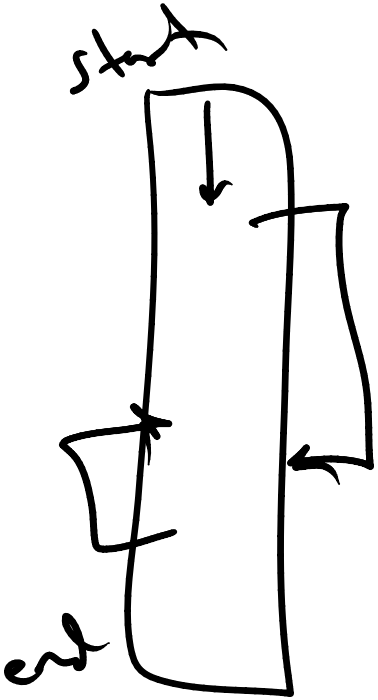
Event-Driven Programming and State Machines

Cyrus Bazeghi

Winter 2010

Traditional Program Structures



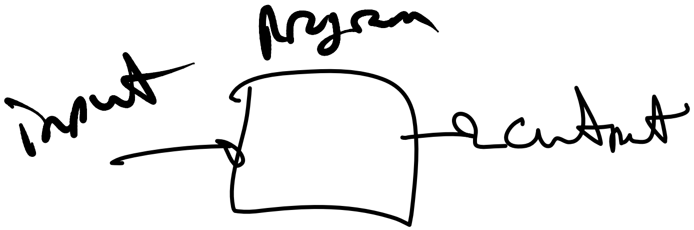
if/else

switch

for

while

interrupts



```
while(1){
    do nthg
}
```

Programming Embedded Systems (1.2)

1. Asynchronous — any input @ any time
   any order

2. Simultaneous — inputs & outputs

3. Sequence of inputs & outputs are unknowable & re-orderable

4. No "end" or "exit"

Programming Embedded Systems (2.2)

inputs — sensors, switches, voltages
timers/counters, user input
keypad

outputs — leds / display
move something
switch on/off
in general change something
physical

Events and Services Framework (1.4)

- Conceptual framework
  - gives a formalize methodology
- excellent method for event driven programs & F.S.M
- emphasize design first not jumping into implementation
- define low level functions

Events and Services Framework (2.4)

Rule #1

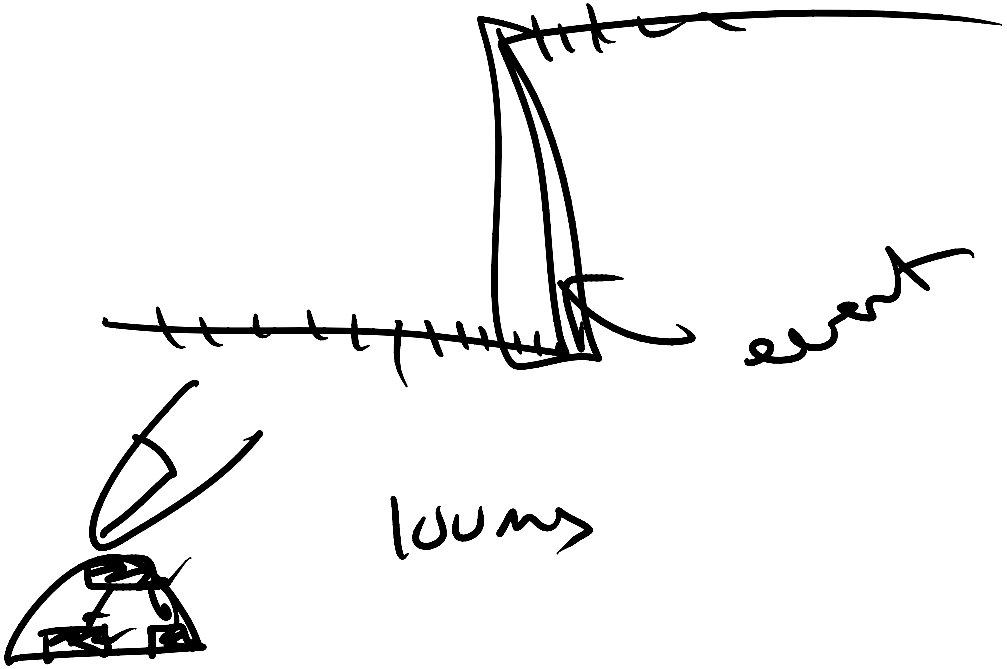
tasks break down into 2 fundamental classes

(1) event detect
(2) services

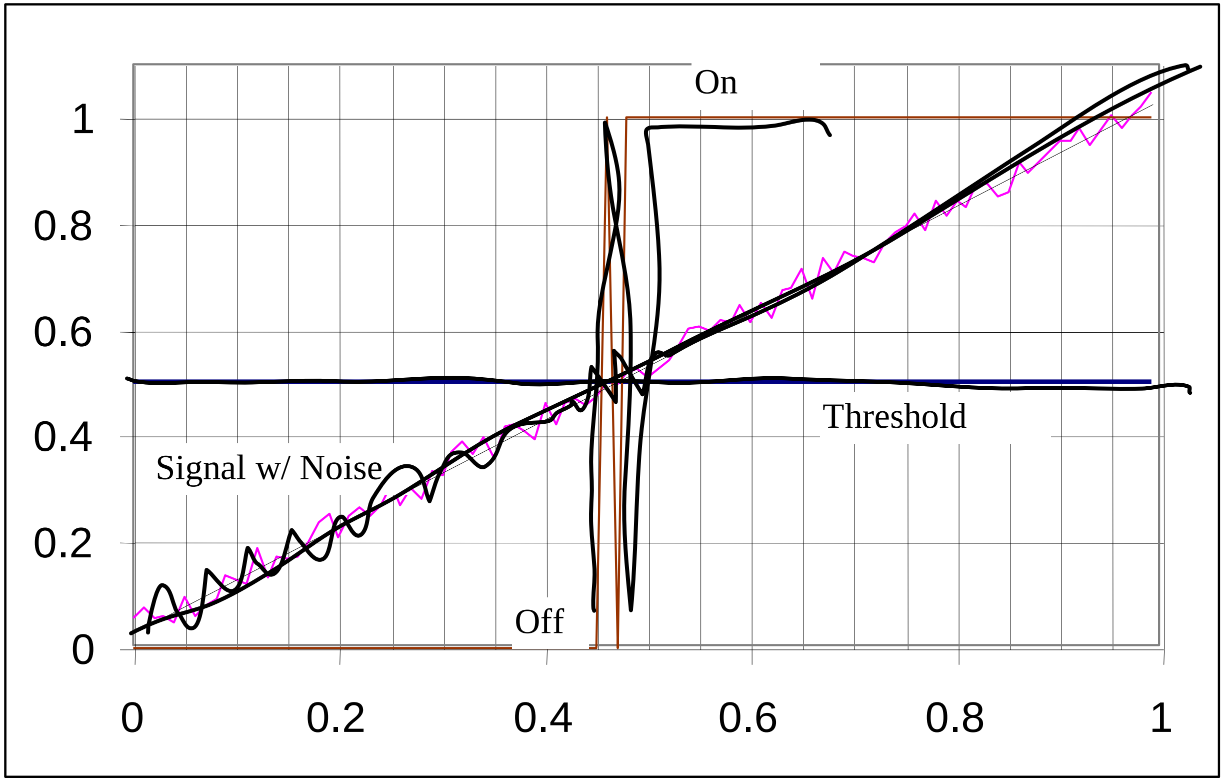
} non-blocking & fast

What is an Event?

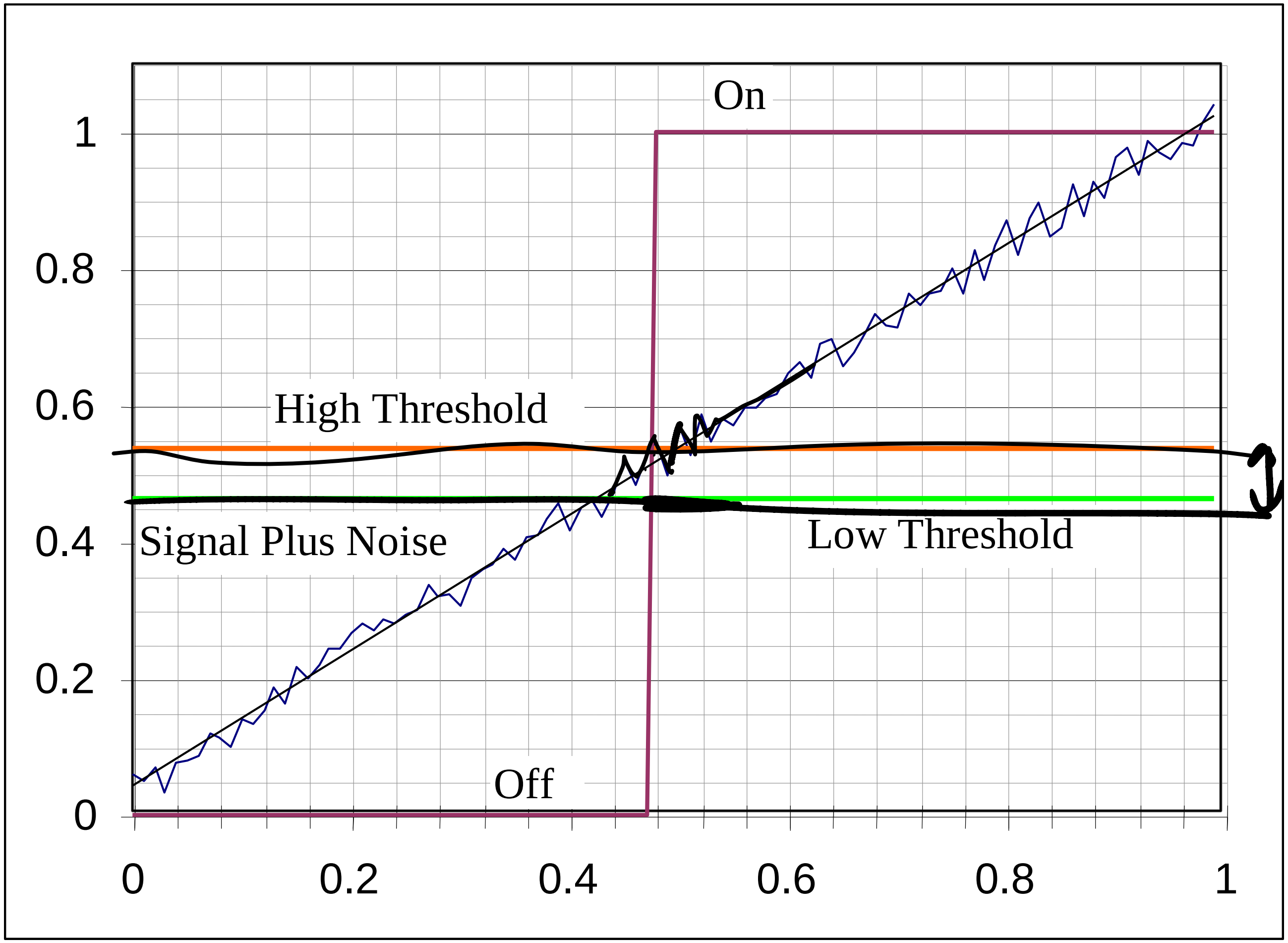
Event "detectable change"



What Happens with Noise?



Add Hysteresis



noise margin

Events and Services Framework (3.4)

Corollary to Rule #1

- keep event detectable & service routines as short as possible

- make non-blocking.

Events and Services Framework (4.4)

Complete Program Structure

```
Init HW/SW

while(1) {
    test for event       } round robin
    service those events
        normally done with
            a state machine
}
```

Announcements

1) email me with partner grade
1(suck) → 10(excellent)

2) Do what are bad at ; in partnership

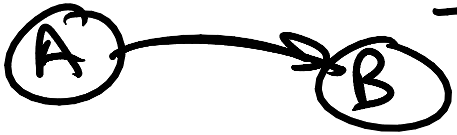
3) Lab report due in my office (E2319)
or to John by 6pm

4) C help

State Machines (1.4)

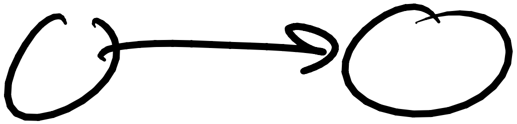
+ Desciption of an abstract machine

+ At any point time in <u>one</u> state



+ Next state is what
F(input, current state) = nextstate

+ idealized - instant state changes



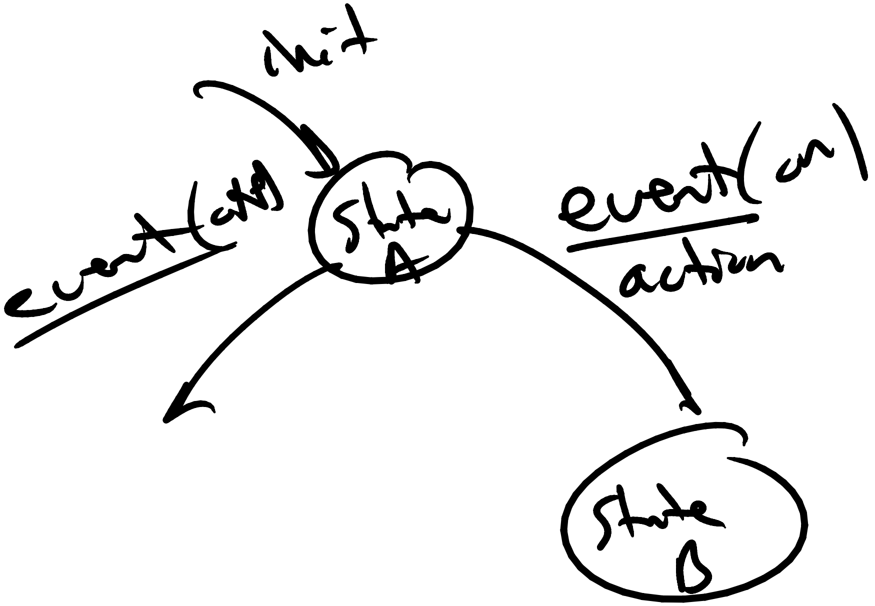
State Machines (2.4)

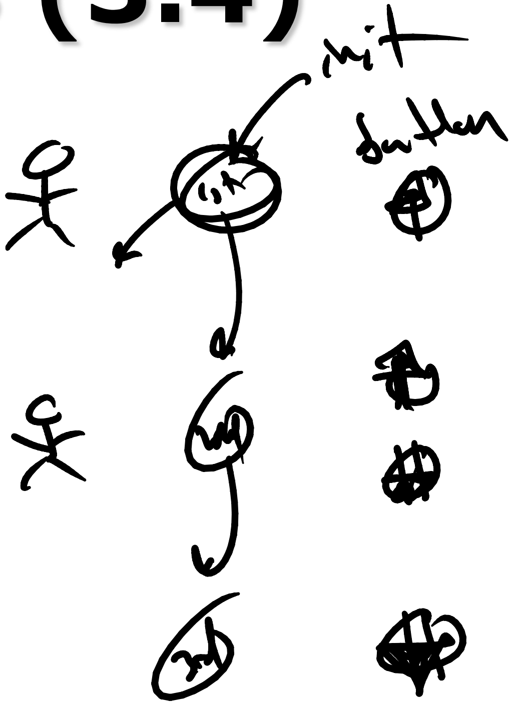
+ Useful tool for desc. behavior of event driven program

+ Allows you to explore design before implementing

+ perfect fit E.D.P.

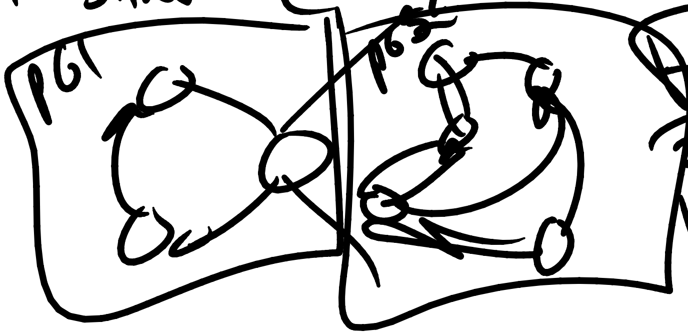
State Machines (3.4)





event = timer

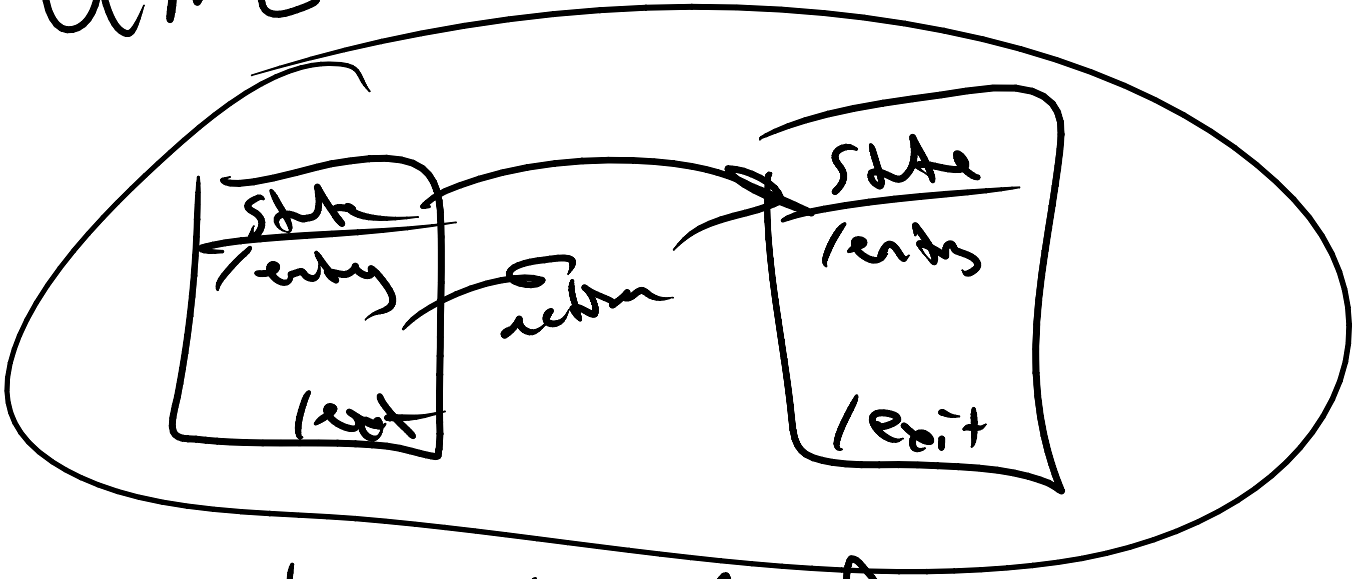
guard = light state (on/off)



F.S.M ≡ M.F.S.M

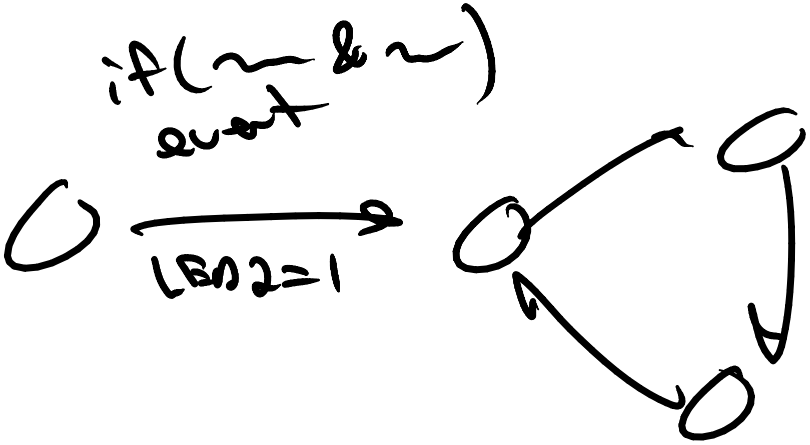
State Machines (4.4)

UML –



Visio Prof.

Finite State Diagram (FSD) or State Transition Diagram (STD)

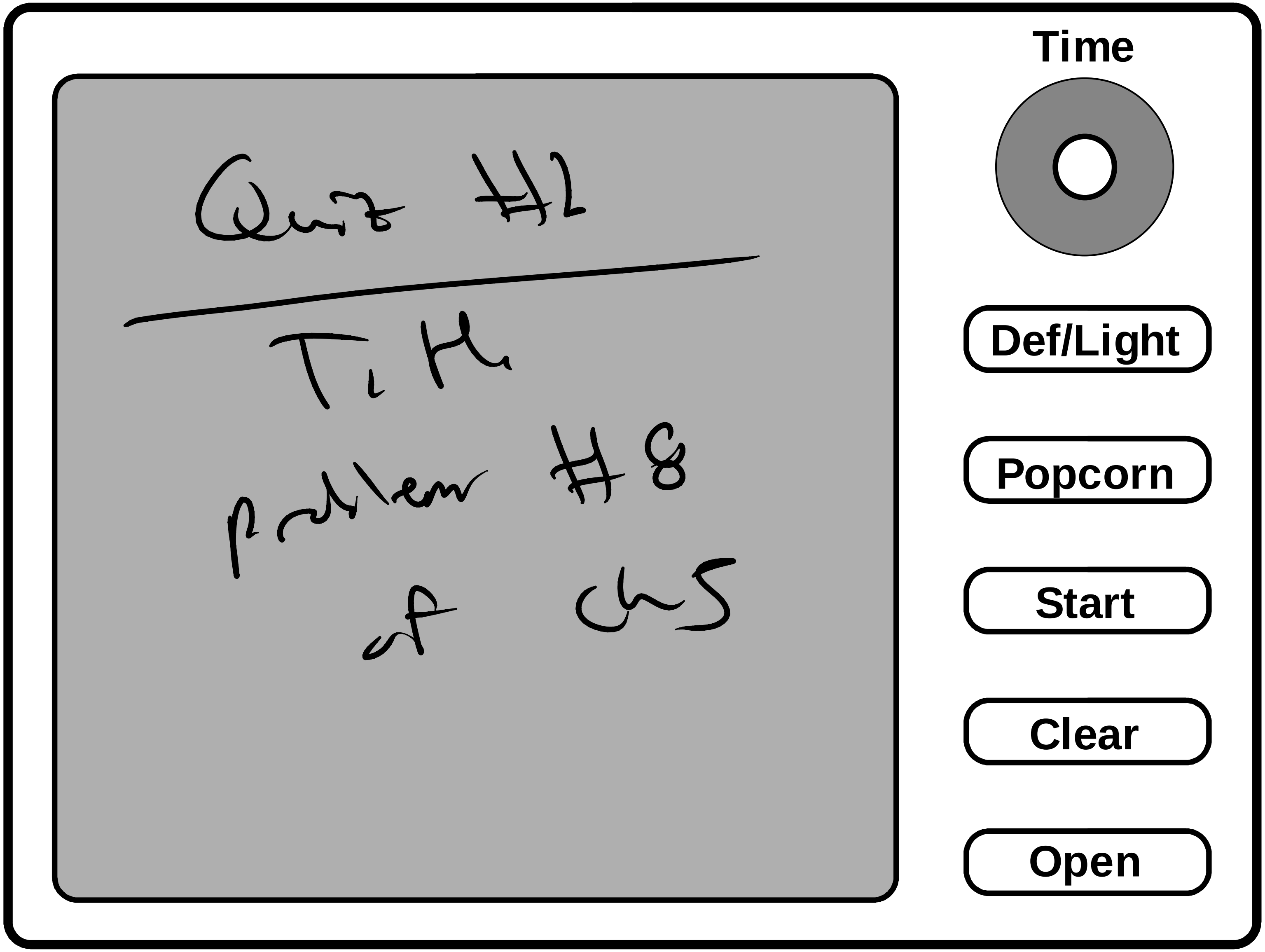


Example: Combination Lock

Combination = 2-1-8

Example: Smart Combination Lock

Could make many changes to make more “robust”



Quiz #2

Title

problem #8

of ch5

Time

Def/Light

Popcorn

Start

Clear

Open

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

- **<u>Register</u>** each Event Function and Service Functions in pairs:

  SES_Register(MyEventChecker, MyServiceFunction);

- **<u>Start the process</u>** by calling **SES_HandleEvents()** in an infinite loop.

```
while(1)
{
        SES_HandleEvents();
}
```

Timer Library

- 8 timers available to you (0-7)
- Initialize timer functionality by calling the function:
  TMR_Init()
- Initialize a timer by calling the function:
  TMR_InitTimer(0,TIME_INTERVAL);
  - TIME_INTERVAL = number of timer ticks (1 tick = 4.1ms)
- Check to see if the timer has expired by calling:
  TMR_IsTimerExpired(timer number);
- Clear the timer flags by calling:
  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?