

**EE**

**L**

**47**

**42**

**C**

**:**

**Embedded Systems**

**Homework**

**3**

# QUESTION 1. (10 points)

To answer the questions below, look in the MSP430FR6989 Data Sheet (document: slas789c)

1. What is the FRAM (nonvolatile memory) size? 128 kb
2. What is the SRAM size? 2 kb
3. How many Timer\_A modules with 3 channels does this chip have? 2
4. How many Timer\_A modules with 5 channels does this chip have? 1
5. The eUSCI communication module consists of Channel A (UART, SPI) and Channel B (I2C, SPI). How many eUSCI modules does this chip have? 2
6. What is the “absolute maximum ratings” Vcc value? 4.1 v
7. What is the “recommended operating conditions” Vcc range? 3.0 v
8. Is it acceptable to run the chip based on the absolute maximum ratings? Explain. No because it will reduce the lifespan of the chip and tolerances may cause the supplied voltage to be higher than the maximum rating.

# QUESTION 2. (10 points)

*This question is a continuation of the previous question.*

1. The VLO is an internal RC clock that has the lowest power consumption. What is the current drawn by VLO? What is the typical frequency and min/max frequency values of VLO?
2. The MODOSC (Module Oscillator) is an internal RC clock. What is the current drawn by MODOSC? What is the typical frequency and min/max frequency values of MODOSC?
3. What is the typical resistor value (and min/max values) of the built-in resistor at the pins?
4. What is the maximum current and corresponding power that can be drawn from the pin when Vcc=2.2V? What about Vcc=3.0V?
5. Which vector has a higher interrupt priority Timer0\_A’s A0 or A1 vector? Give the word address for each of these vectors.
6. Which timer has higher interrupt priority, Timer0\_A or Timer1\_A?
7. The variable P1DIR (like many others) is memory-mapped. To what address is this variable mapped? Compute the address in this way: base+offset = address.

# QUESTION 3. (10 points)

1. The rollback-to-zero event, Channels 1 and 2 share the vector A1. Who is responsible for clearing the interrupt flag?
2. The Channel 0 has its own vector. Who is responsible for clearing the interrupt?
3. Code #1 is running and interrupt occurs. Explain what happens regarding saving the status of the CPU.
4. Low-power mode 3 is engaged and an interrupt occurs. Explain what happens regarding saving the status of the CPU.
5. By default, are interrupts in MSP430 preemptable? Explain.
6. How is the interrupt priority in MSP430 determined?
7. What is the ‘reset vector’? Where is it located?
8. A button is interfaced in the active low configuration using the built-in resistor. Should the resistor be pulled-up or pulled-down? What about the active high configuration?

# QUESTION 4. (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 do we avoid a persistent interrupt from occurring?

# QUESTION 5. (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 P1.5 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 P1.5 should be toggled. In the ISR, write an if-statement that positively identify the rollback-to-zero interrupt.

#pragma vector=TIMER\_A1\_VECTOR

\_\_interrupt void TA1\_ISR(){

...

}

Part g) A button connected to P1.2 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 6. (10 points)

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

1. The timer runs on SMCLK and raises periodic interrupts. The responding ISR toggles an LED.
2. The timer runs on ACLK and raises periodic interrupts. The responding ISR toggles an LED.
3. 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.*

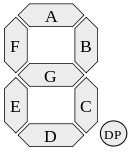
1. The programmer configures the timer to run based on SMCLK and engages LPM3. What happens?
2. The programmer configures the timer to run based on ACLK and engages LPM4. What happens?

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

1. The programmer configures the timer to run based on SMCLK and engages LPM3. What happens?
2. The programmer configures the timer to run based on ACLK and engages LPM4. What happens?

# QUESTION 7. (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 8. (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 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 extracts 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;

# 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.**

The programming model of a communication module is based on the variables below. An 8-bit data is transmitted or received at a time.

|  |  |
| --- | --- |
| TXFLAG bit | 1: ready to transmit; 0: transmission in progress |
| RXFLAG bit | 1: data received; 0: no data received |
| FLAGS register | Contains the flags |
| TXBUFFER register | 8-bit transmit data; writing to this register pulls TXFLAG low and Tx begins |
| RXBUFFER register | 8-bit received data; reading this register clears RXFLAG |

Part a) Write a function that transmits a byte.

Part b) Write a non-block function that receives a byte.