Event-Driven Programming and State Machines

Cyrus Bazeghi
Winter 2010
Traditional Program Structures

if/else switch

interrupt

while (condition)
    do something

end

start

end
Programming Embedded Systems
(1.2)

1. Asynchronous - any input at any time in any order
2. Simultaneous - inputs treated
3. Sequence of inputs and outputs are unchangeable and re-orderable
4. No “end” or “exit”
Programming Embedded Systems (2.2)

inputs - sensors, switches, cables, trans/couplers, user input devices

outputs - LEDs/display, motor, switch output

in general, change smoothly, physically
Events and Services Framework (1.4)

- Conceptual Framework
- Gives a framework

- Excellent method for event driven systems

- Emphasize design first
- Not simply into implementation

- Define low level functions
Events and Services Framework (2.4)

Rule #1

Tackle tasks broken down into 2 fundamental classes

1) Events detected
2) Services

Post
What is an Event?
What Happens with Noise?
Add Hysteresis

![Graph showing hysteresis with low and high thresholds, signal plus noise, and on/off states.]
Events and Services Framework (3.4)

Corollary to Rule #1

1. Keep event detection as short as possible
2. Make non-blocking
Events and Services Framework (4.4)

Complete program structure

```
while (1) {
    test for event; round robin
    service these events
    usually done with a state machine
}
```
Announcements

1) email me with partner grade 1(suck) → 10(excellent)
2) Do what we bad at j in partnership
3) Lab report due in my office (E2719) or to John by 6pm
4) < help
State Machines (1.4)

- Description of an abstract machine.

- At any point in time, in one state.

- Next state is what is next state:
  \[ P(\text{input}, \text{current state}) = \text{next state} \]

- Idealized - instant state change.
State Machines (2.4)

- Useful tool for describing behavior of event driven programs
- Allows you to explore design before implementation
- Perfect fit E.D.L.
State Machines (3.4)

Event (off) → State A

Event (on) → Action

Event = Timer

Guard = Light State (on/off)

CMPE 118 – Intro. to Mechatronics
State Machines (4.4)
Finite State Diagram (FSD) or State Transition Diagram (STD)
Example: Combination Lock

Combination = 2-1-8
Example: Smart Combination Lock

Could make many changes to make more “robust”
Quiz #1

Time
Def/Light
Popcorn
Start
Clear
Open

Problem #8 of ch5
SES – Software Events and Services

• **Initialize SES** by calling: `SES_Init();`
• **Event-Checking Functions**
  - prototyped with the parameter `EVENT_PARAM`  
    `MyEventChecker(EVENT_PARAM)`  
  - return unsigned char = 0 if event not detected  
    return unsigned char ≠ 0 if event detected  
  - to pass data from the Event-Checking Function to its Service Function, use `SET_SHARED_BYTE_TO(foo)`; or  
    `SET_SHARED_WORD_TO(foo)`;  
  - Data passed between functions must be static

• **Service Functions**
  - prototyped with the parameter: `SERVICE_PARAM`  
    `MyServiceFunction(SERVICE_PARAM)`  
  - no return value  
  - to read the data passed from the Event-Checking Function, use `GET_SHARED_BYTE()` if it’s 8-bit data, or  
    `GET_SHARED_WORD()` if it’s 16-bit data.
SES – Software Events and Services

- **Register** each Event Function and Service Functions in pairs:

  ```
  SES_Register(MyEventChecker, MyServiceFunction);
  ```

- **Start the process** by calling

  ```
  SES_HandleEvents();
  ```

```c
while(1)
{
    SES_HandleEvents();
}
```
Timer Library

- 8 timers available to you (0-7)

- Initialize timer functionality by calling the function:
  \texttt{TMR_Init()}

- Initialize a timer by calling the function:
  \texttt{TMR_InitTimer(0,\text{TIME\_INTERVAL});}
  \quad \text{\texttt{TIME\_INTERVAL = number of timer ticks (1 tick = 4.1ms)}}

- Check to see if the timer has expired by calling:
  \texttt{TMR_IsTimerExpired(timer number)};

- Clear the timer flags by calling:
  \texttt{TMR_ClearTimerExpired(timer number)};
Roach Library

• You need to initialize the functions by calling `RoachInit();`

• Functions available for controlling the motors (see documentation for full details):
  ```
  LeftMtrSpeed(x);  RightMtrSpeed(x);
  ```
  – x is a number from -10 (reverse) to 10 (forward)

• Functions available for checking the bumpers:
  ```
  uchar ReadBumpers();
  ```

• Function available for reading the light level:
  ```
  uchar LightLevel();
  ```
Pseudo-Code (PDL)

- PDL = Program Design Language

- Pseudo-code is written in ENGLISH.
- Doesn’t use the syntax of any particular programming language.
- It is a written, low-level exploration of an implementation of an algorithm.
- It can form the first level of comments for your code.
Questions?