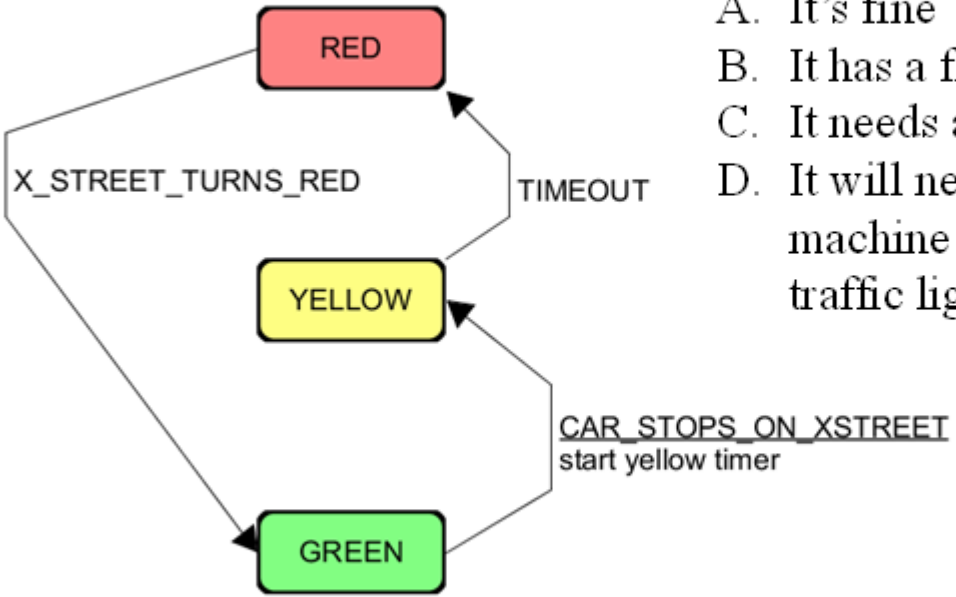
CMPE-013/L

Introduction to "C" Programming

Max Lichtenstein

Piazza Poll: How bad is this state machine for a traffic light?



A. It's fine
B. It has a flaw or two
C. It needs a lot of work
D. It will never work, a state machine can't run a traffic light

Roadmap

- Lab 7 overview
  - The State Machine
  - Static libraries
  - FreeRunningTimers
- Events and services review
- State Machine review
- BREAK
- State machine design:
  - What is a state, really?
  - What is an event, really?
- Alarm Clock design activity
  - Working in small groups
  - Max picks best design
  - Then, coding as a class

```
struct packed {
    int a : 2
    int b : 2
    using : 4
    int c : 8
}
```

16 bits

2 bytes

size of (packed) = 2

Notes/Announcements

- Office hours rescheduled for TODAY 2-3
- Battleboats partners Lab 9
  - Start picking them!
    - By next Tuesday please
    - Piazza is good
  - Solo option
- Extra Credit Lab 10?
  - Piazza poll after class //

Lab 7 Notes

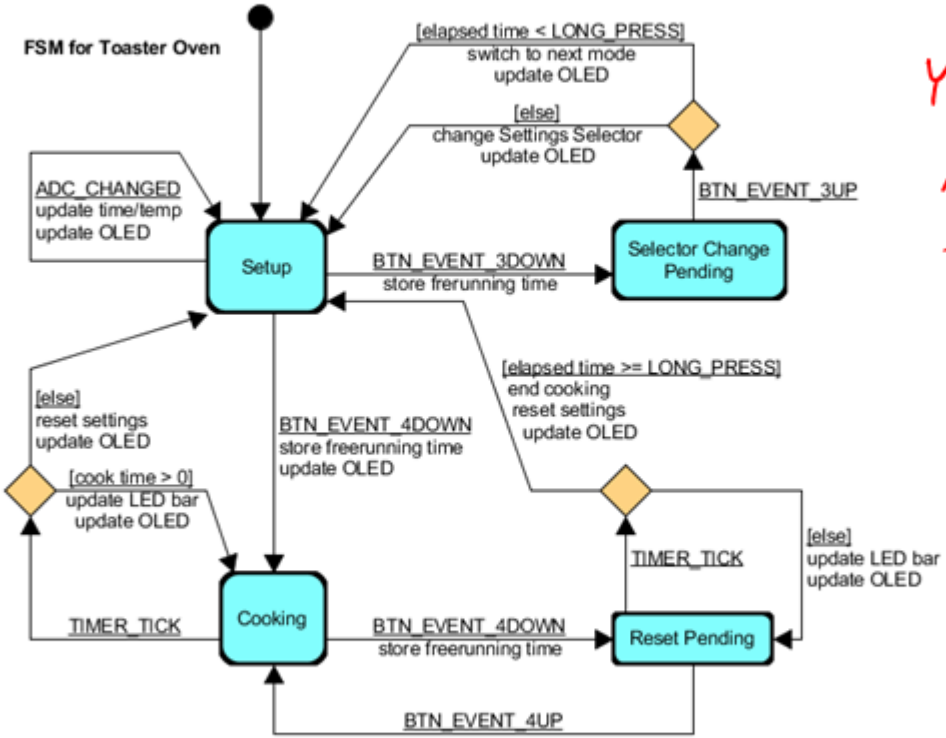
- Lab 7 due next Monday
  - No autochecker! (can still check submissions, but not code)

- This line:

> You will need to able to understand and reason about state machines to do well in this lab. Make sure you do the reading before attempting the lab. Use the example bounce FSM provided here to rewrite your Bounce code using a state machine implementation as a test of your understanding. Ask a TA/tutor for help in working through this and you'll understand much better how to implement this lab.

- Static library uses the version of buttons.h.
  "Too many args to ButtonsCheckEvent"

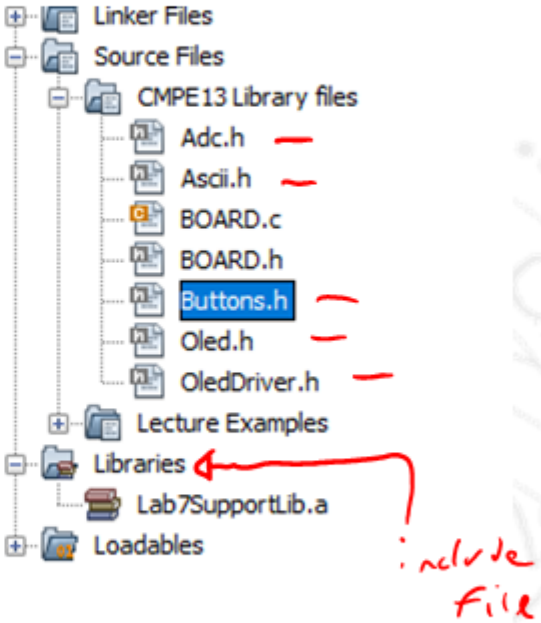
Lab 07 Overview



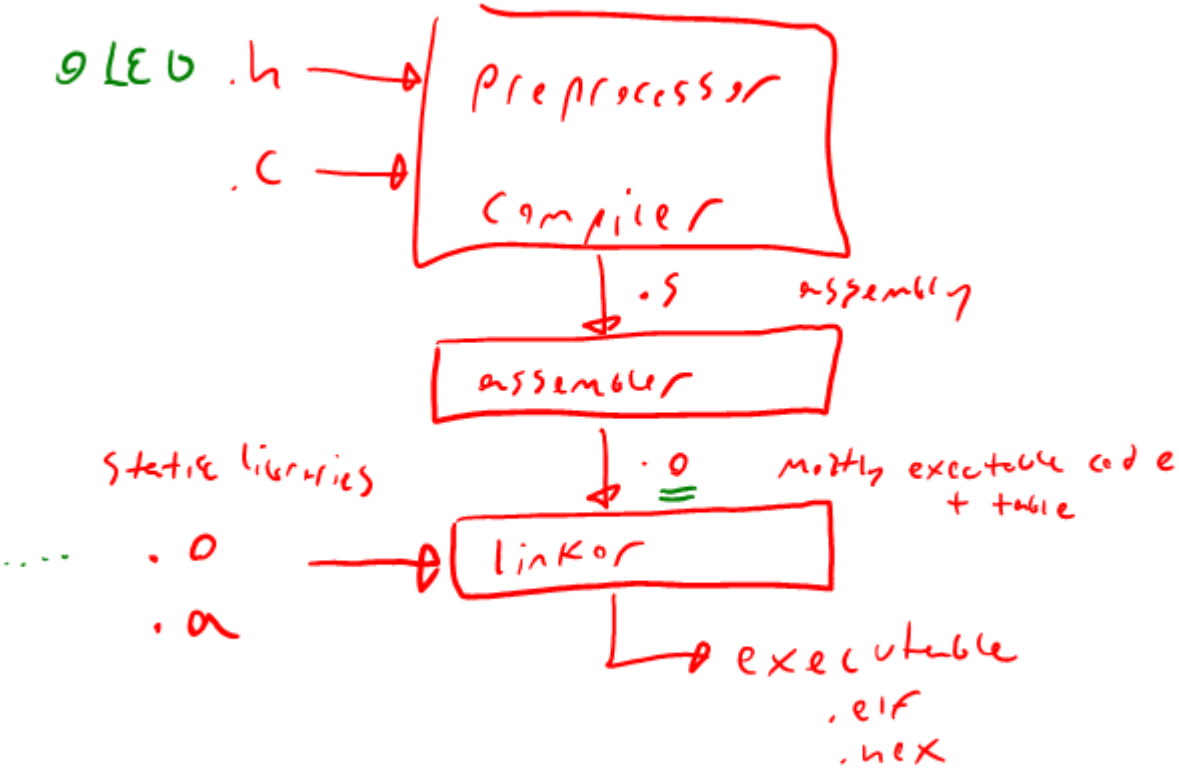
Your MAP

Static Libraries

- Don't need .c files!
- Essentially, .o files in a nice package
  - Hide source code ✓
  - Protect source code ✓
  - Make code portable ✓



C Toolchain:


OLED.h
.c
preprocessor
compiler
.s
assembly
assembler
.o
mostly executable code
+ table
static libraries
.o
.a
linker
executable
.elf
.hex

Static Libraries

Hex | Ascii

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  21 3C 61 72 63 68 3E 0A 2F 20 20 20 20 20 20 20  !<arch>./
00000010  20 20 20 20 20 20 20 20 31 35 32 36 35 36 39 39          15265699
00000020  38 34 20 20 30 20 20 20 20 20 20 30 20 20 20 20  84  0     0
00000030  30 20 20 20 20 20 20 20 36 34 30 20 20 20 20 20  0       640
00000040  20 20 60 0A 00 00 00 23 00 00 02 C4 00 00 02 C4    `....#...Ä...Ä
00000050  00 00 02 C4 00 00 02 C4 00 00 02 C4 00 00 0E F4  ...Ä...Ä...Ä...ô
00000060  00 00 19 00 00 00 19 00 00 00 19 00 00 00 19 00  ................
00000070  00 00 19 00 00 00 19 00 00 00 39 9C 00 00 39 9C  ..........9œ..9œ
00000080  00 00 42 F8 00 00 42 F8 00 00 42 F8 00 00 42 F8  ..Bø..Bø..Bø..Bø
00000090  00 00 42 F8 00 00 42 F8 00 00 42 F8 00 00 42 F8  ..Bø..Bø..Bø..Bø
000000A0  00 00 42 F8 00 00 42 F8 00 00 42 F8 00 00 52 D0  ..Bø..Bø..Bø..RÐ
000000B0  00 00 52 D0 00 00 52 D0 00 00 52 D0 00 00 52 D0  ..RÐ..RÐ..RÐ..RÐ
000000C0  00 00 52 D0 00 00 52 D0 00 00 52 D0 00 00 52 D0  ..RÐ..RÐ..RÐ..RÐ
000000D0  00 00 52 D0 41 64 63 49 6E 69 74 00 41 64 63 43  ..RÐAdcInit.AdcC
000000E0  68 61 6E 67 65 64 00 41 64 63 52 65 61 64 00 41  hanged.AdcRead.A
000000F0  64 63 48 61 6E 64 6C 65 72 00 5F 5F 76 65 63 74  dcHandler.__vect
00000100  6F 72 5F 64 69 73 70 61 74 63 68 5F 32 37 00 61  or_dispatch_27.a
00000110  73 63 69 69 00 42 4F 41 52 44 5F 49 6E 69 74 00  scii.BOARD_Init.
00000120  53 45 52 49 41 4C 5F 49 6E 69 74 00 42 4F 41 52  SERIAL_Init.BOAR
```

```
00000260  49 6E 76 65 72 74 65 64 00 4F 6C 65 64 44 72 69  Inverted.OledDri
00000270  76 65 72 53 65 74 44 69 73 70 6C 61 79 4E 6F 72  verSetDisplayNor
00000280  6D 61 6C 00 4F 6C 65 64 44 72 69 76 65 72 44 69  mal.OledDriverDi
00000290  73 61 62 6C 65 44 69 73 70 6C 61 79 00 4F 6C 65  sableDisplay.Ole
000002A0  64 44 72 69 76 65 72 55 70 64 61 74 65 44 69 73  dDriverUpdateDis
000002B0  70 6C 61 79 00 5F 4F 6C 65 64 50 75 74 42 75 66  play._OledPutBuf
000002C0  66 65 72 00 41 64 63 2E 6F 2F 20 20 20 20 20 20  fer.Adc.o/
000002D0  20 20 20 20 31 35 32 36 35 36 39 39 38 34 20 20      1526569984
000002E0  30 20 20 20 20 20 30 20 20 20 20 20 31 30 30 36  0     0     1006
000002F0  36 36 20 20 33 30 36 30 20 20 20 20 20 20 60 0A  66  3060      `.
00000300  7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00  .ELF............
00000310  01 00 08 00 01 00 00 00 00 00 00 00 00 00 00 00  ................
00000320  EC 03 00 00 01 10 FF 70 34 00 00 00 00 00 28 00  ì.....ÿp4.....(.
00000330  0F 00 0C 00 F8 FF BD 27 04 00 BE AF 21 F0 A0 03  ....øÿ½'..¾¯!ð .
00000340  00 00 02 3C 02 00 03 24 00 00 43 AC 00 00 02 3C  ...<...$..C¬...<
00000350  00 1C 03 3C 00 00 43 AC 00 00 02 3C 00 08 03 3C  ...<..C¬...<...<
00000360  00 00 43 AC 00 00 02 3C 00 03 03 3C 00 00 43 AC  ..C¬...<...<..C¬
```

Free Running Timer

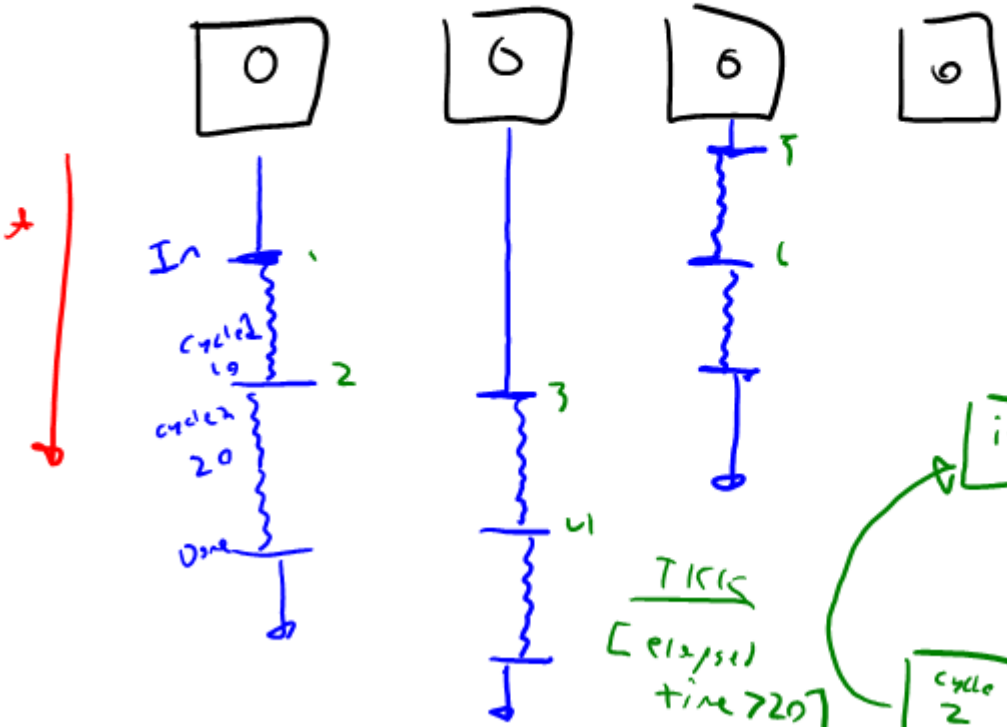
11.29 global
11.03 start time
.26 elapsed

- Used when you need lots of timers
- A way of using one hardware timer (ie, one ISR) to make lots of software timers
  - One ISR keeps track of a "global" time
  - Each process keeps track of its own start times (and other important times)
  - Each process can find its own elapsed time with:

```
myElapsedTime = globalTime - myStartTime
```

Smart Washing Machine x 4

Timer x 1



TICK

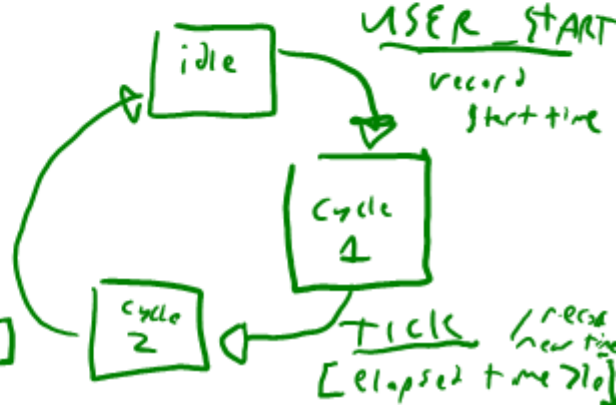
[elapsed time > 20]

```
if(global time % 100){
    set TICK=1}
```

global Time

global_tick_event

```
ISR () {
    global time ++;
    global_tick_
        event =
            TRUE
}
```

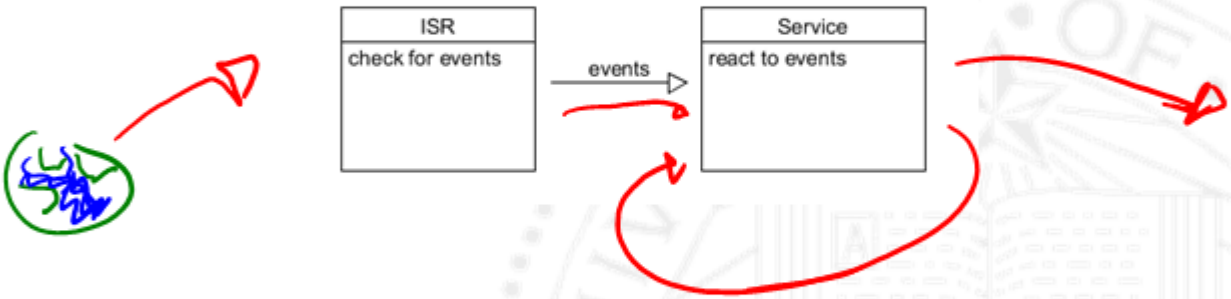


Events and Services review

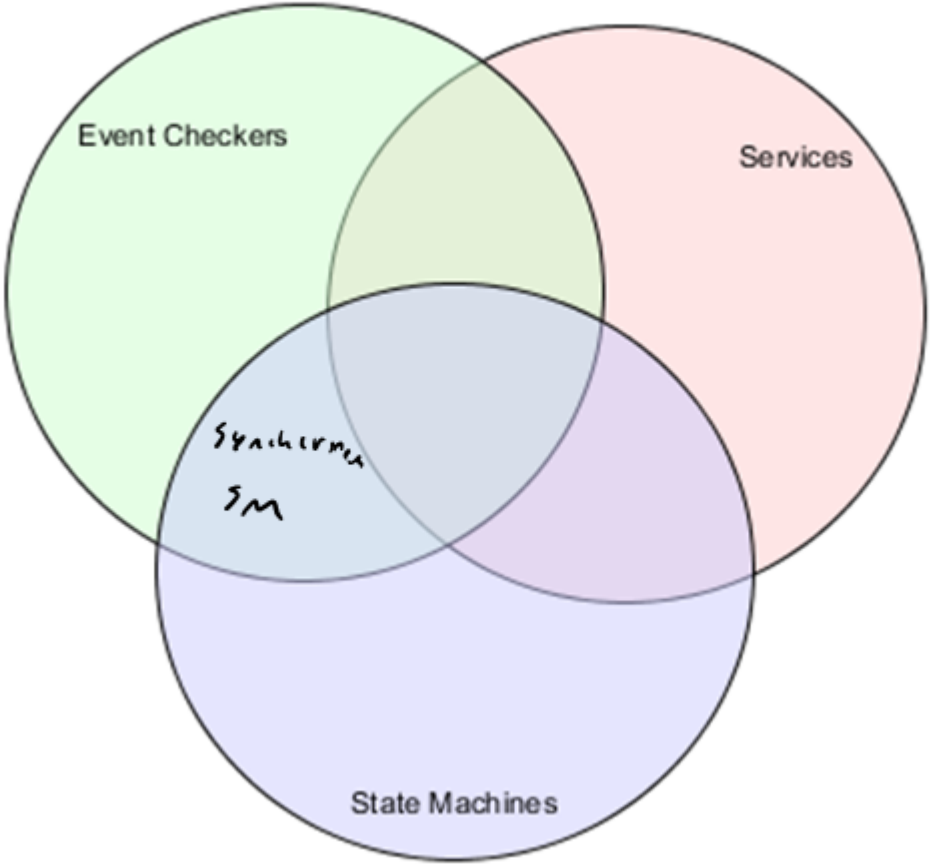
Events and Services

- Events:
  - Momentary
  - "Fed" to services
  - Checked by synchronous code Mostly
- Services:
  - Run when they receive events
  - Behavior depends on which event they receive
  - Often implemented as SMs
  - "Consume" event after completion

Events and Services

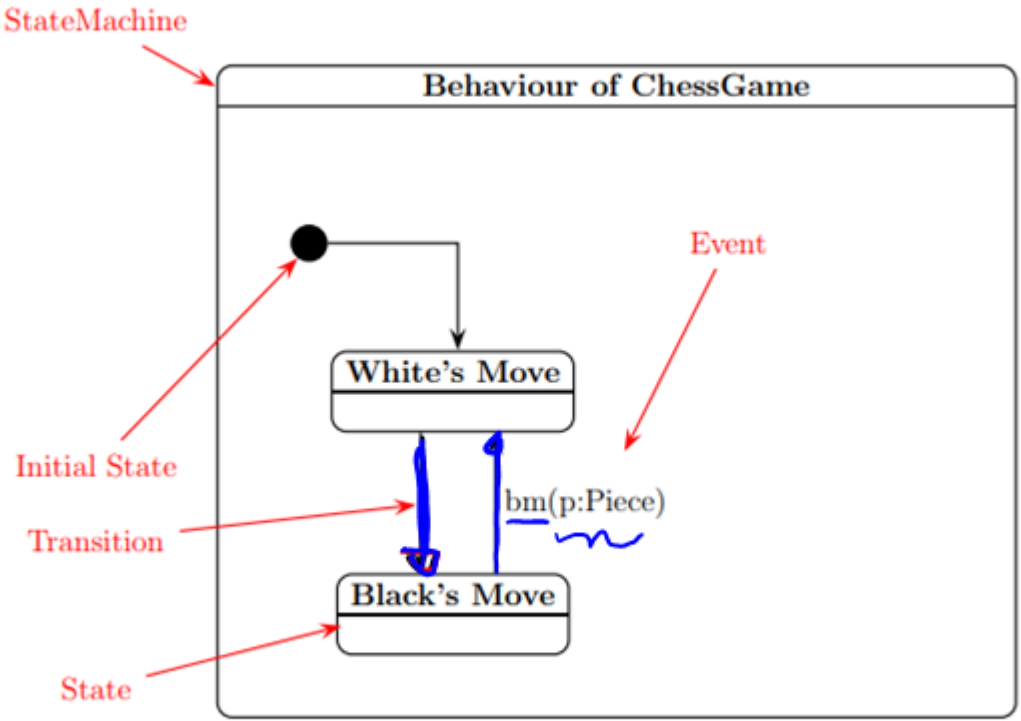


Events and Services



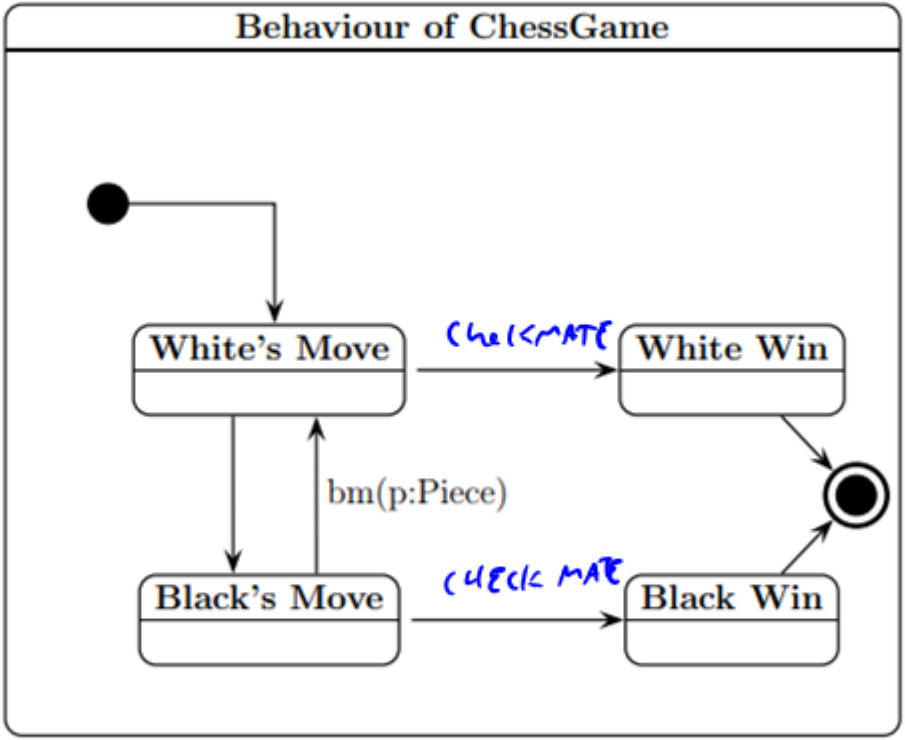
State Machine Review

Chess (1)



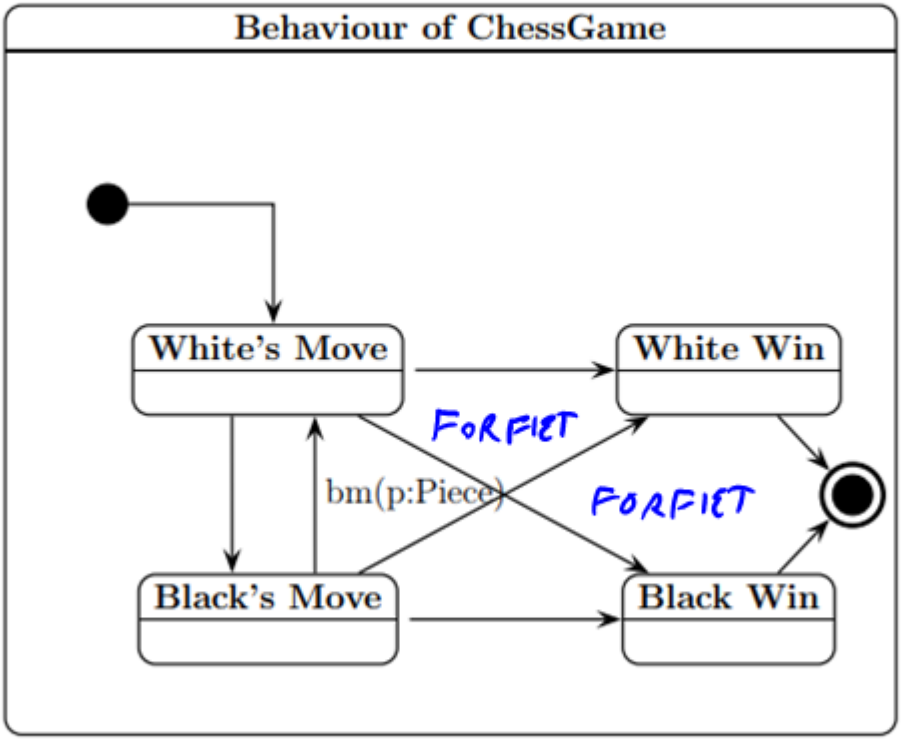
From *Formal Specification of Software: UML State Machines*
Bernhard Beckert, UNIVERSITÄT KOBLENZ-LANDAU

Chess (2)



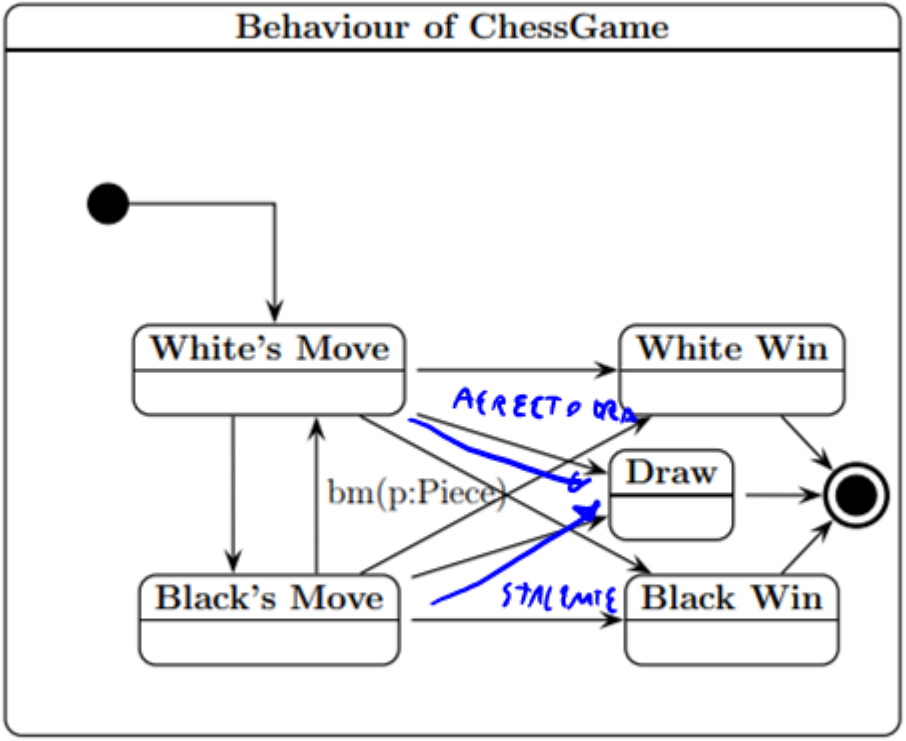
From *Formal Specification of Software: UML State Machines*
Bernhard Beckert , UNIVERSITÄT KOBLENZ-LANDAU

Chess (3)



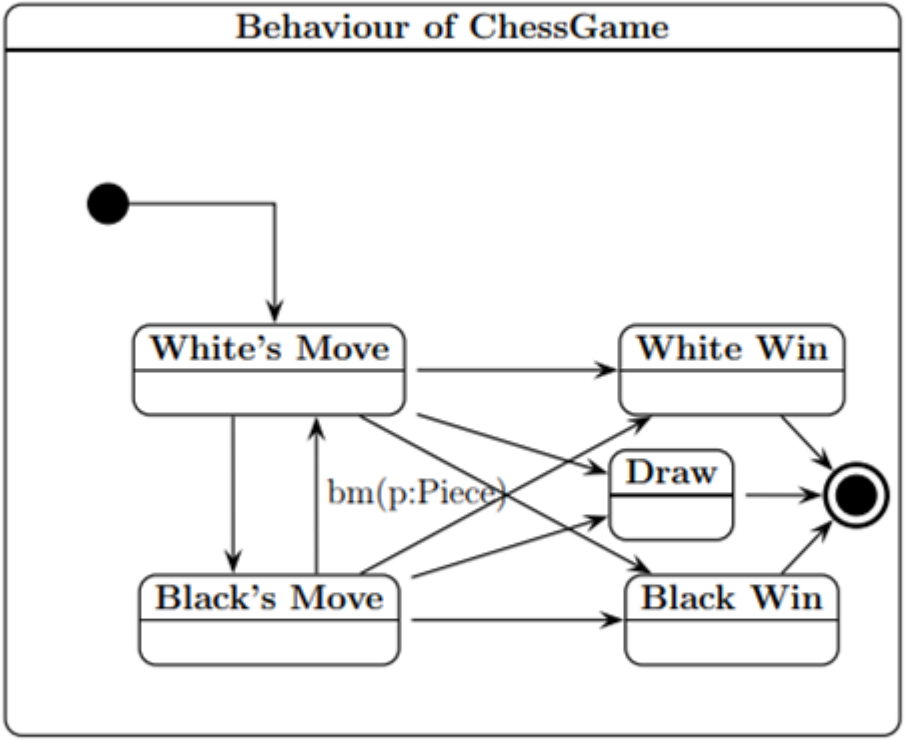
From *Formal Specification of Software: UML State Machines*
Bernhard Beckert , UNIVERSITÄT KOBLENZ-LANDAU

Chess (4)



From *Formal Specification of Software: UML State Machines*
Bernhard Beckert , UNIVERSITÄT KOBLENZ-LANDAU

Chess (4)



From *Formal Specification of Software: UML State Machines*
Bernhard Beckert , UNIVERSITÄT KOBLENZ-LANDAU

x -> Nextitem ∴ Not it null

BREAK

x

State Machine Design

Designing State Machines

- First, high level:
  - In English, what do you want to do?
    - "The device has three modes: Clock, stopwatch, and count-down timer. The user can switch between modes by pressing a button. Also, they can enter an "adjustment" mode where they use the knob to adjust the time"
  - Next, figure out some basic states (maybe 3 or 4)
  - Next, figure out what events are important

Designing State Machines (2)

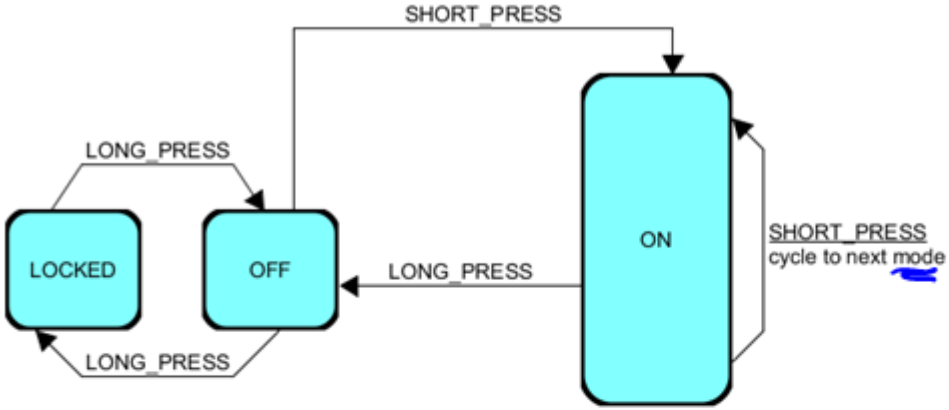
- Draw your basic states
  - Keep it simple, for now
- Draw some event transitions between them
  - Reflect: go through your state machine several times
  - Have someone else go through your state machine
  - What could work better? What does it still need?
- At this point, you will probably need to add some auxillary states

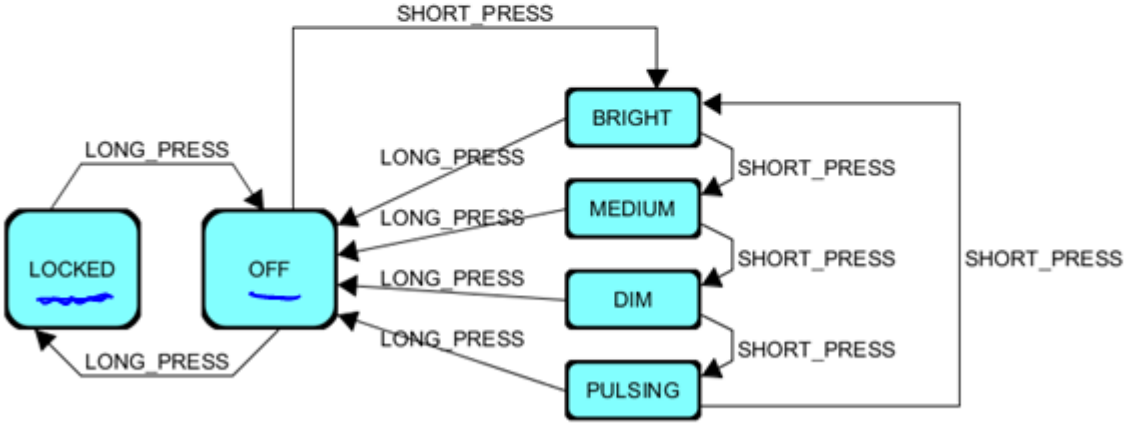
Designing State Machines (3)

- Iteratively add things to your state machine using the previous steps
  - If it starts to get complicated, pause and see what you can axe
- When your SM diagram is very nice and you like it, code it up!
  - Start with just 1 or 2 states. Keep it simple.
  - Add 1 state at a time, with transitions.
  - Test as you go!

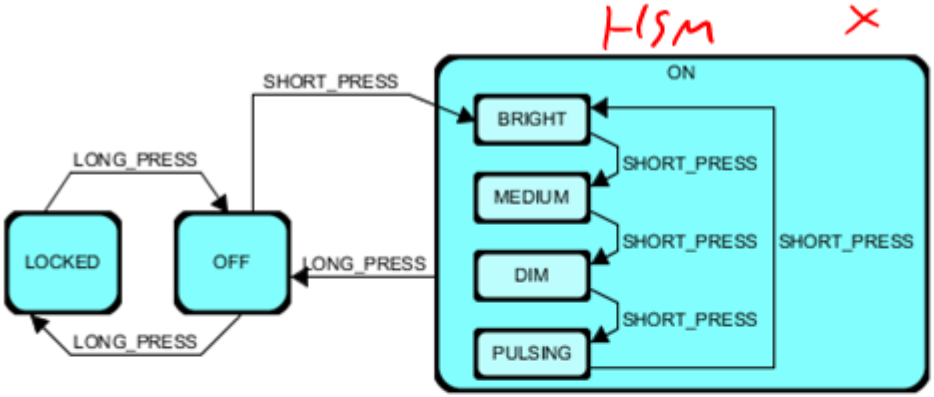
State Machine Design State Machine

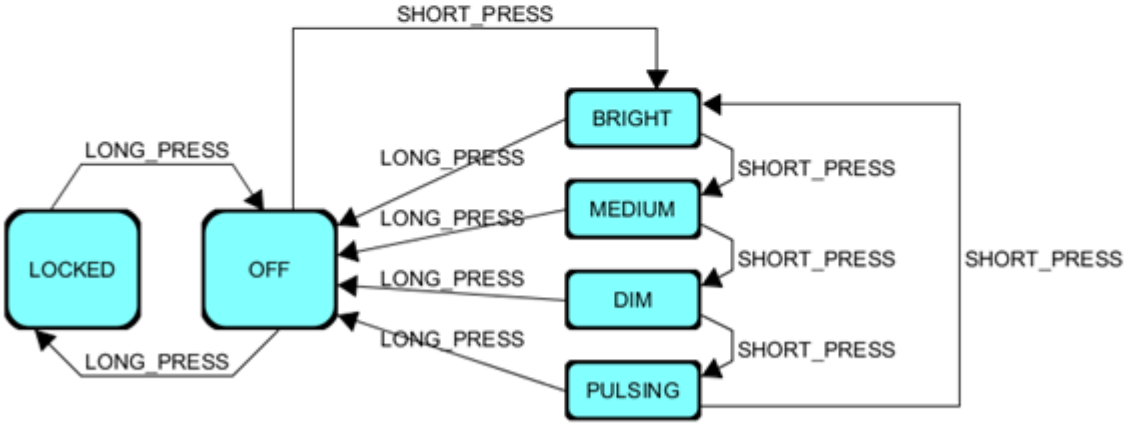
Picking States



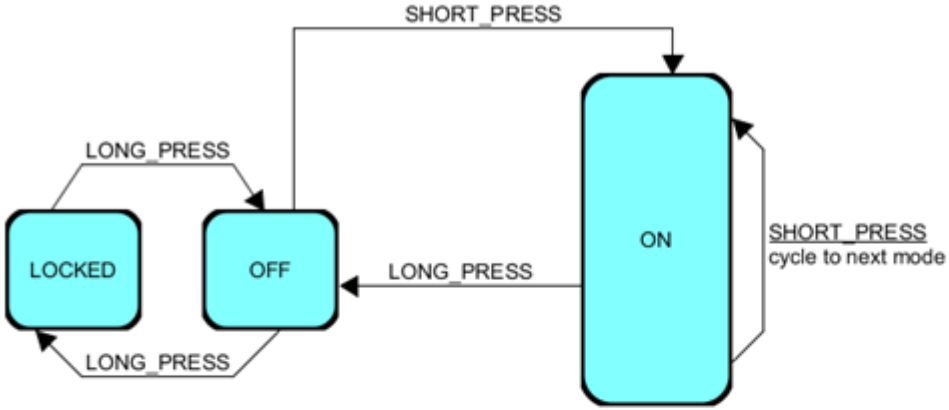


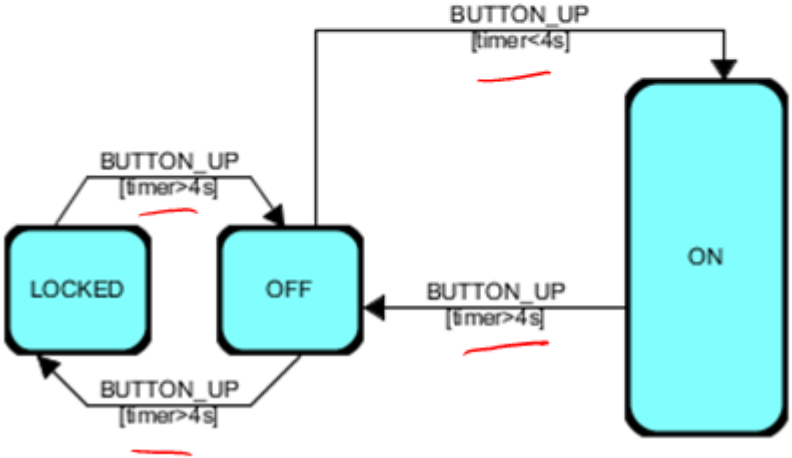
Picking States





Picking Events





SM Design Activity

Design Activity Agenda

1. Break into groups of 2-5 people (5-10 mins)
   1. On green paper, write names of participants
   2. Discuss how you want the UI to work. Write it down in English ✓
   3. List the states and events you need ✓
2. State Machine First Draft (5-10 mins)
   1. On green paper, draw your basic state machine. Get as detailed as you feel comfortable ✓
   2. MaxL will review the designs with the class ✓
3. State Machine Second Draft (5-10 minutes)
   1. Get new green paper ✓
   2. Revise design – Incorporate feedback, make clean and simple as possible
4. Final SM design as a class ✓
5. Coding as a class
   1. We'll see how far we get!

Design activity spec

- Combination Clock/Timer/Stopwatch
  - Clock keeps track of global time
    - Can be adjusted
  - Timer counts down from a value
    - User adjusts value, then presses a button to start
  - Stopwatch counts up from 0
    - Can be paused or reset
  - No two modes interfere with each other!
    - Each clock/time keeps ticking during other modes
- You design the UI