Embedded Systems Design: A Unified Hardware/Software Introduction

Chapter 8: State Machine and Concurrent Process Model

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
- Real-Time Systems

Introduction

- Describing embedded system's processing behavior
  - Can be extremely difficult
    - Complexity increasing with increasing IC capacity
      - Past: washing machines, small games, etc.
        - Hundreds of lines of code
      - Today: TV set-top boxes, Cell phone, etc.
        - Hundreds of thousands of lines of code
    - Desired behavior often not fully understood in beginning
      - Many implementation bugs due to description mistakes/omissions
  - English (or other natural language) common starting point
    - Precise description difficult to impossible
    - Example: Motor Vehicle Code – thousands of pages long...

An example of trying to be precise in English

- 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) The provisions of this section shall not relieve a pedestrian from the duty of using due care for his or her safety. No pedestrian shall suddenly leave a curb or other place of safety and walk or run into the path of a vehicle which is so close as to constitute an immediate hazard. No pedestrian shall unnecessarily stop or delay traffic while in a marked or unmarked crosswalk.
    - (c) The provisions of subdivision (b) shall not relieve a driver of a vehicle from the duty of exercising due care for the safety of any pedestrian within any marked crosswalk or within any unmarked crosswalk at an intersection.
  - All that just for crossing the street (and there's much more)!

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

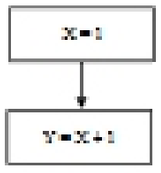
Models vs. languages

- 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;
```



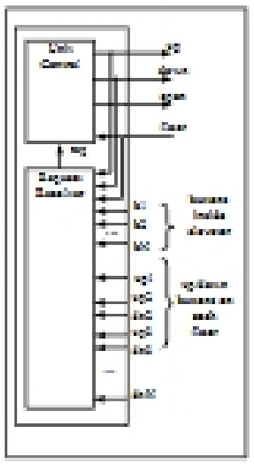
Introductory example: An elevator controller

- Simple elevator controller
  - *Request Resolver* resolves various floor requests into single requested floor
  - *Unit Control* moves elevator to this requested floor
- Try capturing in C...

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..."

System interface



Elevator controller using a sequential program model

Sequential program model

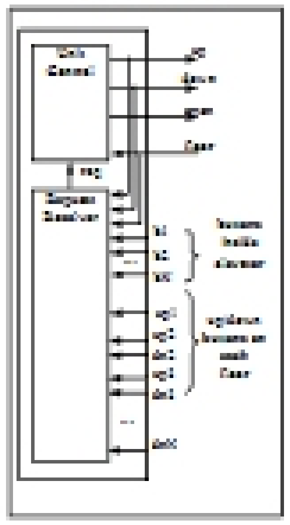
```
Inputs: int floor; bit b1..bN; up1..upN-1; dn2..dnN;
Outputs: bit up, down, open;
Global variables: int req;

void UnitControl()                    void RequestResolver()
{                                     {
  up = down = 0; open = 1;              while (1)
  while (1) {                             ...
    while (req == floor);                 req = ...
    open = 0;                             ...
    if (req > floor) { up = 1;}         }
    else {down = 1;}
    while (req != floor);             void main()
    up = down = 0;                    {
    open = 1;                           Call concurrently:
    delay(10);                            UnitControl() and
  }                                       RequestResolver()
}                                     }
```

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

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..."

System interface

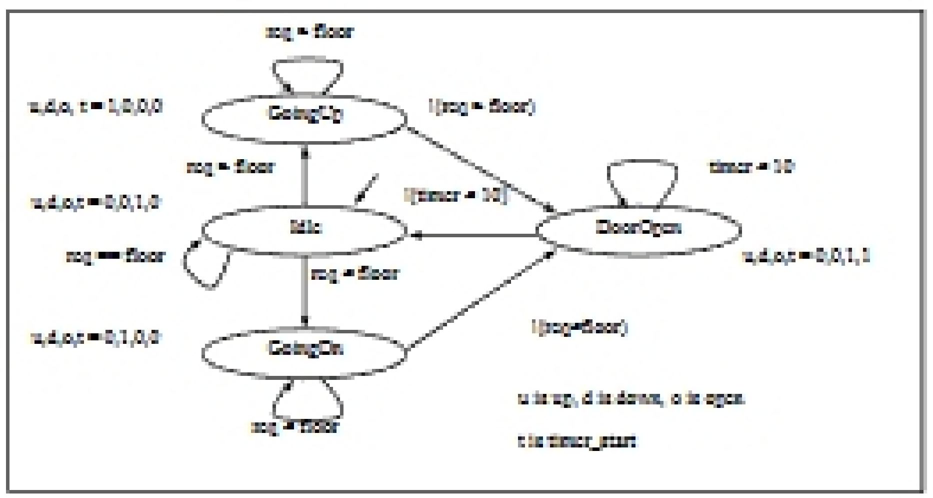


Finite-state machine (FSM) model

- 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
    - E.g., *Idle, GoingUp, GoingDn, DoorOpen*
  - Possible transitions from one state to another based on input
    - E.g., req > floor
  - Actions that occur in each state
    - E.g., In the *GoingUp* state, u,d,o,t = 1,0,0,0 (up = 1, down, open, and timer_start = 0)
- Try it...

Finite-state machine (FSM) model

***UnitControl* process using a state machine**



Formal definition

- An FSM is a 6-tuple $F<S, I, O, F, H, s_0>$
  - S is a set of all states $\{s_0, s_1, \ldots, s_l\}$
  - I is a set of inputs $\{i_0, i_1, \ldots, i_m\}$
  - O is a set of outputs $\{o_0, o_1, \ldots, o_n\}$
  - F is a next-state function ($S \times I \rightarrow S$)
  - H is an output function ($S \rightarrow O$)
  - s_0 is an initial state
- Moore-type
  - Associates outputs with states (as given above, H maps $S \rightarrow O$)
- Mealy-type
  - Associates outputs with transitions (H maps $S \times I \rightarrow O$)
- Shorthand notations to simplify descriptions
  - Implicitly assign 0 to all unassigned outputs in a state
  - Implicitly AND every transition condition with clock edge (FSM is synchronous)