**Kettering University**

**Microcomputers I**

Lab Exercise 6

**I/O Programming**

**State Machines in Software**

**Switch Debouncers**

Stopwatch

Spring 2022

**Prelab (10%):** Go over this handout rigorously, do Assignments **1 through 5**, and then upload one handout (prelab) per group to Blackboard in **.pdf** by **11:59 pm** on the **Tuesday** before your lab day.

**Lab report:** Upload one lab handout (report) per group to Blackboard (in **.pdf**) by **11:59 pm** on the **Sunday** following the lab day and **after** you have done all the assignments, answered all the questions, and shown your lab work to the lab instructor **individually**. A demo sign-up sheet will be posted if necessary.

In the lab report, please correct your prelab incorrect answers, if any.

If you manually scan your prelabs or lab reports for submission purposes, you may scan only the relevant pages of the handout, the pages that should be graded.

Names: Dylan Lozon, Kayla Jones

**Assignments**

**Note:** The members of each group should work on the prelab *independently* and then compare their results.

Prelab starts here:

1. **Reading Assignment**

**Switch contact bounce problem and switch debouncers**

When you push and release a button like the one shown in Figure 1, you are expecting it to generate ONE clean pulse as depicted in this figure:

(a)

(b)

Vout

Vcc

Push

GND

Vcc

Vout

1. **(a) Ideal pushbutton with no bounce (b) ideal output waveform**

A real pushbutton does not, however, behave like that: once the wiper hits the contact (after the button is pushed), the wiper bounces (i.e., makes and breaks contact) a few times before settling (*contact-bounce* problem). This may also happen when the button is released resulting in *unwanted* and *problematic* pulses as illustrated in Figure 2. Therefore, when you hit the pushbutton just one time, it looks like you have pressed and released it a few times!

(a)

(b)

Vcc

Vout

Push

GND

Vcc

Vout

1. **(a) Real pushbutton (b) real output waveform**

There are different ways to handle the contact-bounce problem. One way to solve the issue is to use an R-S latch as shown in Figure 3. This circuit is similarly able to hide the possible bounces that could occur when the button is released.

The pushbuttons on the Dragon 12+ board are bouncy, and therefore in today’s lab, you need a switch contact debouncer, a software-based one. In this technique and in general, once Vout changes, you wait for, say, 2 msec, and read Vout again; if the value is still the same, you will interpret the voltage change as a push or release; otherwise, just ignore it. Note that bounce duration, hence the above 2 ms, depends on the type of pushbutton.

Let us call this technique *wait and see*. In the above example, however, you do not have to read Vout again; just wait. It is sufficient to debounce the switch! We call it *wait* *and go*.

S

Vout

Push

Vcc

Q

R

R2

R1

GND

GND

Q’

1. **RS-latch based switch contact debouncer**
2. In the space provided for Figure 1, draw a transition graph to control a counter: when PB1 is pushed and released the counter is count enabled. (Do not worry about the clock signal of the counter. Let us assume that a 1 Hz clock is provided.) When PB1 is pushed again the counter is disabled; when PB1 is released the counter should remain disabled, until PB1 is pushed and released again. Let us assume that PB1 is bounceless. The counter should be reset when PB0 is pressed provided that the counter has stopped counting. The pushbuttons are normally high. Show the active-high signal CE, Count Enable, and the active-high signal R, Reset, in your graph.

CE

1

0

0/1

1/1

R

Count

0

PB1 = 1

Stop

PB0 = 0

Stop

Stop

1

0

0

1

1

0

1. **Incomplete transition graph to be completed**
2. Using the above graph, complete the partially drawn flowchart shown in Figure 2 to describe an 8-bit Stopwatch: the counter in Assignment 2 is the 8-bit Timer of the Stopwatch. The content of this counter will be displayed on the 8 individual LEDs of the Dragon 12+ board. PB1 (SW4) is to start/stop the Stopwatch as explained above. The timer should increment at around 1 Hz while counting. When it stops, the counter will reset to 0 if PB0 is pressed. Note that the pushbuttons are bouncy.

Case current state

01

11

10

State: Memory

00

?

Initialization

PB1

1

Current state ← %00

1

PB1

Current state ← %11

PB1

1

PB0

1

Read PORT H

Read PORT H

Read PORT H

Read PORT H

PB0: SW5

PB1: SW4

PB: Pushbutton

Reset counter

Current state ← %10

1. **Incomplete flowchart to be completed**
2. Briefly, clearly, and legibly explain which PB has to be debounced?

PB1 needs to be debounced because the counter will be made inaccurate if it is not debounced.

1. How do you write a Switch-like construct in assembly language for the following example? Let us assume that state is a memory location:

Switch (state)

{

case 0 :

…

break;

case 1 :

…

break;

case 2 :

…

break;

case 3 :

…

}

; switch statement starts here

ldaa state

cmpa #0

beq case0

cmpa #1

beq case1

cmpa #2

beq case2

bra default ; if you have only 4 possible cases, you could name

; the 4th case explicitly and put error handling here

; instead of using default

case1:

; do thing ; case1 code goes here

bra endCase ; bra endCase is essentially our break statement

case2:

; do thing

bra endCase

case3:

; do thing

bra endCase

default:

; do thing

; we don’t need a break here because it’s the last one

endCase:

Prelab ends here.