Name: SOLUTION **RIN:**

Q.1 (1 pnts): What color is the car that we use in the lab? Red (with a white protoboard on top)

Q.2 (2 pnts): True or [✓ **False**]: An `int16_t` can hold more unique values than a `uint16_t` as it can go negative.

They both can only hold 2^16 unique bit patterns

Q.3 (5 pnts): Two variables, `uint8_t hi, lo;`, are used to set the most significant and least significant bytes (MSB and LSB), respectively, of `uint16_t combo = 0;`. **Cross out** any commands below that **do not** successfully perform this operation.

~~`combo = hi + lo;`~~

`combo = hi * 256 + lo;`

`combo = hi<<8 + lo;`

`combo = hi<<8 | lo;`

~~`combo = hi | lo>>8;`~~ Must shift the MSByte up

~~`{combo |= lo; combo &= hi<<8;}`~~ Would work with |= instead of &=

Q.4 (9 pnts): For each expression below, determine the final value in Hexadecimal.

0xAB=1010 1011 0xBA=1011 1010

`0xAB | 0xBA   =` 0xBB 1011 1011

`0xAB || 0xBA  =` 0x01

`0xAB | ~0xBA  =` 0xEF 1110 1111

`0xAB | !0xBA  =` 0xAB

`0xAB || !0xBA =` 0x01

`~0xAB | 0xBA  =` 0xFE 1111 1110

Q.5 (2 pnts): [✓ **True**] or **False**: The following two statements are equivalent in their result.

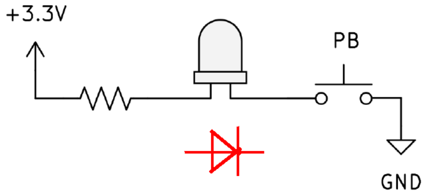
`P2DIR = 0xD8;` `P2DIR = 0x1B<<3;` 0001 1011<<3=1101 1000=0xD8

Q.6 (4 pnts): What will be printed on the terminal after this code segment runs? Reproduce the output *exactly*.

```
uint8_t a = 127;
int8_t b = -1;
printf("a = %u, b = %d, a = %x, b+1 = %u \n\r",a,b,a,b+1);
```

a = 127, b = -1, a = 7f, b+1 = 0

Q.7 (3 pnts): Consider the circuit to the right and assuming a properly sized resistor. What must happen to ensure the LED will light when the pushbutton is pressed? You may answer with a drawing or words.



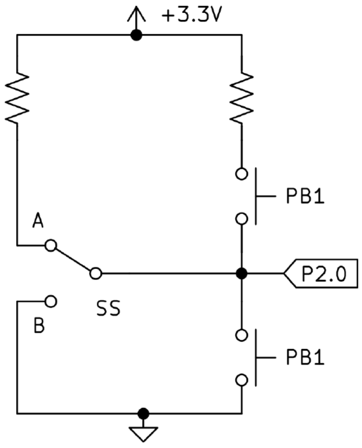
The LED must be oriented so that the diode passes current to ground

Q.8 (2 pnts): If the LED above is replaced with a BiColor LED, how would your answer to the above question change?

The orientation of the BiLED is not critical, it just changes the color: red to green

Q.9 (4 pnts): For the circuit below, find all combinations of the switch settings such that the logic sensed by P2.0 is TRUE. Denote for each combination the state of all switches (SS: position A or B, Pushbuttons: PRESSED or UNPRESSED).

SS	BP1	BP2
A	UNPRESSED	UNPRESSED
A	PRESSED	UNPRESSED



Q.10 (4 pnts): What is the final value for variable `i` after the below code segment runs?

```
uint8_t i = 0;
uint8_t j = 55;
while(i <= j){
    i += 5;
    j--;
}
```

i = 50

Name: **RIN:**

Q.11 (6 pnts): Given the code segment below, indicate which pins are are known to be **inputs** and **outputs**.

```
P1DIR &= 0x30;
P1DIR |= ~0x03;
P1OUT = 0x0FA2;
```

XXXX XXXX

&=0011 0000 -> 00XX 0000

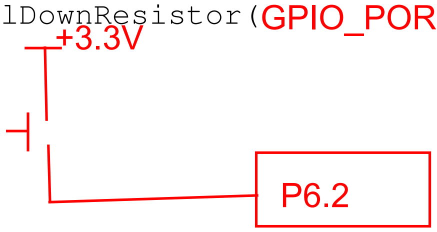
|= 1111 1100 -> 1111 1100

INPUTS: P1.1, P1.0

OUTPUTS: P1.7, P1.6, P1.5, P1.4, P1.3, P1.2

Q.12 (6 pnts): Fill in the missing arguments for configuring a pushbutton input connected to P6.2. Additionally, draw an appropriate pushbutton circuit that would work with the initialization code.

```
GPIO_setAsInputPinWithPullDownResistor(GPIO_PORT_P6, GPIO_PIN2 );
```



Q.13 (6 pnts): Add a comment to each line to interpret what each line is doing. Is the code below a **Blocking** or [✓ **Non-Blocking**] implementation for checking if a specific signal occurred on a GPIO pin? Assume that the xxx and yyy are the appropriate values the pins desired.

The GPIO signal checking is Non-Blocking, the delay for ovf_cntr to be 50 IS Blocking.

```
while(1){
    lp_cnt++;                          Increment the loop count
                                       Clear the interrupt overflow counter
    ovf_cntr = 0;

    while( ovf_cntr < 50 );            Blocking wait for 50 counter overflows

    GPIO_setOutputLowOnPin(yyy,yyy));
                                       Set specified output pin low
    if(!GPIO_getInputPinValue(xxx,xxx)){   Get specified input pin value, if pin is low then
                                       Set specified output pin high
        GPIO_setOutputHighOnPin(yyy,yyy));

        lp_cnt = 0;                    Reset the loop count back to 0 (only when a press happens)
    }
    printf("Loops since press: %u\r\n",lp_cnt);
}                                      Print # of loops since last button press
                                       (a loop is checking for a press
                                       after 50 interrupt overflows)
```

For all questions on this page: consider a generic UP counting timer with a desired 40 Hz reset frequency. The input clock divider for the timer is set to 8 and the timer count period is set to 12500.

Q.14 (6 pnts): What must the base clock frequency be (the clock source used as the input) for the described timer **and** how often does the timer increment, in seconds?

(clock divider/SMCLK) * 12500 = 1/40

SMCLK = 40*clock divider*12500 = 40*8*12500 = 4 MHz

T_TCLK = 8/4MHz = 2 us

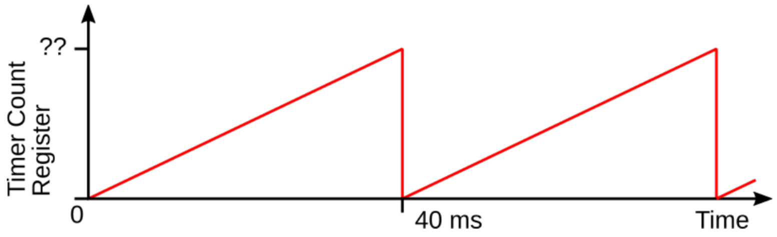
Q.15 (4 pnts): It is instead desired to have the reset frequency be 20 Hz. What two changes to the given configuration could be made to affect this change? Note that these two changes produce the change independently; only one of the changes would be necessary.

Twice the period: 25000
Larger clock divider by 2x = 16
Slower SMCLK by half = 2 MHz <-- Not looking for this answer, but we'll accept it.

Q.16 (4 pnts): Assuming the original reset period of 40 Hz, an interrupt function for the timer is written such that the global variable `track` increments each reset. Provide line(s) of code that would reliably produce a program delay of 10 s using this support.

```
track = 0;
while(track < 400);          //10s/(1/40) = 400
```

Q.17 (4 pnts): What numeric value for the timer is denoted by the "??" in the figure below? Give that value.



Note that the 40 ms was corrected to 25 ms during the exam.

This is just the number of timer counts: 12500

Name: **RIN:**

You may use shorthand code for the remaining problems: As long as it is clear what function/defined value you are referring to, you may shorten the name of the DriverLib functions/defined values to save time and/or space.

A simple program is desired to control one BiColor LED (**BLED: P2.0,P2.1**) and one bidirectional motor (**MOTOR ENABLE: P1.0, MOTOR DIRECTION: P1.1**), via a slideswitch (**SS: P3.0**) and one pushbutton (**PB: P3.1**). When the state of any of the inputs change, the program should hold the new state for a minimum of 1 second, measured using `Timer_A1` with a period of **25 ms**.

All input pins have external **pull-up** resistors and the motor turns on when the enable is driven **high**.

Q.18 (10 pnts): Complete the timer initialization function to initialize both the timer and timer interrupt function (bottom of page).

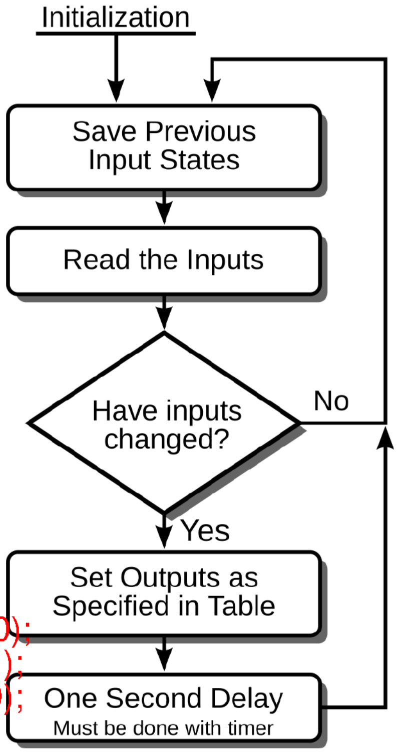
Any valid set of divider and timer period is acceptable here.

```
void TimerInit(){

    Timer_A_UpModeConfig timer_cfg;
    timer_cfg.clockSource = TIMER_A_CLOCKSOURCE_SMCLK; // 24 MHz

    timer_cfg.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_24;  //or 48

    timer_cfg.timerPeriod = 24999;                                  //and 49999

    timer_cfg.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_ENABLE;

    timer_cfg.timerClear = TIMER_A_DO_CLEAR;     //optional

    Timer_A_configureUpMode(TIMER_A1_BASE, &timer_cfg);

    Timer_A_enableInterrupt(TIMER_A1_BASE);//optional

    Timer_A_registerInterrupt(TIMER_A1_BASE,
        TIMER_A_CCRX_AND_OVERFLOW_INTERRUPT,Timer_Int );

}

void Timer_Int(){     (we meant to ask for this, but didn't. No points for this)
    Timer_A_clearInterruptFlag( TIMER_A1_BASE );
    timer_counter++;
}
```

Q.19 (18 pnts): Convert the main program loop shown in the flow chart into valid C code. For the "Read the Inputs" block, you may assume that a function exists, `getInputs()`, which reads the inputs and saves the raw output of the reads to the global variables `uint8_t SS, PB`. **Your code must explicitly follow the flow chart.** Use the back of this page if more space is needed.

SS[1](P3.0)	PB[1](P3.1)	BLED[2](P2.0,P2.1)	Motor[3](P1.0,P1.1)
OFF	OFF/ON	RED	OFF
ON	OFF	GREEN	ON, Forward
ON	ON	GREEN	ON, Reverse

1: For Pushbutton: ON == Pressed, Slideswitch: ON == HIGH.
2: BLED RED/GREEN pin states are up to you.
3: If motor direction is 1, Motor will spin forward.



```
// Defined global variables
uint8_t SS, PB, timer_counter;
// Initialiations are here. You do not need to call them
while(1){
    uint8_t SSprev = SS;
    uint8_t PBprev = PB;
    getInputs();
    if((SS!=SSprev)||(PB!=PBprev){
        if(SS==0){
            // BiLED RED and Motor Off
            GPIO_setOutputHighOnPin(GPIO_PORT_P2,GPIO_PIN0);
            GPIO_setOutputLowOnPin(GPIO_PORT_P2,GPIO_PIN1);
            GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);
        }else{
            // BiLED GREEN and Motor On
            GPIO_setOutputLowOnPin(GPIO_PORT_P2,GPIO_PIN0);
            GPIO_setOutputHighOnPin(GPIO_PORT_P2,GPIO_PIN1);
            GPIO_setOutputHighOnPin(GPIO_PORT_P1,GPIO_PIN0);
            if(PB){ // PB is not pressed (pressed/unpressed not important)
                // Motor forward
                GPIO_setOutputHighOnPin(GPIO_PORT_P1,GPIO_PIN1);
            }else{
                // Motor reverse
                GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN1);
            }
        }
        // One second delay
        timer_counter = 0;
        while(timer_counter<40);
    }

}
```

C-Coding int variables: [u]int#_t, #=Number of bits.
Uint8_t [0,255] **int8_t** [-128,127] **uint16_t** [0,65535] **int16_t** [-32768,32767]
uint32_t [0,4294967295] **int32_t** [-2147483648,2147483647]
float: 32-bit $\pm[1.4*10^{-45},3.4*10^{38}]$ Digit Accuracy:6
double: 64-bit $\pm[4.9\text{x}10^{-324},1.8\text{x}10^{308}]$ Digit Accuracy:15
Bitwise Operations: & | ^ ~
Logical Operations: ! && || == != > < >= <= (Result: true 1 / false 0)
Math Operators: + - * / % **bool**:in C the bool keyword only valid in lowercase
Value Operators: ! << >> ++ --
Functions: <return type> function_name(<arg_type> in1,<arg_type> in2,...)
void no arguments or return type **DON'T FORGET TO DECLARE!**
printf("format string",var1,var2,...)
printf formats: **%u** decimal unsigned integer, **%d** decimal signed integer,
%x or %X Hexadecimal integer (no 0x added), **%c** character
%lu Unsigned Decimal Number **%ld** Signed Decimal Number, **%f** float
\n Move down line, **\r** Move to beginning of line, **\t** tab, **\b** backspace
IO functions:
void **putchar**(uint8_t val) uint8_t **getchar**() uint8_t **getchar_nw**()
val to terminal get keypress (blocking) get keypress (non-blocking)
Arrays:**<**type**>** arrayname[maxsize] ={};

Bit Masking: **&**-set bits low, **|**-set bits high
^ (Exclusive OR) toggles the value of a bit
Set low:PxOUT &= ~0x26;**Set High**:PxOUT |= 0x49;
Toggle:PxOUT ^= 0x01 (eg:0101 ^= 1111 => 1010)

```
  X X X X X X X X
& 1 1 0 1 1 0 0 1   ~0x26
= X X 0 X X 0 0 X
```

```
  X X X X X X X X
| 0 1 0 0 1 0 0 1   0x49
= X 1 X X 1 X X 1
```

Base convert:
0001=0x1 0110=0x6 1011=0xB
0010=0x2 0111=0x7 1100=0xC
0011=0x3 1000=0x8 1101=0xD
0100=0x4 1001=0x9 1110=0xE
0101=0x5 1010=0xA 1111=0xF

false=0 true=any other

GPIO Registers x=1..11 (port#)
PxDIR: 0-Input,1-Output
PxOUT: Set state of outputs
PxIN: Read value of pins

Usually requires two bitmasking cmds.: &=,|=

Layout: Bit/Pin order: 76543210
Do not modify other bits if not necessary

GPIO DriverLib
uint8_t **GPIO_getInputPinValue**(uint8_t port,uint8_t pins)
Return GPIO_INPUT_PIN_LOW/GPIO_INPUT_PIN_HIGH
void **GPIO_setOutputLowOnPin**(uint8_t port,uint8_t pins)
void **GPIO_setOutputHighOnPin**(uint8_t port,uint8_t pins)
void **GPIO_toggleOutputOnPin**(uint8_t port,uint8_t pins)
void **GPIO_setAsOutputPin**(uint8_t port,uint8_t pins)
void **GPIO_setAsInputPin**(uint8_t port,uint8_t pins)
void **GPIO_setAsInputPinWithPullUpResistor /**
void **GPIO_setAsInputPinWithPullDownResistor**
(uint8_t port,uint8_t pins)

Possible ports:
GPIO_PORT_Px
Possible pins:
GPIO_PINy

x=1..11, y=0..7

Multiple pins announcement:
GPIO_PIN0|GPIO_PIN1

GPIO Interrupt
void **GPIO_enableInterrupt**/**GPIO_disableInterrupt**(uint8_t port,uint8_t pins)
void **GPIO_interruptEdgeSelect**(uint8_t port,uint8_t pins,uint8_t edgeSelect)
edgeSelect=GPIO_LOW_TO_HIGH_TRANSITION or GPIO_HIGH_TO_LOW_TRANSITION
Register: void **GPIO_registerInterrupt**(uint8_t port,<function_name>)
Check: uint16_t **GPIO_getEnabledInterruptStatus**(uint8_t port)
returns bitwise OR of pins that triggered interrupt (eg. GPIO_PIN1|GPIO_PIN3)
Clear: void **GPIO_clearInterruptFlag**(uint8_t port,uint8_t pins)

Debouncing: __delay_cycles(#) to wait # of SMCLK cycles.
How to calc # for delay time:delay_time/(1/freq)   1 MHz = 1000000 Hz

Other Functions:
Absolute value: int32_t abs(int32_t number)
Round up: double ceil(double number), **Round down:** double floor(double number)
Random number:uint32_t rand(), Seed random number: void srand(uint32_t seed)

Timer:
Modes: Up Mode(0~SpecifiedValue~reset);**Up Down Mode**(0~SpecifiedValue~0);
Continuous Mode(0~0xFFFF (65535)~reset)
Timer_A DriverLib Configuration struct Timer_A_Up/UpDownModeConfig fields:
 .clockSource = TIMER_A_CLOCKSOURCE_x
       x=EXTERNAL,ACLK,SMCLK,INVERTED_EXTERNAL_TXCLK
 .clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_y
       y=1,2,3,4,5,6,7,8,10,12,14,16,20,24,28,32,40,48,56,64
 .timerPeriod = 0 to 65535   (Sets value of CCR0)
 .timerClear = TIMER_A_v_CLEAR,TIMER_A_v_CLEAR   v=DO,SKIP
 .timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_ENABLE or DISABLE
Initial:
void **Timer_A_configureUpMode**(uint32_t timer,Timer_A_UpModeConfig *config)
Start:void **Timer_A_startCounter**(uint32_t timer , uint16_t timerMode)

timerMode=TIMER_A_UP_MODE,TIMER_A_UPDOWN_MODE,TIMER_A_CONTINUOUS_MODE
Operation:
void **Timer_A_stopTimer**(uint32_t timer)
void **Timer_A_clearTimer**(uint32_t timer)
uint16_t **Timer_A_getCounterValue**(uint32_t timer)
Others:
Struct Timer_A_ContinuousModeConfig fields: without timerPeriod
Initial:**Timer_X_configureUp/Updown/ContinuousMode**(timer,&config)(X=A,B……)
Default timer:TIMER_A0_BASE

Timer Interrupt
Situations: TimerA(CCR0-CCR4 reset to 0)/timer counts to CCR0 value
Enable:void **Timer_A_enable/disableInterrupt**(uint32_t timer)
(By default if timer reset to 0 this interrupt triggered)
Enable CCR0 Interrupt: void **Timer_A_enable/disableCaptureCompareInterrupt**(uint32_t timer , uint16_t captureCompareRegister **)**   **(optional)**
Regist:void **Timer_A_registerInterrupt**(uint32_t timer , uint8_t interruptSelect = TIMER_A_CCR0_INTERRUPT/TIMER_A_CCRX_AND_OVERFLOW_INTERRUPT , <function name>)
<Function name>: Only the name no”()”**!**
Check:uint32_t **Timer_A_getEnabledInterruptStatus**(uint32_timer)(all interrupts)
    Returns either TIMER_A_INTERRUPT_PENDING or TIMER_A_INTERRUPT_NOT_PENDING

Clear: **Reset:** void **Timer_A_clearInterruptFlag**(uint32_t timer)
  CCR:void **Timer_A_clearCaptureCompareInterrupt**(uint32_t timer , uint16_t captureCompareRegister)

$$f_{TCLK} = \frac{f_{SMCLK}}{N_{div}} \quad N_{timer} = 1 + CCR0 \quad T_{timer} = N_{timer} T_{TCLK} \quad f_{timer} = \frac{f_{TCLK}}{N_{timer}}$$

$$T_{TCLK} = N_{div} T_{SMCLK} = N_{div} \frac{N_{div}}{f_{SMCLK}} \qquad N_{timer} = T_{timer} f_{TCLK}$$