that a given condition is to be tested and the exit path is to be chosen accordingly. The condition expression may include one or more inputs to the FSM.
Conditional Output Box

- **Conditional output box**

- Denotes output signals that are of the Mealy type.

- The condition that determines whether such outputs are generated is specified in the decision box.
ASMs representing simple FSMs

- Algorithmic state machines can model both Mealy and Moore Finite State Machines
- They can also model machines that are of the mixed type
Generalized FSM

Based on RTL Hardware Design by P. Chu
Moore FSM – Example 2: State diagram
# Moore FSM – Example 2: State table

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state ( w = 0 )</th>
<th>Next state ( w = 1 )</th>
<th>Output ( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
ASM Chart for Moore FSM – Example 2
Example 2: VHDL code (1)

USE ieee.std_logic_1164.all;

ENTITY simple IS
  PORT ( Clock : IN STD_LOGIC;
         Reset : IN STD_LOGIC;
         w : IN STD_LOGIC;
         z : OUT STD_LOGIC );
END simple;

ARCHITECTURE Behavior OF simple IS
  TYPE State_type IS (A, B, C);
  SIGNAL y : State_type;
BEGIN
  PROCESS ( Reset, Clock )
  BEGIN
    IF Reset = '1' THEN
      y <= A;
    ELSIF rising_edge(Clock) THEN
      ...
CASE y IS
  WHEN A =>
    IF w = '1' THEN
      y <= B ;
    ELSE
      y <= A ;
    END IF ;
  WHEN B =>
    IF w = '1' THEN
      y <= C ;
    ELSE
      y <= A ;
    END IF ;
  WHEN C =>
    IF w = '1' THEN
      y <= C ;
    ELSE
      y <= A ;
    END IF ;
END CASE ;
Example 2: VHDL code (3)

END IF;
END PROCESS;

z <= '1' WHEN y = C ELSE '0';

END Behavior;
Mealy FSM – Example 3: State diagram

\[ \begin{align*}
A & \quad w = 0 / z = 0 \\
B & \quad w = 1 / z = 1 \\
\text{Reset} & \quad w = 1 / z = 0
\end{align*} \]
ASM Chart for Mealy FSM – Example 3
Example 3: VHDL code (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Mealy IS
    PORT ( Clock : IN STD_LOGIC;
           Reset : IN STD_LOGIC;
           w     : IN STD_LOGIC;
           z     : OUT STD_LOGIC );
END Mealy;

ARCHITECTURE Behavior OF Mealy IS
    TYPE State_type IS (A, B);
    SIGNAL y : State_type;
BEGIN
    PROCESS ( Reset, Clock )
    BEGIN
        IF Reset = '1' THEN
            y <= A;
        ELSIF rising_edge(Clock) THEN

Example 3: VHDL code (2)

```
CASE y IS
  WHEN A =>
    IF w = '1' THEN
      y <= B ;
    ELSE
      y <= A ;
    END IF ;
  WHEN B =>
    IF w = '1' THEN
      y <= B ;
    ELSE
      y <= A ;
    END IF ;
END CASE ;
```
Example 3: VHDL code (3)

END IF ;
END PROCESS ;

z <= '1' WHEN (y = B) AND (w='1') ELSE '0' ;

END Behavior ;
Control Unit Example: Arbiter (1)
Control Unit Example: Arbiter (2)
Control Unit Example: Arbiter (3)
ASM Chart for Control Unit - Example 4
Example 4: VHDL code (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY arbiter IS
    PORT ( Clock, Reset : IN STD_LOGIC ;
          r        : IN STD_LOGIC_VECTOR(1 TO 3) ;
          g        : OUT STD_LOGIC_VECTOR(1 TO 3) ) ;
END arbiter ;

ARCHITECTURE Behavior OF arbiter IS
    TYPE State_type IS (Idle, gnt1, gnt2, gnt3) ;
    SIGNAL y : State_type ;
BEGIN
  PROCESS ( Reset, Clock )
  BEGIN
    IF Reset = '1' THEN y <= Idle ;
    ELSIF rising_edge(Clock) THEN
      CASE y IS
        WHEN Idle =>
          IF r(1) = '1' THEN y <= gnt1 ;
          ELSIF r(2) = '1' THEN y <= gnt2 ;
          ELSIF r(3) = '1' THEN y <= gnt3 ;
          ELSE y <= Idle ;
          END IF ;
        WHEN gnt1 =>
          IF r(1) = '1' THEN y <= gnt1 ;
          ELSE y <= Idle ;
          END IF ;
        WHEN gnt2 =>
          IF r(2) = '1' THEN y <= gnt2 ;
          ELSE y <= Idle ;
          END IF ;
        END CASE ;
  END IF ;
END PROCESS ;
Example 4: VHDL code (3)

```vhdl
WHEN gnt3 =>
    IF r(3) = '1' THEN y <= gnt3 ;
    ELSE y <= Idle ;
    END IF ;
END CASE ;
END IF ;
END PROCESS ;

g(1) <= '1' WHEN y = gnt1 ELSE '0' ;
g(2) <= '1' WHEN y = gnt2 ELSE '0' ;
g(3) <= '1' WHEN y = gnt3 ELSE '0' ;
END Behavior ;
```
Problem 2

Assuming ASM chart given on the next slide, supplement timing waveforms given in the answer sheet with the correct values of signals State, $g_1$, $g_2$, $g_3$, in the interval from 0 to 575 ns.
ASM Chart
ASM Summary by Prof. Chu

- ASM (algorithmic state machine) chart
  - Flowchart-like diagram
  - Provides the same info as a state diagram
  - More descriptive, better for complex description

- ASM block
  - One state box
  - One or more optional decision boxes:
    - with T (1) or F (0) exit path
  - One or more conditional output boxes:
    - for Mealy output
Figure 10.4  ASM block.
ASM Chart Rules

• Difference between a regular flowchart and an ASM chart:
  – Transition governed by clock
  – Transition occurs between ASM blocks

• Basic rules:
  – For a given input combination, there is one unique exit path from the current ASM block
  – Any closed loop in an ASM chart must include a state box

Based on RTL Hardware Design by P. Chu
Incorrect ASM Charts

Based on RTL Hardware Design by P. Chu
Generalized FSM

Based on RTL Hardware Design by P. Chu
Alternative Coding Styles
by Dr. Chu
(to be used with caution)
Traditional Coding Style

process(clock, reset)

Inputs

Next State function

Present State Register

Next State

clock reset

Mealy Output function

Mealy Outputs

Moore Output function

Moore Outputs

concurrent statements

Moore Outputs
Alternative Coding Style 1

Process (Present State, Inputs)

Inputs

Next State function

Next State

Present State Register

Process (clock, reset)

Present State

Process (Present State, Inputs)

Mealy Output function

Mealy Outputs

Moore Output function

Moore Outputs
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
port(
  clk, reset: in std_logic;
  mem, rw, burst: in std_logic;
  oe, we, we_me: out std_logic);
end mem_ctrl;

architecture mult_seg_arch of mem_ctrl is
  type mc_state_type is
    (idle, read1, read2, read3, read4, write);
signal state_reg, state_next: mc_state_type;
begin
  -- state register
  process(clk, reset)
  begin
    if (reset='1') then
      state_reg <= idle;
    elsif (clk'event and clk='1') then
      state_reg <= state_next;
    end if;
  end process;

-- next-state logic

process(state_reg, mem, rw, burst)
begin
  case state_reg is
    when idle =>
      if mem='1' then
        if rw='1' then
          state_next <= read1;
        else
          state_next <= write;
        end if;
      else
        state_next <= idle;
      end if;
    when write =>
      state_next <= idle;
  end case;
end process;
when read1 =>
  if (burst='1') then
    state_next <= read2;
  else
    state_next <= idle;
  end if;
when read2 =>
  state_next <= read3;
when read3 =>
  state_next <= read4;
when read4 =>
  state_next <= idle;
end case;
end process;
-- moore output logic
process(state_reg)
begin
    we <= '0'; -- default value
    oe <= '0'; -- default value
    case state_reg is
        when idle =>
        when write =>
            we <= '1';
        when read1 =>
            oe <= '1';
        when read2 =>
            oe <= '1';
        when read3 =>
            oe <= '1';
        when read4 =>
            oe <= '1';
    end case;
end process;
— mealy output logic

process(state_reg, mem, rw)
begin
  we_me <= '0'; — default value
  case state_reg is
    when idle =>
      if (mem='1') and (rw='0') then
        we_me <= '1';
      end if;
    when write =>
    when read1 =>
    when read2 =>
    when read3 =>
    when read4 =>
      end case;
  end process;
end mult_seg_arch;
Alternative Coding Style 2

Process(Present State, Inputs)

Process(clk, reset)
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
port (    clk, reset: in std_logic;
    mem, rw, burst: in std_logic;
    oe, we, we_me: out std_logic);
end mem_ctrl;

architecture mult_seg_arch of mem_ctrl is
type mc_state_type is
    (idle, read1, read2, read3, read4, write);
signal state_reg, state_next: mc_state_type;
begin
    -- state register
    process(clk, reset)
    begin
        if (reset='1') then
            state_reg <= idle;
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
        end if;
    end process;
end mult_seg_arch;
process(state_reg, mem, rw, burst)
begin
  oe <= '0';  -- default values
  we <= '0';
  we_me <= '0';
  case state_reg is
  when idle =>
    if mem='1' then
      if rw='1' then
        state_next <= read1;
      else
        state_next <= write;
      end if;
    end if;
  else
    state_next <= idle;
  end if;
  when write =>
    state_next <= idle;
  we <= '1';
when read1 =>
    if (burst='1') then
        state_next <= read2;
    else
        state_next <= idle;
    end if;
    oe <= '1';
when read2 =>
    state_next <= read3;
    oe <= '1';
when read3 =>
    state_next <= read4;
    oe <= '1';
when read4 =>
    state_next <= idle;
    oe <= '1';
end case;
end process;