ECE 474A/57A
Computer-Aided Logic Design

Lecture 6B
Hierarchical/Concurrent State Machine Models (HCFSM)

Slides Modified From: Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

Outline

- Models vs. Languages
- State Machine Model
  - FSM/FSMD
  - HCFSM and Statecharts Language
  - Program-State Machine (PSM) Model
- *Concurrent Process Model*
  - *Communication*
  - *Synchronization*
  - *Implementation*
- *Dataflow Model*

Introduction

- Describing embedded system's processing behavior extremely difficult
  - Complexity increasing with increasing IC capacity
  - Desired behavior often not fully understood in beginning
    - Many implementation bugs due to description mistakes/omissions

washing machines, small games, etc. required *hundreds* of lines of code

TV set-top boxes, cell phones, etc. required *hundreds of thousands* of lines of code

An example of trying to be precise in English

- English (or other natural language) common starting point
  - Precise description difficult/ impossible
  - Example: Motor Vehicle Code is thousands of pages long

California Vehicle Code: Right-of-way of crosswalks

21950. (a) The driver of a vehicle shall yield the right-of-way to a pedestrian crossing the roadway within any marked crosswalk or within any unmarked crosswalk at an intersection, except as otherwise provided in this chapter.

(b) This section does not relieve a pedestrian from the duty of using due care for his or her safety. No pedestrian may suddenly leave a curb or other place of safety and walk or run into the path of a vehicle that is so close as to constitute an immediate hazard. No pedestrian may unnecessarily stop or delay traffic while in a marked or unmarked crosswalk.

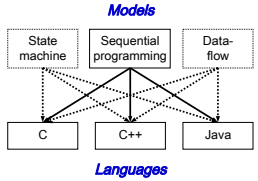
(c) The driver of a vehicle approaching a pedestrian within any marked or unmarked crosswalk shall exercise all due care and shall reduce the speed of the vehicle or take any other action relating to the operation of the vehicle as necessary to safeguard the safety of the pedestrian …

Models and languages

- How can we (precisely) capture behavior?
  - We may think of languages (C, C++), but *computation model* is the key
- Common computation models:
  - Sequential program model
    - Statements, rules for composing statements, semantics for executing them
  - Communicating process model
    - Multiple sequential programs running concurrently
  - State machine model
    - For control dominated systems, monitors control inputs, sets control outputs
  - Dataflow model
    - For data dominated systems, transforms input data streams into output streams
  - Object-oriented model
    - For breaking complex software into simpler, well-defined pieces

Models vs. languages

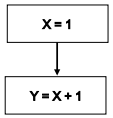
- Computation models describe system behavior
  - Conceptual notion, e.g., recipe, sequential program
- Languages capture models
  - Concrete form, e.g., English, C
- Variety of languages can capture one model
  - E.g., sequential program model → C,C ++, Java
- One language can capture variety of models
  - E.g., C++ → sequential program model, object-oriented model, state machine model
- Certain languages better at capturing certain computation models



Text versus Graphics

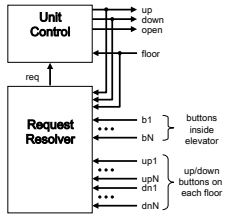
- Models versus languages not to be confused with text versus graphics
  - Text and graphics are just two types of languages
    - Text: letters, numbers
    - Graphics: circles, arrows (plus some letters, numbers)

```
X = 1;
Y = X + 1;
```



Introductory example: An elevator controller

- Simple elevator controller
  - *Request Resolver* resolves various floor requests into single requested floor
  - *Unit Control* moves elevator to this requested floor

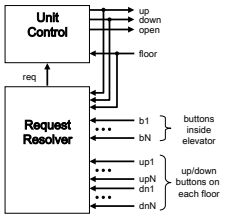


Partial English description

"Move the elevator either up or down to reach the requested floor. Once at the requested floor, open the door for at least 10 seconds, and keep it open until the requested floor changes. Ensure the door is never open while moving. Don't change directions unless there are no higher requests when moving up or no lower requests when moving down…"

Introductory example: An elevator controller

- Try capturing in a sequential programming model like C



Inputs: int floor; bit b1..bN; up1..upN-1; dn2..dnN;
Outputs: bit up, down, open;
Global variables: int req;

```
void UnitControl() {
    up = down = 0; open = 1;

    while (1) {
        while (req == floor);
        open = 0;

        if (req > floor){ up = 1; }
        else {down = 1;}

        while (req != floor);
        up = down = 0;
        open = 1;
        delay(10);
    }
}
```

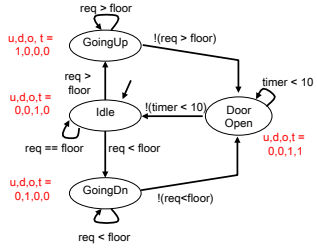
```
void RequestResolver() {
    while (1) {
        •••
        req = ...
        •••
    }
}
```

```
void main() {
    Call concurrently:
    UnitControl() and
    RequestResolver()
}
```

** You might have come up with something having even more if statements*

Finite-state machine (FSM) model

- Trying to capture this behavior as sequential program is a bit awkward
- Instead, we might consider an FSM model, describing the system as
  - Possible states
    - *Idle*, *GoingUp*, *GoingDn*, *DoorOpen*
  - Possible transitions from one state to another based on input
    - req > floor, req < floor, etc.
  - Actions that occur in each state
    - up = 1, down, open, and timer_start = 0



u is up, d is down, o is open
t is timer_start

Formal definition

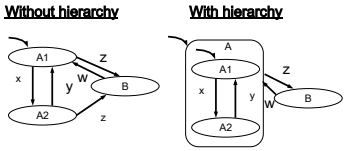
- Formal FSM definition M = (X, Y, S, δ, λ, so)
  - X is the input alphabet
  - Y is the output alphabet
  - S is a finite set of states
  - δ is the transition function, δ : X x S→S
    - Given and input and state, what is the next state
  - λ is the output function, λ : S → Y
    - Mealy FSM, λ : X x S → Y
  - so is the initial state

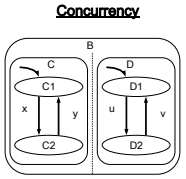
- HLFSM
  - Convert later to a FSM + D

Covered in Earlier Lectures

HCFSM and the Statecharts language

- Hierarchical/concurrent state machine model (HCFSM)
  - Extension to state machine model to support hierarchy and concurrency
  - States can be decomposed into another state machine
    - *With hierarchy* has identical functionality as *Without hierarchy*, but has one less transition (*z*)
    - Known as OR-decomposition
  - States can execute concurrently
    - Known as AND-decomposition

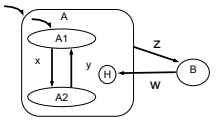




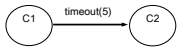
HCFSM and the Statecharts language

- Statecharts
  - Graphical language to capture HCFSM
- Numerous additional constructs available to improve state capture
  - history
    - Enter most recently visited when return instead of initial state
  - timeout
    - transition with time limit as condition
    - transition taken if source state active for defined time limit
  - many others - joins, forks, conditional, selections, …

History



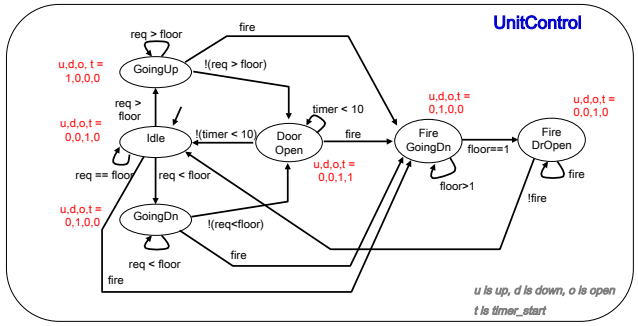
Timeout



UnitControl with *FireMode*

Without Hierarchy

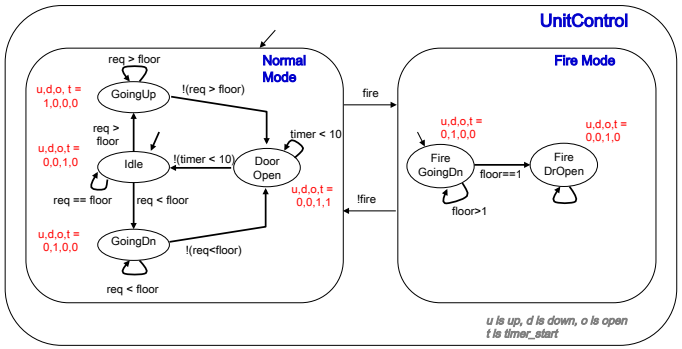
- FireMode - When *fire* is true, move elevator to 1st floor and open door



UnitControl with *FireMode*

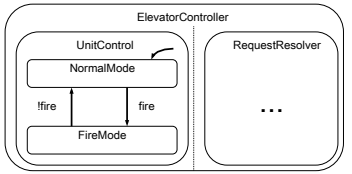
Without Hierarchy

- FireMode - When *fire* is true, move elevator to 1st floor and open door



UnitControl with *FireMode*

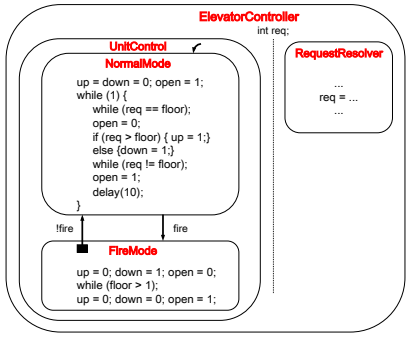
- Also add concurrency to our model



Program-state machine model (PSM)

HCFSM plus sequential program model

- Program-State Machine (PSM)
  - Extends state machine to allow use of sequential code or FSM to define state actions
  - Includes hierarchy and concurrency extensions of HCFSM
  - Stricter hierarchy than HCFSM used in Statecharts
- Examples
  - SpecCharts: extension of VHDL SpecC: extension of C



Black square originating within FireMode indicates ! fire is a *transition-on-completion* taken only if the condition is true and program state is complete

Summary

- Computation models are distinct from languages
- Finding appropriate model to capture embedded system is an important step
  - Model shapes the way we think of the system
  - Language should capture model easily
    - Ideally should have features that directly capture constructs of model
- Many choices
  - Sequential program model is popular
  - State machine models good for control
    - Extensions like HCFSM provide additional power
  - Concurrent process model for multi-task systems
    - Communication and synchronization methods exist, scheduling is critical
  - Dataflow model good for signal processing