Lecture 3-A

Finite State Machines

Asst. Prof. Tolga Ayav, Ph.D.

Department of Computer Engineering
İzmir Institute of Technology
Finite state machines (FSMs) are powerful design elements used to implement algorithms in hardware.

An FSM is a 6-tuple, \(<Z, X, Y, \delta, \lambda, z0>\), where:

- \(Z\) is a set of states \{z0, z1, ..., zl\},
- \(X\) is a set of inputs \{x0, x1, ..., xm\},
- \(Y\) is a set of outputs \{y0, y1, ..., yn\},
- \(\delta\) is a next-state function (i.e., transitions), mapping states and inputs to states, \((Z \times X \rightarrow Y)\)
- \(\lambda\) is an output function, mapping current states to outputs \((Z \rightarrow Y)\),
- and
- \(z0\) is an initial state.
Moore and Mealy FSMs

Classical automata:

input $X$ → Internal state $Z$ → output $Y$

clock

Next state $Z^+$ computed by function $\delta$
Output computed by function $\lambda$

• Moore-automata:
  $Y = \lambda(Z)$; $Z^+ = \delta(X, Z)$
• Mealy-automata
  $Y = \lambda(X, Z)$; $Z^+ = \delta(X, Z)$
Outputs depend on states. $\lambda : (Z \rightarrow Y)$
FSM Model: Mealy

Outputs depends on states and inputs. $\lambda : (X \times Z \rightarrow Y)$
FSMD (Finite-state machines with datapaths)

When using an FSM for embedded system design, the inputs and outputs represent boolean data types, and the functions therefore represent boolean functions with boolean operations. This model is sufficient for purely control systems that do not input or output data. However, when we must deal with data, two new features would be helpful: more complex data types (such as integers or floating point numbers), and variables to store data.

An FSMD is a 7-tuple, \(<Z, X, Y, V, \delta, \lambda, z0>\), where:

- \(Z\) is a set of states \(\{z_0, z_1, \ldots, z_l\}\),
- \(X\) is a set of inputs \(\{x_0, x_1, \ldots, x_m\}\),
- \(Y\) is a set of outputs \(\{y_0, y_1, \ldots, y_n\}\),
- \(V\) is a set of variables \(\{v_0, v_1, \ldots, v_n\}\),
- \(\delta\) is a next-state function (i.e., transitions), mapping states and inputs to states, \((Z \times X \rightarrow Y)\)
- \(\lambda\) is an output function, mapping current states to outputs \((Z \rightarrow Y)\), and
- \(z_0\) is an initial state.
FSMD (Finite-state machines with datapaths)

- Various data types are allowed (like the ones in a typical programming language)
- During execution of the model, the complete system state consists not only of the current state, but also the values of all variables
- $\delta$ and $\lambda$ includes arithmetic operations, not only boolean operations.
- $\lambda$ also includes variable updates.
Describing a system as a state machine

1. List all possible states, giving each a descriptive name.
2. Declare all variables.
3. For each state, list the possible transitions, with associated conditions, to other states.
4. For each state and/or transition, list the associated actions.
5. For each state, ensure that exiting transition conditions are exclusive (no two conditions could be true simultaneously) and complete (one of the conditions is true at any time).
Design a Simple Seat Belt Controller:

The controller's job is to turn on a buzzer if a person sits in a seat and does not fasten the seat belt within a fixed amount of time. This system has three inputs and one output.

The inputs are:
- A sensor for the seat to know when a person has sat down, labeled as `seat`.
- A seat belt sensor that tells when the belt is fastened, labeled as `belt`.
- A timer that goes off when the required time interval has elapsed, labeled as `timer`.

The output is the buzzer, labeled as `buzzer`.

The input states are:
- `seat`: 1 for person sat down, 0 for no person.
- `belt`: 1 for seat belt fastened, 0 for not fastened.
- `timer`: 1 for expired, 0 for not expired.

The output state is:
- `timer_on`: when the timer has expired.

The diagram shows the flow of data from the inputs to the output, with arrows indicating the transitions between states.
State Diagram for the Seat Belt Controller

Inputs/outputs
\((- = no\ action)\)

- Idle
- Buzzer
- Seated
- Belted

Transitions:
- No seat/ -
- Seat/timer on
- No seat/ -
- Timer/buzzer on
- Belt/ -
- No belt/timer on

- No seat/ buzzer off
- Belt/ buzzer off

Izmir Institute of Technology
Real-Time and Embedded System Design
C Code:

```c
#define IDLE 0
#define SEATED 1
#define BELTED 2
#define BUZZER 3

void FSM()
{
    switch(state)
    {
    case IDLE:
        if(seat) {state=SEATED; timer_on=1;}
        break;
    case SEATED:
        if(belt) state=BELTED;
        else if(timer) state=BUZZER;
        break;
    case BELTED:
        if(!seat) state=IDLE;
        else if(!belt) state=SEATED;
        break;
    case BUZZER:
        if(belt) state=BELTED;
        else if(!seat) state=IDLE;
        break;
    }
}

void main()
{
    while(1) FSM();
}
```

Izmir Institute of Technology

Real-Time and Embedded System Design
Adding Hierarchy/Concurrency: HCFSM

In the seatbelt controller example, assume that any state goes to a newly defined state when a push button is pressed. This requires many transitions in the diagram.

(a) three-state example without hierarchy, (b) same example with hierarchy,

Seatbelt controller, cruise controller etc. must be run in concurrency.

(c) concurrency.
The program-state machine (PSM) model extends state machines to allow use of sequential program code to define a state’s actions (including extensions for complex data types and variables), as well as including the hierarchy and concurrency extensions of HCFSM.
1. Construct an FSMD model of automobile cruise controller. Define the sets of FSMD (For example; V={SA, SD, G, K} etc.) and draw the diagram.

2. Write a C program implementing this FSMD.

3. Write a C program running this FSMD and the seatbelt controller FSM concurrently.