Introduction to Embedded Systems
Notes for EECS 124
Spring 2008
Edward A. Lee, Sanjit A. Seshia, and Claire Tomlin
eal@eecs.berkeley.edu, sseshia@eecs.berkeley.edu, tomlin@eecs.berkeley.edu
Electrical Engineering & Computer Sciences
University of California, Berkeley
February 25, 2008

Chapter 3

Concurrency

This chapter, yet to be written, gives an overview of concurrent models of computation and their application to embedded systems.

3.1 Interrupts

3.2 Threads

3.3 Process Networks

3.4 Dataflow

3.5 Synchronous/Reactive Systems

3.6 Discrete-Event Systems

Exercises

1. Figure 3.1 gives the sketch of a program that performs some function repeatedly for three seconds. The function is invoked by calling the procedure foo(). The program begins by setting up a timer interrupt to occur once per second (the code to do this setup is not shown). Each time the interrupt occurs, the specified interrupt service routine is called. That routine decrements a counter until the counter reaches zero. The main() procedure initializes the counter with value 3 and then invokes foo() until the counter reaches zero.

   (a) We wish to assume that the segments of code in the grey boxes, labeled **A**, **B**, and **C**, are

```
#include <avr/interrupt.h>
volatile uint16_t timer_count = 0;

// Interrupt service routine.
SIGNAL(SIG_OUTPUT_COMPARE1A) {

  if(timer_count > 0) {
    timer_count--;                    A
  }
}

// Main program.
int main(void) {
  // Set up interrupts to occur
  // once per second.
  ...

  // Start a 3 second timer.
  timer_count = 3;                    B

  // Do something repeatedly
   // for 3 seconds.
  while(timer_count > 0) {
    foo();                            C
  }
}
```

Figure 3.1: Sketch of a C program that performs some function by calling procedure foo() repeatedly for 3 seconds, using a timer interrupt to determine when to stop.

atomic. State conditions that make this assumption valid.

(b) Construct a state machine model for this program. The transitions in your state machine should be labeled with "guard/action", where the action can be any of **A**, **B**, **C**, or nothing. The actions **A**, **B**, or **C** should correspond to the sections of code in the grey boxes with the corresponding labels. You may assume these actions are atomic.

(c) Is your state machine deterministic? What does it tell you about how many times foo() may be invoked? Do all the possible behaviors of your model correspond to what the programmer likely intended?

Note that there are many possible answers. Simple models are preferred over elaborate ones, and complete ones (where everything is defined) over incomplete ones. Feel free to give more than one model.