State machines
State machines

- Known as Finite State Machines (FSM)
- Mathematical model of computation where system has a single state
- Triggering conditions can change that state
- FSMs are defined completely by both their states and the transitions between them

Events
State machines

State

- The system only exists in one state at a time
- State persists through time
- Certain conditions can change the state to another state
  - These are specific to the current state
State machines

Transitions

- Events trigger transitions between states
- A combination of events can be used
- Transitions are all mutually exclusive
- At any given time there must be a valid transition for a state
  - If no transition is explicitly stated, an implied loopback transition exists
town

up

down button

down button E

up button

up button event
State machines

Benefits

• Provides a formal way to reason about a system
  – Allows for testing before writing any code
• Can be easily visualized
• Are language independent
• States are only dependent on current state and current inputs
State machines

When to use

- Can be used whenever there are a finite set of states for the system
  - Car transmission
  - Stoplight
  - Vending machine
  - Toaster oven
  - Video games
State machines
Use in the SeaSlug

- Transmission protocol
  - Mission management
  - Parameter management
- Operating state
  - Handling errors/system faults
- Calibration
  - Rudder
  - Radio controller
State machines

Diagrams

condition2
action2

state_1

condition1
action1

state_2
typedef enum { STATE_1, STATE_2 } SystemState;
static SystemState state;
{
    switch (state) {
    case STATE_1:
        // Default:
        if (condition1) {
            Action1();
            state = STATE_2;
        }
        break;
    case STATE_2:
        if (condition2) {
            Action2();
            state = STATE_1;
        }
    }
}
State machines

Integrating

Example

typedef enum { STATE_1, STATE_2 } SystemState;
static SystemState state;
int main (void) {
    // Initialize system

    // Event loop
    while (1) {
        // State machine
        switch (state) {
            ...
        }
    }
}
Guard condition

4up & f00

4up & f00

4up & 1000

f00
Lock / Bomb 3 1 4

NR

1R

UL

3A

2R

L

No 600m

1 2 3 4 5 6 7 8 9

L 4
CMPE-013/L

Bounce (or Hardware)

Maxwell James Dunne
Bounce

- Digital I/O
- A/D
- Timers
- Debouncing
Stotic Events;

main() {
    ...
}

3

Timer1 Init {
    Events = BCEL();
}
Debouncing
Last event = Down
1) Start with one button
B TN 4
2) Copy & paste
3) User input
Lighting all

00000.
Button 1
\texttt{event 1 = 1 UP}

Button 2
\texttt{event 1 = 2 down}
\texttt{if (2 down \&\& event \_1)}
LATE
TRISE 1 0
LATE = 0x01
LATE = 'c'
1023
1222
OLED

999
1000
19×8
968
SFR, LATE