CPES Assignment - II

- Adwait Thattey
UG3, CSE
S20170010004

Total 10 questions Done

Chapter 7 , Q3:
The following questions are about how to determine the function
f : (L, H) → {0, . . . , 2B − 1},
for an accelerometer, which given a proper acceleration x yields a digital number
f (x). We will assume that x has units of "g's," where 1g is the acceleration of
gravity, approximately g = 9.8meters/second2 .

a) (a) Let the bias b ∈ {0, . . . , 2B −1} be the output of the ADC when the accelerometer measures no proper acceleration. How can you measure b?

Answer:
We can place the accelerometer horizontally and there will be no gravity in the x-y plane

b) Let a ∈ {0, . . . , 2B − 1} be the difference in output of the ADC when the accelerometer measures 0g and 1g of acceleration. This is the ADC conversion of the sensitivity of the accelerometer. How can you measure a?

Ans: Place the accelerometer along the gravity axis

c) Suppose you have measurements of a and b from parts (3b) and (3a). Give an affine function mode for the accelerometer, assuming the proper acceleration is x in units of g's. Discuss how accurate this model is.

Ans: The affine function model is

$$f(x) = ax + b.$$

d) Given a measurement f (x) (under the affine model), find x, the proper acceleration in g's.

Ans: $x = (f(x) - b) / ɑ$

e) Suppose you have an ideal 8-bit digital accelerometer that produces the value f (x) = 128 when the proper acceleration is 0g, value f (x) = 1 when the proper acceleration is 3g to the right, and value f (x) = 255 when the proper acceleration is 3g to the left. Find the sensitivity a and bias b. What is the dynamic range (in decibels) of this accelerometer? Assume the accelerometer never yields f (x) = 0.

Ans: $D_{dB} = 20 \log_{10} (6/0.024) = 48_{dB}$.

Chapter 7, Q1:

Show that the composition $f \circ g$ of two affine functions f and g is affine/

Ans:

Assume $f(x) = a1x+b1$ and $g(x) = a2x+b2$

Then

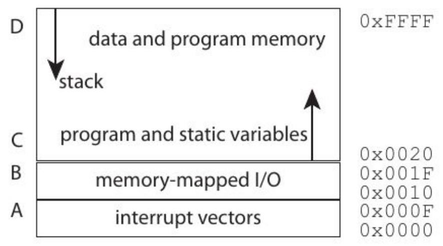
$(f \circ g)(x) = a1 (a2x+b2) + b1 \Rightarrow (a1a2)x + (a1b2+b1)$

which is an affine function $(f \circ g)(x) = a3x+b3$

where a3=a1*a2 and b3 = a1*b2 + b1

Chapter 9 , Q3:

Consider the C program and simplified memory map for a 16-bit microcontroller shown below. Assume that the stack grows from the top (area D) and that the program and static variables are stored in the bottom (area C) of the data and program memory region. Also, assume that the entire address space has physical memory associated with it.



```
#include <stdio.h>
#define FOO 0x0010
int n;
int* m;
void foo(int a) {
if (a > 0) {
n = n + 1;
foo(n);
}
}
int main() {
n = 0;
m = (int*)FOO;
foo(*m);
printf("n = %d\n", n);
}
```

You may assume that in this system, an int is a 16-bit number, that there is no operating system and no
memory protection, and that the program has been compiled and loaded into area C of the memory.

(a) For each of the variables n, m, and a, indicate where in memory (region A, B, C, or D) the variable will be stored.

Ans:

n and m will be in C, and a will be in D.

(b) Determine what the program will do if the contents at address 0x0010 is 0 upon entry.

Ans: The program prints 0 and exits.

(c) Determine what the program will do if the contents of memory location 0x0010 is 1 upon entry.

Ans: The stack gets overflowed and incorrect results will be produced

Chapter 11, Q4:

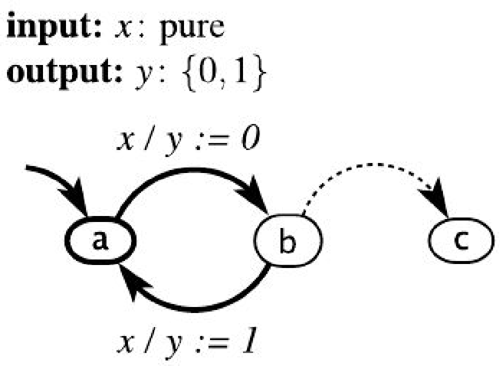
The producer/consumer pattern implementation in Example 11.13 has the draw-
back that the size of the queue used to buffer messages is unbounded. A program
could fail by exhausting all available memory (which will cause malloc to fail).

```
 int size = 0;
 pthread_cond_t sent = PTHREAD_COND_INITIALIZER;
 pthread_cond_t received = PTHREAD_COND_INITIALIZE // Procedure to
send a message.
 void send(int message) {
```

```
pthread_mutex_lock(&mutex);
while (size >= 5) {
pthread_cond_wait(&received, &mutex);
}
if (head == 0) {
head = malloc(sizeof(element_t));
head->payload = message;
head->next = 0;
tail = head;
} else {
tail->next = malloc(sizeof(element_t));
tail = tail->next;
tail->payload = message;
tail->next = 0;
}
size++;
pthread_cond_signal(&sent);
pthread_mutex_unlock(&mutex);
}
// Procedure to get a message.
int get() {
element_t* element;
int result;
pthread_mutex_lock(&mutex);
// Wait until the size is non-zero.
while (size == 0) {
pthread_cond_wait(&sent, &mutex);
}
result = head->payload;
element = head;
head = head->next;
free(element);
size--;
if (size < 5) {
pthread_cond_signal(&received);
}
pthread_mutex_unlock(&mutex);
return result;
```

```
}
```

Chapter 13: Q2:
Consider the following state machine:



(Recall that the dashed line represents a default transition.) For each of the following LTL formulas, determine whether it is true or false, and if it is false, give a Counterexample:

(a) x =⇒ Fb

Ans: true

(b) G(x =⇒ F(y = 1))

Ans: false.

(c) (Gx) =⇒ F(y = 1)

Ans: true

(d) (Gx) =⇒ GF(y = 1)

Ans: true

(e) G((b ∧ ¬x) =⇒ FGc)

Ans: true

(f) G((b ∧ ¬x) =⇒ Gc)

Ans: false

(g) (GF¬x) =⇒ FGc

Ans: false

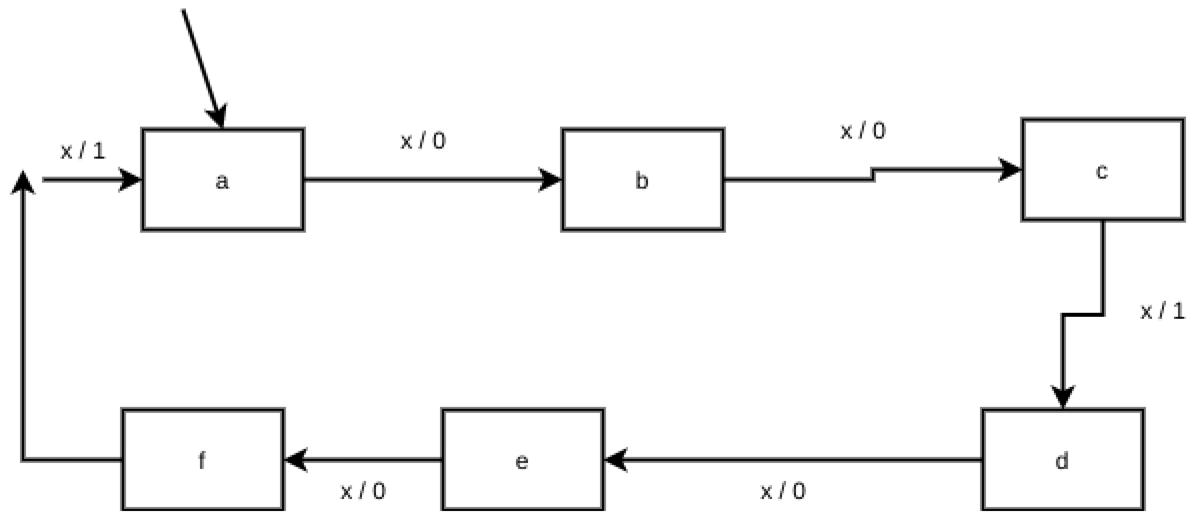
Chapter 14: Q7

Consider a state machine with a pure input x, and output y of type {0, 1}. Assume the states are States = {a, b, c, d, e, f }, and the initial state is a. The update function is given by the following table (ignoring stuttering):

$(currentState, input)$	$(nextState, output)$
(a, x)	$(b, 1)$
(b, x)	$(c, 0)$
(c, x)	$(d, 0)$
(d, x)	$(e, 1)$
(e, x)	$(f, 0)$
(f, x)	$(a, 0)$

(a) Draw the state transition diagram for this machine.

Ans



b) Ignoring stuttering, give all possible behaviors for this machine.

Ans:

The machine only stutters when input is absemnt. Thus we should consider only inputs

x = (p, p, p, · · ·), where p is present

The only output is

y = (1, 0, 0, 1, 0, 0, 1, 0, 0, · · ·).

Chapter 13: Q1:

For each of the following questions, give a short answer

1. True or False: If GFp holds for a state machine A, then so does FGp.

False

2. True or False: G(Gp) holds for a trace if and only if Gp holds.

True

Ans: The output is present at times t = 1, 2.5, 4, · · · .

(b) Sketch the output b if the input a is present only at times t = 0, 1, 2, 3, · · · .

Ans: The output is present at times t = 1, 3, 5, 7, · · · .

(c) Assuming that the input a can be any discrete signal at all, find a lower bound on the amount of time between events b. What input signal a (if any) achieves this lower bound?

Ans: The lower bound is 1. There is no input a that achieves this bound, but an input that comes arbitrarily close is where a is present at times t = 1 + ε, 2 + 2ε, 3 + 3ε, · · · , for any ε > 0.

Chapter 8, Q3:
Assuming fixed-point numbers with format 1.15 as described in the boxes on pages 234 and 235, show that the only two numbers that cause overflow when multiplied are −1 and −1. That is, if either number is anything other than −1 in the 1.15 format, then extracting the 16 shaded bits in the boxes does not result in overflow.

Overflow will occur if two bin digits to the left of the bin point are different. If the digits are 01, then the product is positive. The representation is

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

We can obtain result 1 only by multiplying −1 by −1.
Hence, the only way to get overflow is to multiply −1 by −1.

Chapter 8, Q4:

Consider a dashboard display that displays “normal” when brakes in the car operate normally and “emergency” when there is a failure. The intended behavior is that once “emergency” has been displayed, “normal” will not again be displayed. That is, “emergency” remains on the display until the system is reset.
In the following code, assume that the variable display defines what is displayed. Whatever its value, that is what appears on the dashboard.

```
volatile static uint8_t alerted;
volatile static char* display;
void ISRA() {
if (alerted == 0) {
display = "normal";
}
}
void ISRB() {
display = "emergency";
alerted = 1;
}
void main() {
alerted = 0;
...set up interrupts...
...enable interrupts...
...
}
```

Assume that ISRA is an interrupt service routine that is invoked when the brakes are applied by the driver. Assume that ISRB is invoked if a sensor indicates that the brakes are being applied at the same time that the accelerator pedal is depressed. Assume that neither ISR can interrupt itself, but that ISRB has higher priority than ISRA, and hence ISRB can interrupt ISRA, but ISRA cannot interrupt ISRB.

Assume further (unrealistically) that each line of code is atomic.

(a) Does this program always exhibit the intended behavior? Explain. In the remaining parts of this problem, you will construct various models that will either demonstrate that the behavior is correct or will illustrate how it can be incorrect.

Ans: No. Emergency will be overwritten by normal

Chapter 11, Q2
Suppose that two int global variables a and b are shared among several threads. Suppose that lock a
and lock b are two mutex locks that guard access to a and b. Suppose you cannot assume that reads and
writes of int global variables are atomic. Consider the following code:

```
int a, b;
pthread_mutex_t lock_a
= PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t lock_b
= PTHREAD_MUTEX_INITIALIZER;
void procedure1(int arg) {
pthread_mutex_lock(&lock_a);
if (a == arg) {
procedure2(arg);
}
pthread_mutex_unlock(&lock_a);
}
void procedure2(int arg) {
pthread_mutex_lock(&lock_b);
b = arg;
pthread_mutex_unlock(&lock_b);
}
```

Suppose that to ensure that deadlocks do not occur, the development team has agreed that lock b should always be acquired before lock a by any thread that acquires both locks. Note that the code listed above is not the only code in the program. Moreover, for performance reasons, the team insists that no lock be acquired unnecessarily. Consequently, it would not be acceptable to modify procedure1 as follows:

```
void procedure1(int arg) {
pthread_mutex_lock(&lock_b);
pthread_mutex_lock(&lock_a);
if (a == arg) {
procedure2(arg);
}
pthread_mutex_unlock(&lock_a);
pthread_mutex_unlock(&lock_b);
}
```

A thread calling procedure1 will acquire lock b unnecessarily when a is not equal to arg. Give a design for procedure1 that minimizes unnecessary acquisitions of lock b. Does your solution eliminate unnecessary acquisitions of lock b? Is there any solution that does this?

Ans:

```
void procedure1(int arg) {
int result;
pthread_mutex_lock(&lock_a);
result = (a == arg);
pthread_mutex_unlock(&lock_a);
if (result) {
pthread_mutex_lock(&lock_b);
pthread_mutex_lock(&lock_a);
if (a == arg) {

}
}
procedure2(arg);
}
pthread_mutex_unlock(&lock_a);
pthread_mutex_unlock(&lock_b);
```