State Machines
State Machines

- State machines provide a convenient way to model the dynamic behavior of systems
  - Many applications naturally are represented with a set of discrete states
  - Complicated systems can be broken down into multiple hierarchical state machines

- State machines can be implemented in hardware or software
  - We’ll show how to implement them in software
State machines in hardware

• A state machine consists of
  – a state memory
  – some output logic
  – some excitation logic that determines the next state

• At each tick of the clock, the system changes state depending on the inputs to the excitation logic and the current state
  – While in a state, the outputs are held steady (for a Moore machine) and are just a function of the current state
State machines in software

• We represent the state with a single variable called the state variable

• We simulate the regular ticking of the clock with an infinite loop that runs at regular intervals

• Within one iteration of this loop, we determine the current state, set the outputs for this state, and determine the next state

• In C, this can be done with a switch statement, or a set of if-then-else statements, inside a loop
Generic state machine pseudo code

Set initial value of state variable
Do all other initialization
WHILE (TRUE) DO
  wait until start of next clock period

  IF (STATE = STATE1) THEN
    do outputs for this state
    determine next value of STATE

  ELSE IF (STATE = STATE2) THEN
    do outputs for this state
    determine next value of STATE

  ELSE IF (STATE = STATE3) THEN
    do outputs for this state
    determine next value of STATE

  :
Example – elevator controller

• The elevator goes between two floors, F1 and F2

• The controller has two control inputs:
  – S is a sensor signal that is asserted when the elevator has arrived at a floor
  – B is a button signal that is asserted when the elevator is supposed to go to the other floor

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>At floor 2</td>
<td>command motor off</td>
</tr>
<tr>
<td>DN</td>
<td>Going down...</td>
<td>command motor down</td>
</tr>
<tr>
<td>F1</td>
<td>At floor 1</td>
<td>command motor off</td>
</tr>
<tr>
<td>UP</td>
<td>Going up...</td>
<td>command motor up</td>
</tr>
</tbody>
</table>

![Elevator Controller Diagram](image-url)
Pseudo code

Initialize STATE to F1
WHILE (TRUE) DO
  IF (STATE = F1) THEN
    Command motor off /* actions for this state */
    IF (B) THEN /* next state logic */
      STATE = UP
  ELSE IF (STATE = UP) THEN
    Command motor to go up /* actions for this state */
    IF (S) THEN /* next state logic */
      STATE = F2
  ELSE IF (STATE = F2) THEN
    Command motor off /* actions for this state */
    IF (B) THEN /* next state logic */
      STATE = DN
  ELSE IF (STATE = DN) THEN
    Command motor to go down /* actions for this state */
    IF (S) THEN /* next state logic */
      STATE = F1
  ELSE
    invalid state - try to recover
Elevator controller (continued)

• Hardware connections
  – PT0, PT1: control H-bridge to drive motor
  – PT2: button input B
  – PT3: sensor input S

• Loop time interval
  – Use RTI system
  – Use interval of 1/8 sec

• Assume
  – PT1:PT0 = 1:0 drives motor to go up
  – PT1:PT0 = 0:1 drives motor to go down
/* State machine program
   Bill Hoff
   *********************************************/
#include <hidef.h>  /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */

// Define our states
#define F1  0   // at floor 1
#define UP  1   // going up
#define F2  2   // at floor 2
#define DN  3   // going down

void main(void) {
    char state;

    /* Set up RTI system for the slowest count rate.
       The Osc-clock is divided by (N+1)*2^(M+9), where
       N = 0..15, M=0..7
       So the slowest rate is when N=15, M=7.
       The 8 MHz clock is divided by 16x2^16 = 2^20 = 1048576.
    */
    RTICTL = 0x7f;  // RTI will time out about every 1/8 sec
    DDRT = 0x03;    // PT1,PT0 are outputs to H-bridge

    EnableInterrupts;
state = F1; // assume that we start at floor 1
for(;;) {
    // Wait till RTI flag is set
    while (! (CRGFLG & 0x80)) ; // RTIF is bit 7
    CRGFLG = 0x80; // clear flag
    switch (state) {
        case F1: // At floor 1
            PTT = 0x00; // Turn motor off
            if (PTT & 0x04) // Check (B) button pushed
                state = UP; // Next state is: go up
            break;
        case UP: // Going up
            PTT = 0x02; // Command motor to go up
            if (PTT & 0x08) // Check (S) if we are at floor
                state = F2; // Next state is: at floor F2
            break;
        case F2: // At floor 2
            PTT = 0x00; // Turn motor off
            if (PTT & 0x04) // Check if button pushed
                state = DN; // Next state is: go down
            break;
        case DN: // Going down
            PTT = 0x01; // Command motor to go down
            if (PTT & 0x08) // Check (S) if we are at floor
                state = F1; // Next state is: at floor F1
            break;
        default: // invalid state
            state = F1; // should reset?
    }
    _FEED_COP(); /* feeds the dog */
} /* loop forever */
Issues

• This was a simple example and there are some factors you would need to deal with in a real system, such as

• Initialization and resetting
  – The elevator does not necessarily start at floor 1
  – You could use another sensor to figure out its true location
  – Or, you could have a separate initialization sequence of states to put it into a known location

• Transition from “at floor” to “moving”
  – Assume elevator is at a floor and you command the motor to start moving
  – The elevator may not leave the floor (ie., S is still asserted) before the next iteration; therefore you would incorrectly deduce that the elevator already reached its destination
  – Fixes?
Example – combination lock

- A user must enter four numbers to open a combination lock.
- After all four numbers have been entered, the system asserts either the OPEN signal (if the correct combination was entered) or the ERR signal (if any of the numbers was incorrect).
- Note that the machine must wait until all four digits have been entered before asserting either OPEN or ERR.

- OPEN is asserted in state S4.
- ERR is asserted in state S4A.
- There is an asynchronous RESET signal.
Example – traffic light controller

• Intersection has stoplights for
  – North-south traffic (NR, NY, NG)
  – East-west traffic (ER, EY, EG)

• Sensors
  – NS asserted if a car is waiting to go north or south
  – EW asserted if a car is waiting to go east or west
Example – traffic light controller

• **Outputs**
  – NR, NY, NG, ER, EY, EG

• **Inputs**
  – NS, EW

• **Outputs should sequence through**
  – NR, EG
  – NR, EY
  – NR, ER
  – NG, ER
  – NY, ER
  – NR, ER
  – (and then repeat)
Example – traffic light controller

- A stoplight is green for minimum time TG1
  - After TG1, if a car is waiting in the other direction, switch to other direction green
  - If no car waiting, stay green for a total maximum time TG2

- Yellow is on for time TY

- Both reds are on for time TR
Timing

• We will need a timer to time how long we are in each state

• We will use a variable called $t$, and increment it every time through the main iteration loop
  – We just compare $t$ to TG1, TG2, TY, or TR
  – The values TG1, TG2, etc are specified in terms of main loop counts

• Let’s say that we use the RTI system to time the main loop, at say a rate of 8 Hz
  – Then a time of say, 15 seconds, corresponds to $8 \times 15$ counts
State diagram

<table>
<thead>
<tr>
<th>State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>NG ER (1st period)</td>
</tr>
<tr>
<td>S1</td>
<td>NG ER (2nd period)</td>
</tr>
<tr>
<td>S2</td>
<td>NY ER</td>
</tr>
<tr>
<td>S3</td>
<td>NR ER</td>
</tr>
<tr>
<td>S4</td>
<td>NR EG (1st period)</td>
</tr>
<tr>
<td>S5</td>
<td>NR EG (2nd period)</td>
</tr>
<tr>
<td>S6</td>
<td>NR EY</td>
</tr>
<tr>
<td>S7</td>
<td>NR ER</td>
</tr>
</tbody>
</table>

- Timer counter $t$ is reset when transitions are made
- Since action happens on a transition, this is a Mealy machine

Links are labeled with: expression; action
Traffic light system (continued)

- Hardware connections
  - PT5:PT0: outputs to lights
  - PT6:PT5: inputs from sensors

// Define masks for the output bits
#define NG  0x01    // PT0
#define NY  0x02    // PT1
#define NR  0x04    // PT2
#define EG  0x08    // PT3
#define EY  0x10    // PT4
#define ER  0x20    // PT5

// Define masks for the input bits
#define NS  0x40    // PT6
#define EW  0x80    // PT7
Traffic light system (continued)

• State declarations

// Define our states
#define S0 0   // North green, east red (1st period)
#define S1 1   // North green, east red (2nd period)
#define S2 2   // North yellow, east red
#define S3 3   // North red, east red
#define S4 4   // North red, east green (1st period)
#define S5 5   // North red, east green (2nd period)
#define S6 6   // North red, east yellow
#define S7 7   // North red, east red

• Time interval declarations (assume RTI period is 8 Hz)

// Define times (in units of RTI timer counts)
#define TG1 15*8   // 1st green period = 15 sec
#define TG2 30*8   // 2nd green period = 30 sec
#define TY 5*8     // Yellow = 5 sec
#define TR 3*8     // Both red = 3 sec