EEL 4742C: Embedded Systems
Homework 3

QUESTION 1. **(10 points)**

Part a) Complete the table below for the listed interrupt events.

Interrupt event	Enable bit	Flag bit	Register (containing the enable/flag bits)	Vector
TAR rollback to zero				
Channel 0 compare event (TAR = TACCR0)				
Channel 1 compare event (TAR = TACCR1)				
Channel 2 compare event (TAR = TACCR2)				
Port 1 input change (8 events)				

Part b) The "persistent interrupt" is a bug in the code that exhibits a behavior that's similar to an infinite loop. The ISR exits and gets called back immediately an infinite amount of times even though the interrupt event occurred once. The persistent interrupt occurs when three conditions are met. What are these conditions? How to avoid a persistent interrupt from occurring?

QUESTION 2. **(10 points)**

Part a) Write a line of C code that enables the global interrupts.

Part b) Write a line of C code that enables Timer_A to raise an interrupt in the continuous mode (rollback to zero event). You only need to set the appropriate enable bit (xIE).

Part c) Write a line of C code that enables Timer_A to raise an interrupt in the up mode. Do this based on Channel 0's flag. You only need to set the appropriate interrupt enable bit (xIE).

Part d) Write a line of C code that enables Timer_A Channel 1 to raise an interrupt. You only need to set the appropriate interrupt enable bit (xIE).

Part e) Timer_A is configured in the up mode. The Channel 0 interrupt event is enabled. Complete the ISR so that it toggles the LED at P1.4 each time the interrupt is raised.

```
#pragma vector=TIMER_A0_VECTOR
__interrupt void TA0_ISR(){
      ...
}
```

Part f) Timer_A is configured in the continuous mode. The TAR rollback to zero event is enabled for interrupts; when it occurs, the LED at P1.1 should be toggled. Channel 2's compare event is also enabled for interrupts. When this event occurs, the LED at P1.4 should be toggled (TACCR2 doesn't need to be updated). Complete the ISR below.

```
#pragma vector=TIMER_A1_VECTOR
__interrupt void TA1_ISR(){
      ...
}
```

Part g) A button connected to P1.7 is enabled to raise interrupts. When the interrupt occurs, the LED at P1.5 should be toggled. Complete the ISR below. Assume that other pins in Port 1 may be enabled to raise interrupts.

```
#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(){
      ...
}
```

QUESTION 3. **(10 points)**

For the next three questions, choose the low-power mode that minimizes the power consumption.

a) The timer runs on SMCLK and raises periodic interrupts. The responding ISR toggles an LED.
b) The timer runs on ACLK and raises periodic interrupts. The responding ISR toggles an LED.
c) A button's interrupt is enabled and no timer is used. When the button is pushed, the responding ISR toggles an LED.

For the next two questions, the microcontroller doesn't support the low-power mode overriding feature.

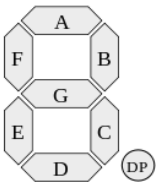
d) The programmer configures the timer to run based on SMCLK and engages LPM3. What happens?
e) The programmer configures the timer to run based on ACLK and engages LPM4. What happens?

For the next two questions, the microcontroller supports the low-power mode overriding feature.

f) The programmer configures the timer to run based on SMCLK and engages LPM3. What happens?
g) The programmer configures the timer to run based on ACLK and engages LPM4. What happens?

QUESTION 4. **(10 points)**

Part a) The memory-mapped register LCDM1 is 8-bit and controls a 7-segment digit on the LCD screen. Its mapping is shown below. A segment is turned on by writing 1 to its bit. The array LCDHexChar[] stores the shapes of the hex digits (shape of 0 at index 0, …, shape of F at index 15). Show the values of LCDHexChar in binary and in hexadecimal. The first one is shown as an example.




```
LCDM1: MSB [DP  G  F  E  D  C  B  A] LSB

unsigned char LCDHexChar[16] = {

      index 0: ___00111111_________    ___0x3F____________

      index 1: ____________________    ____________________

      index 2: ____________________    ____________________

      ...

      index 15: ____________________    ____________________
}
```

Part b) The LCD screen has eight 7-segment digits that are controlled by the registers LCDM1 to LCDM8, all having the same format. LCDM1 controls the rightmost digit and LCDM8 controls the leftmost digit. The registers LCDM1-LCDM8 are located at adjacent memory addresses with LCDM1 having the lowest address. They can be treated as an array. Write a piece of code that turns on all the segments of the seven digits. Start by declaring a pointer so that the registers can be treated as an array.

```
      unsigned char * LCDptr = ...;       // Declaring a pointer;
                                          // Point it to the right address
```

QUESTION 5. **(10 points)**

This question is a continuation of the previous question.

Part a) The value n is one byte, therefore, it can be viewed as two hex digits. Write a C code that extracts the two hex digits from n into the variables shown below. Then, print the digits on the rightmost two segments of the LCD display and ensure the remaining digits are off. Print leading zeros. Use the variables LCDptr and LCDHexChar from the previous question.

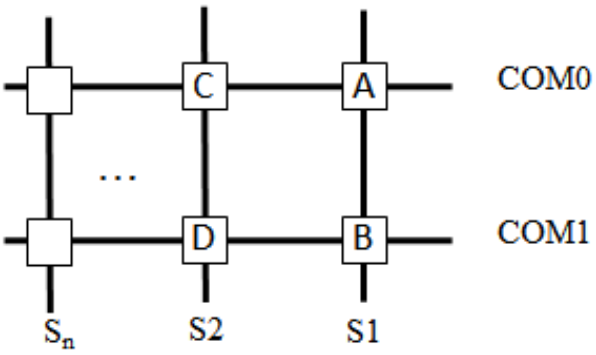
```
unsigned char n = ...;
unsigned char left_digit, right_digit;
```

Part b) The value x is one byte and contains a decimal number that's between 0 and 99. Write a C code that extract the two digits from the value into the variables below. Then print these digits on the LCD display and ensure the remaining digits are off. Don't print leading zeros.

```
unsigned char x = ...;
unsigned char left_digit, right_digit;
```

QUESTION 6. **(10 points)**

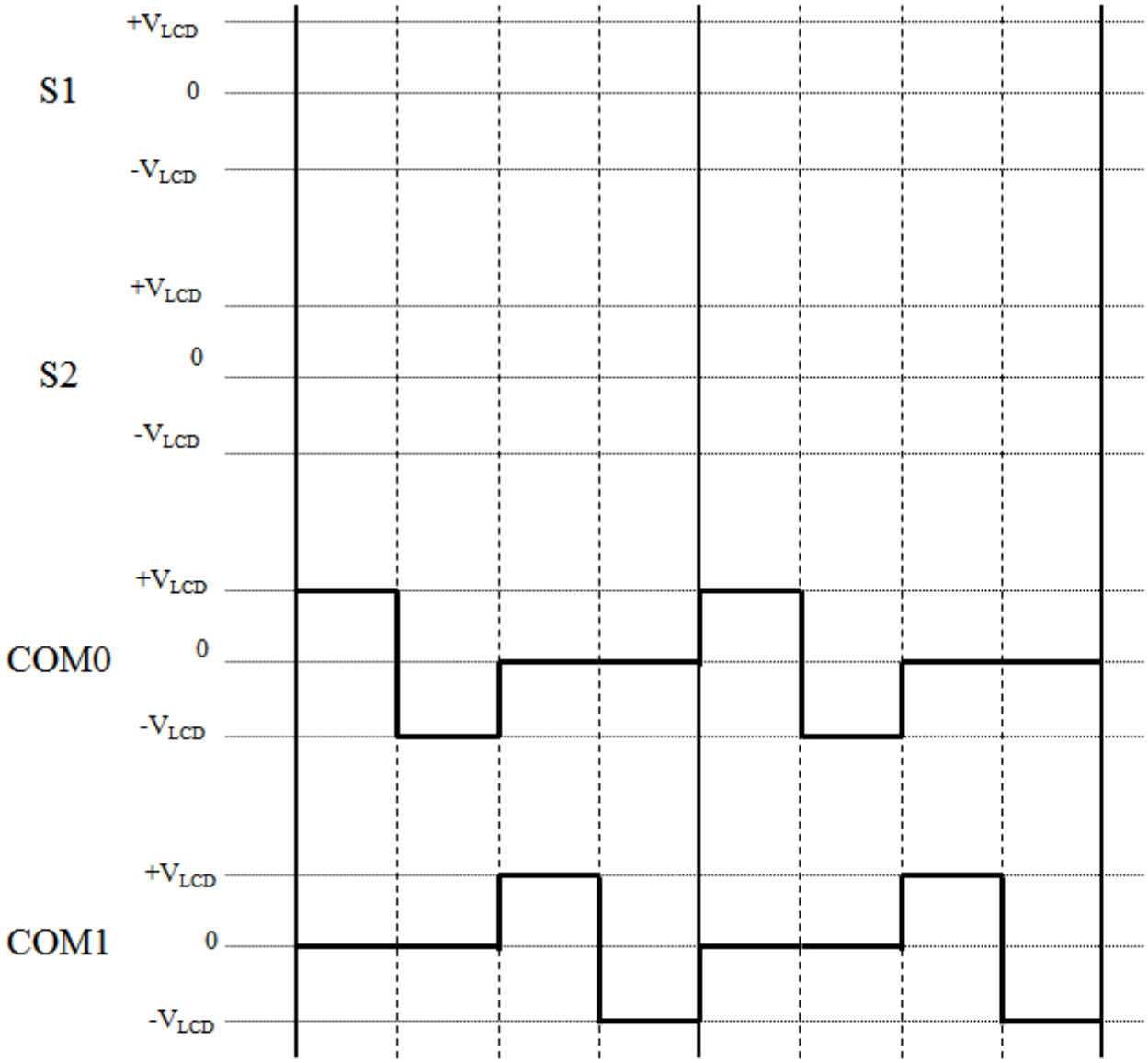
An LCD display is interfaced based on two-way multiplexing as shown below. The segments A and B are wired on the S1 line and the segments C and D are wired on the S2 line.



Draw the S1 and S2 lines on the next figure to control the segments as requested below.

First period:	A on, B off	Second period:	A off, B on
First period:	C off, D off	Second period:	C on, D onn

Figure



QUESTION 7. **(10 points)**

a) An LCD display is interfaced with 4-way multiplexing how many COM lines are there?
b) An LCD display is interfaced with 4-way multiplexing and has 100 segments. How many S lines are there?
c) The COM lines cycle through the same patterns infinitely independent of what's shown on the segments. True or false? Explain.
d) The S lines depend on what's being shown on the segments. True or false? Explain.

The next questions are based on the setup of the previous question.

e) The four segments (A,B,C,D) are controlled with a memory-mapped variable called LCDM1 whose format is shown below. Write two values in LCDM1 to accomplish the values of the first and second period of the previous question. A segment is turned on by writing 1 to its bit in LCDM1. It's okay to write 0's to the leftmost 4 bits.

```
LCDM1:  MSB [ _ _ _ _    A B C D] LSB
```

f) Writing to LCDM1 is a rather straightforward task. Who is responsible for controlling the COM and S lines that you drew in the previous question?

QUESTION 8. **(10 points)**

Part a) Channel 1 is scheduling periodic interrupts every 25K cycles (K=1024). Show how the interrupts are scheduled and demonstrate that the operation is still correct when a rollback to zero occurs on TACCR1. Show enough cases that span two rollbacks to zero.

Part b) Channel 2 is scheduling periodic interrupts every 35K cycles (K=1024). Show how the interrupts are scheduled and demonstrate that the operation is still correct when a rollback to zero occurs on TACCR2. Show enough cases that span two rollbacks to zero.

QUESTION 9. **(10 points)**

Part a) The timer is running in the up mode with a period of 1000 cycles. Channel 1 engages the output mode (Set/Reset) and sets its register TACCR1 = 300. Explain what happens. Draw a timeline.

Part b) Repeat the previous part based on the (Reset/Set) mode.

QUESTION 10. **(10 points)**

When a certain event occurs (e.g. button push), our program needs to time a duration of 0.25 seconds and raise an interrupt when this duration elapses. *(we need to time this interval only after the button is pushed, and not repeatedly)*

Part a) Can we set this up using the up mode? If yes, explain how.

Part b) Can we set this up using the continuous mode? If yes, explain how.

Part c) If both modes are possible, explain the preferable mode.

Part d) If the timer is used intermittently, is it possible to turn the timer on/off throughout the program? How is this done?

Part e) Timer_A is running in the continuous mode and none of its interrupts is enabled. TAR has cycled the full range (0 to 64 K) a few times and the program didn't interact with the timer's bit fields. At this point, what is the value of each of the flags: TAIFG, CCIFG flags of Channels 0, 1, 2?

Part f) In our program, when an interrupt event occurs, it's disabled for a little while (e.g. 3 seconds) and then it's re-enabled. When is the best time to clear the interrupt's flag: when it occurs or when it's re-enabled?

Practice Questions

Do not submit; these questions were solved in the class.

PRACTICE 1.

Part a) Write a code that runs Timer_A in the continuous mode and flashes the LED when TAR rolls back to zero. Use the ACLK clock signal that's based on the 32 KHz crystal. The timer should raise an interrupt and the ISR toggles the LED. The LED is active high and mapped to Port 1.0. Write the main function and the ISR. Use the low-power mode that saves the most power.

Part b) Repeat the question using the up mode with a timer period of 1 second.

PRACTICE 2.

Write a code that toggles the LED when the button is pushed. The button should raise an interrupt and the ISR toggles the LED. The button is active low and is mapped to Port 1.3. Write the main function and the ISR. Use the low-power mode that saves the most power.

PRACTICE 3.

Write a C code that flashes three LEDs using three channels of Timer_A based on interrupts. The timer should use ACLK which is based on a 32 KHz crystal.

- LED1 mapped to P1.0 is toggled every 0.25s using Channel 0
- LED2 mapped to P1.1 is toggled every 0.5s using Channel 1
- LED3 mapped to P1.2 is toggled every 0.75s using Channel 2
