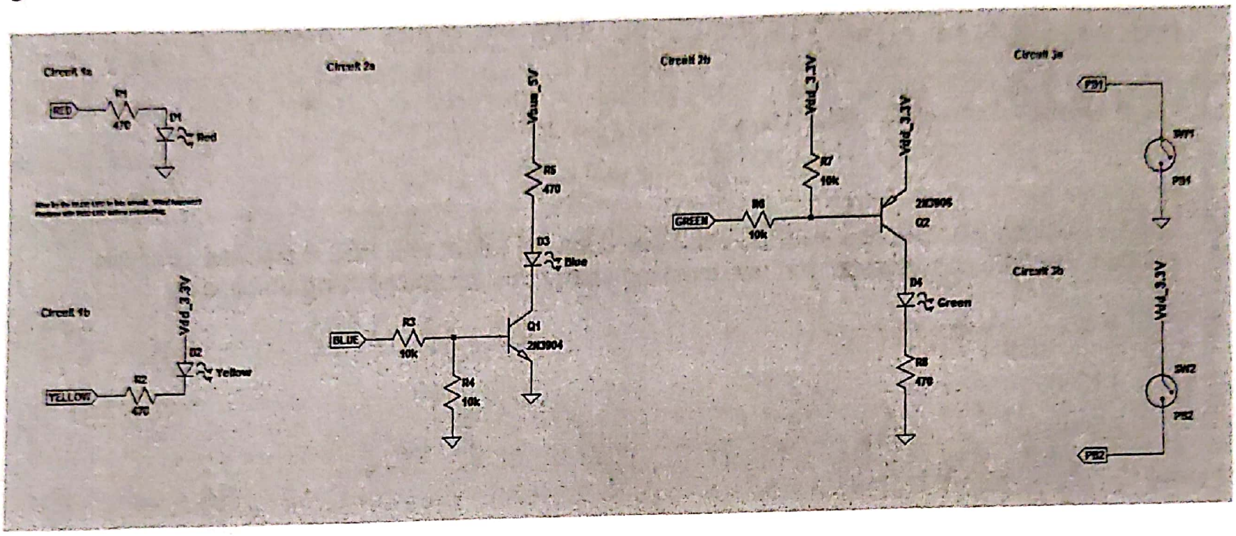
CSE3442 Embedded Controllers I **Name** KEY
Spring 2020, Test 1

Assume that a TM4C123GH6PM controller is used for the following problems. Assume the APB port is used for all GPIO. All work must be done on these test pages. Do not write test solutions on the ethics statement page. Calculators, datasheets, notes, and solved problems are allowed on the exam. The use of devices or software that support assembly or compilation of code is not allowed. Phones or communications devices may not be used during the exam.

1. In lab 2, suppose circuit 2a and 2b are driven by GPIO pins PA2 and PA3, respectively with a common ground to the controller. Also assume circuit 3b is connected to GPIO pin PB4.



a. Write the code to initialize all registers (system control, GPIO) necessary to allow control of the blue and green LEDs. Write the code so that any other GPIO pins are not affected.

```
SYSCTL_RCGCGPIO_R |= 1;
_delay_cycles(3);

GPIO_PORTA_DIR_R |= 4|8;
GPIO_PORTA_DR2R_R |= 4|8;
GPIO_PORTA_DEN_R |= 4|8;
```

5

b. Derive #define BLUE and #define GREEN statementx that allow bit-banded control over the blue and green LEDs.

```
# define BLUE (*((volatile uint32_t)(0x4200 0000 + (0x43FC)*32 +2*4
# define GREEN (*((volatile uint 32_t)(0x4200 0000 + (0x43FC)*32 + 3*4)))
```

4

c. Show the setting to turn on the blue LED and turn off the green LED:

BLUE = 1; GREEN = 1; 3

d. Write the code to initialize all registers (system control, GPIO) to configure the push buttons to be read correctly. Write the code so that any other GPIO pins are not affected.

```
SYSCTL_RCGCGPIO_R |= 2;
_delay_cycles(3),

GPIO_PORTB_DIR &= ~16;
GPIO_PORTB_DEN |= 16;
GPIO_PORTB_PDR_R |= 16;
```

5

e. Write a blocking function void waitPb2(void) that does not return until PB2 is pressed. For this solution, read the data register and use masking operations instead of using bit-banding.

```
void waitPb2()
{
    while (!(GPIO_PORTB_DATA_R & 16));
}
```

3

f. Repeat (e) using bit-banding.

```
# define PUSH_BUTTON (*((volatile unit32_t)(0x42000000 + (0x53FC)*32 + 4*4

void waitPB2()
{
    while (! PUSH_BUTTON);
}
```

5

2. Assuming that fcyc = 20 MHz, complete the following code snippet to require exactly 100 microseconds to execute, assuming the pipeline is full before execution starts. You should calculate the value of N and add up to 10 extra NOPs as needed. Assume that it takes 2 clocks to call the function with BL wait500ms and 2 clocks to return with BX LR when computing the time.

$f_{cyc} = 20 MHz$

How many clocks are needed in total? 10,000,000

1s → 20M clocks

10

500 ms → 10 M

```
.def wait500ms

.thumb
.const
N               .field      2499999        15

.text                                          N= 2499999
                                2 clocks
wait500ms:
                LDR     R0, N           2 clocks
loop:           SUB     R0, R0, #1      N
                CBZ     R0, end         (N-1) + 3
                B       loop            (N-1)* 2
end:






                BX      LR              2

.endm
```

3. Show the settings of the relevant <u>system control</u>, GPIO, and UART registers needed to configure UART2 to communicate with a baud rate of approximately 9600 (as close as possible) assuming a system clock rate of 20 MHz. Assume that 8 data bits are used and there is only one stop bit. Instead of writing code, simple list the register by name and the operation for this configuration in the spaces provided below. Use the scratch paper on the next page as needed.

Usage examples:

ABC_DEF_R = 0x22
ABC_DEF_R |= 0x22
ABC_DEF_R &= ~0x22

U2 Rx = PD6
U2 Tx = PD7

Register Name — Operation

SYSCTL_RCGCUART_R |= 4
SYSCTL_RCGCGPIO_R |= 8
_delay_cycles (3);
GPIO_PORTD_DIR_R |= 128
GPIO_PORTD_DIR_R &= ~64
GPIO_PORTD_DR2R_R |= 128
GPIO_PORTD_DEN_R |= 64|128
GPIO_PORTD_AFSEL_R |= 64|128
GPIO_PORTD_PCTL_R &= ~(0xFF000000)
GPIO_PORTD_PCTL_R |= 0x11000000
UART2_CTL_R = 0
UART2_CC_R = 0
UART2_IBRD_R = 130;
UART2_FBRD_R = 13
UART2_LCRH_R = 0x60|0x10;
UART2_CTL_R = (0x200|0x100|0x1);

4. Show the settings of the relevant system control, GPIO, and timer registers needed to configure TIMER1 to create interrupts at a 60 Hz rate assuming a system clock rate of 20 MHz. Instead of writing code, simple list the register by name and the operation for this configuration in the spaces provided below. Use the scratch paper on the next page as needed.

Register Name	Operation
SYSCTL_RCGCTIMER_R	|= 2
TIMER1CTL_R	&= ~1;
TIMER1_CFG_R	= 0;
TIMER1_TAMR_R	= 2;
TIMER1_TAILR_R	= 333,333;
TIMER1_IMR_R	= 1;
NVIC_EN0_R	|= 1<<(37 - 16);
TIMER1_CTL_R	|= 1;