EE109: Introduction to Embedded Systems
Fall 2023 – Quiz 1
09/26/23, 7PM – 8:15PM

[Complete all the information in the box below.]

Name:___Solutions______________

Student ID: ____

Email: ____________________@usc.edu

Lecture section (Circle One):

Weber	Redekopp	Puvvada	Weber
9:30 a.m.	11 a.m.	12:30 p.m.	2 p.m.

Ques.	**Your score**	**Max score**	**Recommended Time**
1		8	8 min.
2		10	10 min.
3		12	15 min.
4		6	7 min.
5		14	35 min.
Total		50	75 min.

Calculators are ONLY allowed on Question 3 – Analog/Resistive Circuits.
Using them on any other problem is an academic integrity violation.

Only work on this exam will be graded (no work on scratch paper will be considered).

Do NOT write in the upper corner with the QR / Page number code.

1. **(8 pts.) Number Systems**
   1.1. Convert **11001101** binary to unsigned **decimal**: _128+64+8+4+1 = 205 dec.__

   1.2. Convert **91 decimal** to **(unsigned) binary** (use exactly 8 bits):

   91 = 64 + 16 + 8 + 2 + 1

   0b__0101 1011________________

   1.3. Convert **1101101.101** unsigned **binary** to **hexadecimal**:

   _____6 D . A_______________________ hexadecimal

A market stocks **36** different kinds of vegetables and **14** kinds of fruits.

   1.4. If we wanted to assign a unique binary number to JUST the 36 vegetables how many bits would this require? ____6 (2^5 = 32 while 2^6=64)_____

The market wants to assign fixed-size, unique binary numbers to each type of vegetable and fruit (i.e. fixed size means codes for fruits and vegetables should be the same number of bits). However, it wants to differentiate vegetables and fruits quickly by using the **MOST-SIGNIFICANT bit** to identify **vegetables** (i.e. **MSB=0**) from **fruits** (i.e. **MSB = 1**).

   1.5. What is the minimum number of bits required to represent vegetables and fruits to meet this new system? ___7______

2. (**10 pts.**) **Bit Manipulations**. Complete the following **single-line** statements to perform the desired operation stated in the line(s) above the blanks. Assume a standard bit numbering (**bit 7** is the MSB, **bit 0** is the LSB = Least Significant Bit). You may NOT change the structure or code given.

   2.1. Turn on (set to 1), bit **6** and **3** of PORTB without affecting other bits of PORTB.

```
PORTB _|= (1<<6) | (1<<3)_; // mask can be: (9<<3) or 0x48 or 0b01001000
```

   2.2. Turn off (clear to 0), bit **1** of DDRC without affecting other bits of DDRC.

```
DDRC _&= ~(1 << 1)_; // mask can be:  0xfd or 0b11111101 or ~(0x02)
```

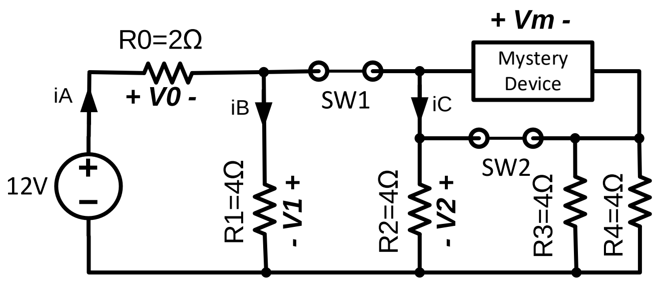
   2.3. Assume bit **4, 3, and 2** of Group D are already configured to be inputs. Complete the if statement to be true ONLY if **bit 4 is 1 (high voltage) and bit 3 and 2 are 0 (low voltage)**.

```
if(_(PIND & 0x1c) == 0x10_) {…};
// 0x1c can be replaced with (1<<4)|(1<<3)|(1<<2) or 0b00011100 or (7 << 2)
// Comparison constant can be (1<<4) or 0b00010000
```

3. **(12 pts.) Resistive Circuits.** Examine the circuit below and to the right and then answer the questions. Notice a.) all resistors labelled Rx are 4 ohms and b.) the mystery device which could be a resistor, wire, or open-circuit.

Show work for potential partial credit. A calculator may be used for this problem only.

For 3.1-3.4, determine and write T (for true) or **F** (for false) assuming **SW1 and SW2 are closed**.



3.1. _T___ **T / F**: If the mystery device is an **open-circuit**, then iA = iB + iC.

3.2. _ T___ **T / F**: If the mystery device is a **resistor** with **non-zero** resistance, then **Vm** will be **0**.

3.3. _T__ **T / F**: If the mystery device is a **wire**, V2 = 12V – V0

3.4. _T__ **T / F**: Regardless of what the mystery device is, **R1, R2, R3, and R4** are in **parallel**.

For 3.5-3.7, assume the mystery device is an OPEN CIRCUIT.

3.5. (2 pts.) With **SW1 and SW2 open**, solve for the voltage **V1** (round to 2 decimal places, if needed). R0 and R1 are in series when both SW open. Thus we can use voltage divider.

V1 = ____**8**______ V = 12V * (4 / (2+4)) = 12 * 2/3 = 8V

3.6. (2 pts.) Now with **SW1 closed and SW2 open**, solve for the voltage **V1** (round to 2 decimal places, if needed).

Now R2 and R1 are in parallel (equal to 2 ohms), and that combination is in series with R0. Also V1 = V2.

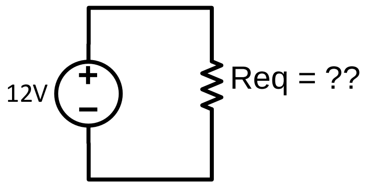
V1 = ____**6**_____ V = 12V * (2 / (2 + 2)) = 12V / 2 = 6V

3.7. (2 pts.) Now with **SW1 closed and SW2 closed**, solve for the voltage **V1** (round to 2 decimal places, if needed).

Now R1-R4 are in parallel (equal to 1 ohms), and that combination is in series with R0.

V1 = ___**4**______ V = 12V * (1 / (2 + 1)) = 12V / 3 = 4V

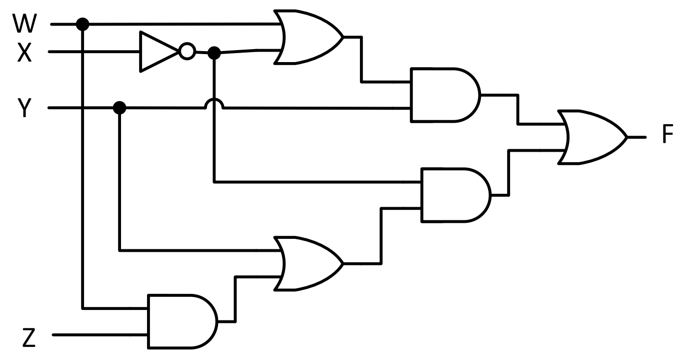
3.8. (2 pt.) **If SW1 closed, SW2 is back to open, and assuming the mystery devices is a 6 ohm resistor,** what is the equivalent resistance of all resistors (round to **1 decimal place** if necessary).



R1 is in parallel with R2 which is parallel with the series of Mystery device and two parallel R3 and R4: (4 || 4 || (6 + (4 || 4))) = 4 || 4 || 8 = 1 / (1/4 + ¼ + 1/8) = 8/5 = 1.6. We then add 2 since R0 is in series with all the others.

Req =____**3.6**________ ohms

4. (6 pts.) **Logic Circuits**: Consider the circuit shown below and reproduced at the bottom for markup.



Questions:

4.1) Given **{W,X,Y,Z} = 1,0,1,0** respectively, what value will **F** output? __**1**____

4.2) How many levels of logic is this circuit? ___**4**___

4.3) It is possible to make the output, F, produce a `**1**` by assigning **only 2 of the 4 inputs** to specific values (i.e. the other two bit values don't matter). Two such pairs exist; find one and list them below.

First bit name: _**X**_ First bit value: _**0**_ Second bit name: _**Y**_ Second bit value: _1_

Or **W,1** **Y,1**

4.4) It is possible to make the output, F, produce a `**0**` by assigning **only 2 of the 4 inputs** to specific values (i.e. the other two bit values don't matter). Two such pairs exist; find one but you may NOT use the exact same two bits as you used in 4.3 above. List your answer below.

First bit name: _**X**_ First bit value: _**1**_ Second bit name: _**Y**_ Second bit value: _**0**_

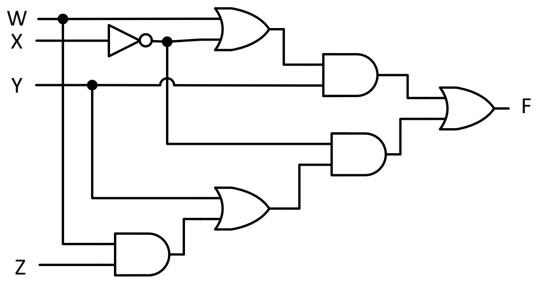
Or **Z,0** **Y,0**

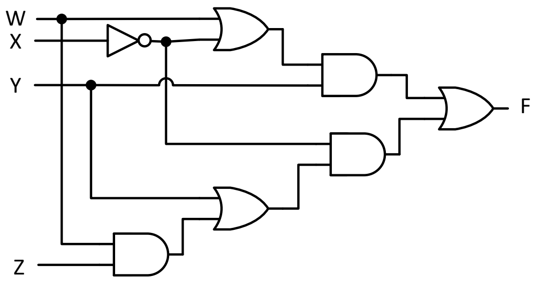
Or X,**1** **W,0**

4.5) By assigning only 2 of the 3 inputs W, Y, and/or Z to specific values, we can make the output, F, ALWAYS equal/match `**not X**`, (i.e. when X=0, F will be 1 and when X=1, F will be 0. Find the two inputs (from W, Y, and Z) and their values. List your answer below.

First bit name: _**W**_ First bit value: _**0**_ Second bit name: _**Y**_ Second bit value: _**1**_

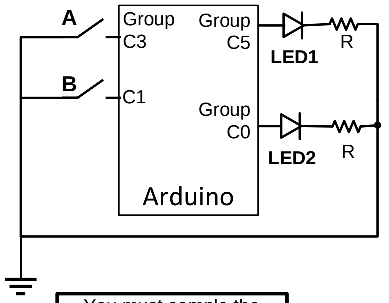
Repeated drawings for your own scratch work and annotation (will not be graded):



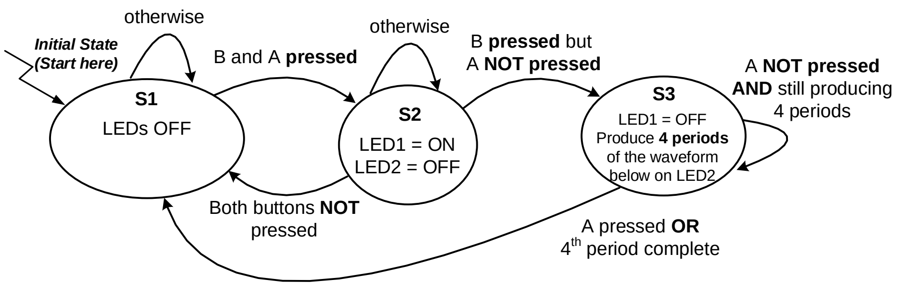


5. (14 pts.) **READ THE ENTIRE PAGE before solving.** Using the Arduino circuit below (exactly as drawn) with **two buttons (A and B) on group C, bit 3 and 1,** respectively**,** and **two LEDs (LED1 and LED2) on** g**roup C, bit 5 and 0, respectively,** **<u>complete</u>** the Arduino C-code program on the following page that implements the behavior described below.

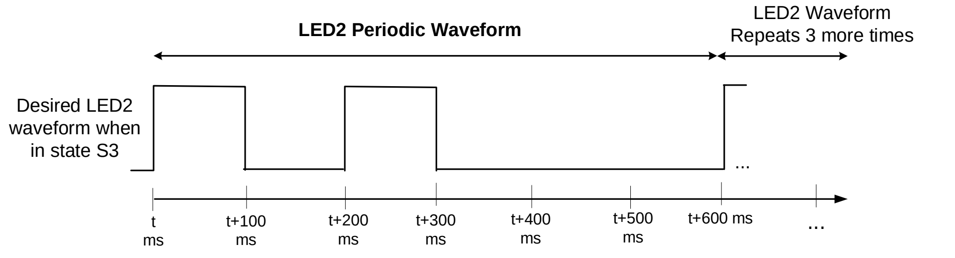
Implement the software state machine specified in the state diagram below. In the S3 state, LED1 is OFF and **4 periods** of the waveform pattern should be generated on **LED2**. If A is pressed during generation of the 4 periods, immediately stop and transition to S1. Otherwise, transition to S1 immediately after the 4th period completes **fully**. Review the state diagram for the specification of how the S1 and S2 should work and what actions should be taken in those state. You must sample the input buttons every **100ms** and make the appropriate state transition. Most of the code is given on the next page. Your task is to fill in the missing blanks on the next page with the correct code.



You must sample the buttons every 100 ms



Initial State (Start here)
otherwise
B and A **pressed**
otherwise
B **pressed** but A **NOT pressed**
A **NOT pressed AND** still producing 4 periods
S1
LEDs OFF
S2
LED1 = ON
LED2 = OFF
S3
LED1 = OFF
Produce **4 periods** of the waveform below on LED2
Both buttons **NOT** pressed
A pressed **OR** 4th period complete



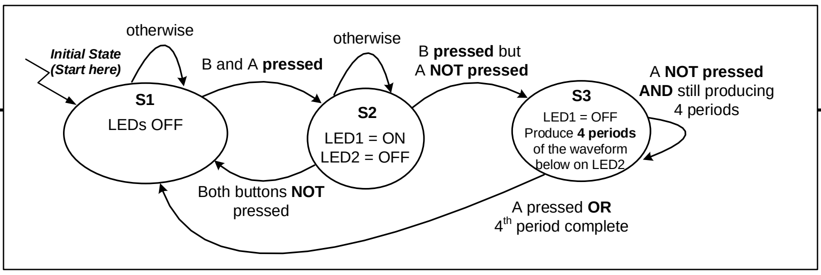
Important Requirements

- The code separates Next State Transition Logic and Output Logic into separate if statements. You must use this approach and cannot alter the code structure.
- The **bit positions for buttons: A and B and outputs: LED1 and LED2 are defined at the start of the program**. These can be used wherever needed in the program.
- The program must **check/sample the button inputs every <u>100 ms</u>** and update the state appropriately.
- You may **NOT add other DELAY** (e.g. _delay_ms()) statements than the one provided.
- You may **NOT alter the code given**, but may only fill in the blanks shown.
- You need not worry about debouncing the button presses.

Complete your code on the page below!

```
#include <avr/io.h>
#include <util/delay.h>
const int A = 3, B = 1;
const int LED1 = 5, LED2 = 0;
enum {S1, S2, S3};
unsigned char btnmask = _0x0a or (1<<A)|(1<<B) or (5 << 1) or 0b00001010__________;
                        // ^^^^^^^^^^^^^^^^ mask for the two bit locations of A and B
int main() {
    char state = S1, cnt = 0; // state variable plus count for waveform
    __DDRC______  _|_= (1 << LED1)|(1 << LED2);  // fill in reg. name and operator
    // pullups for buttons
    PORTC _|= btnmask (or equivalent mask from above)_____________;

    while(1) {  // this is the only loop allowed
        _delay_ms(100);  /* this is the ONLY delay statement allowed */
        char inp = __PINC_________;  // sample inputs by filling in one register name
        // next state logic
        if( state == S1 ) {
            if((inp & ____btnmask (or equiv)____) == __0____ ) {  state = S2; }
        }
        else if(state == S2) {
           if((inp & btnmask) == _(1<<A) or 0x08 or 0b00001000______) {
               state = S3;
               _cnt = 0________________;
           }
           else if((inp & btnmask) == __btnmask (or equiv)______) {
               state = S1;
           }
        }
        else {
            cnt++;
            if((__cnt___ == ___24___) || ((inp & (_(1<<A) or 0x08_)) _==___ 0 )) {
                state = S1;            // stop pattern and go to S1
            }
        }
        // Output logic (LED actions)
        if( state == S1 ) { PORTC _&= ~((1 << LED1)|(1 << LED2)); } // &= ~0x21  OR  &= 0xde
        else if( state == S2 ) { PORTC _|= (1 << LED1)_; }
        else {
            PORTC _&= ~(1 << LED1)__________;
            if(__cnt % 6 == 0 || cnt % 6 == 2____________)
                { PORTC |= (1 << LED2);    }  // turn on at t+0 and t+200
            else if(__cnt % 6 == 1 || cnt % 6 == 3______________) {
                PORTC _&= ~(1<< LED2)_______________);  // turn off at t+100 and t+300
        }    }
    } /* end while */
    return 0;
} /* end main */
```


Initial State (Start here)
otherwise
S1
LEDs OFF
B and A pressed
otherwise
S2
LED1 = ON
LED2 = OFF
Both buttons NOT pressed
B pressed but A NOT pressed
S3
LED1 = OFF
Produce 4 periods of the waveform below on LED2
A NOT pressed AND still producing 4 periods
A pressed OR 4th period complete