Sample Problem Set #2 - SOLUTIONS

Notes:
These problems are typical exam problems; most are drawn from previous homeworks and exams.
This exam is open book, open notes. It may help to bring a calculator.
For partial credit, please show all work, reasoning, and steps leading to solutions.

The exam covers the following topics:
- RS232
- Memory mapping
- Interrupts
- Timer system (input capture and output compare)
- RTI system
- A/D conversion

You should be familiar with the following registers:
- RTICTL
- CRGFLG, CRGIN
- TSCR1, TSCR2
- TCTL1, TCTL2, TCTL3, TCTL4
- TIOS
- TIE
- TFLG1, TFLG2
- SCIBD, SCICR2, SCISR1

It may be helpful to have a calculator.

1. The following program was designed to count rising edges on pin PT4, and to simultaneously generate a square wave on PM0.

```c
int EdgeCount; // Count of edges

void main(void) {
    int i;

    // Set up timer
    TSCR1 = 0x80; // Enable timer (TEN=1), no fast flag clear (TFFCA=0)

    // Set up input capture on channel 4
    TIOS = 0x00; // Zeros indicate input
    TCTL3 = 0x01; // EDG4B:EDG4A = 01 for rising edge
    TFLG1 = 0x10; // Clear any capture flag (C4F)
    TIE = 0x10; // Enable interrupts on channel 4

    DDRM = 0x01; // Set up PM0 for output
    EnableInterrupts;
    EdgeCount = 0;

    for(;;) {
        PTM ^= 0x01; // complement PM0
        for (i=0; i<100; i++) {} // delay awhile
    }
}
```
(a) The interrupt service routine does not clear the flag. What is the behavior of the program, and what does the programmer see on pin PM0?

The ISR interrupts continuously. The main program never gets a chance to run. PM0 stays constant.

(b) Add instruction(s) inside the interrupt service routine to clear the flag properly.

\[ \text{TFLG1 = 0x10; // Clear C4F flag} \]

(c) What is the maximum number of rising edges that can be counted correctly? How would you increase that number?

Since “EdgeCount” is declared as an integer, which is a 16 bit two’s complement number, the maximum number is 32767. To increase that, declare it as an unsigned long.

(d) The programmer notices that the square wave generated on pin PM0 is not always consistent, but sometimes the period is a little longer than other times. Why?

When interrupts occur, they take time to process, which takes time away from the main program.

2. The following HCS12 program generates a square wave on pin PT0:

```c
void interrupt VectorNumber_Vtimch0 oc0_isr(void) {
    EdgeCount++;
}

(a) Assuming a 24 MHz clock, what is the period of the square wave generated?
```
The timer system is configured for the fastest possible rate, using the prescaler value of 000 (line 9). That means it uses the full 24 MHz clock rate. At 0.042 us per count, 6000 counts is 0.25 ms. The complete period is 0.5 ms.

(b) Describe the output signal if the programmer used “TC0 = TCNT + 6000;” on line 3.
The period of the square wave would not be exactly 0.5 ms, but it would be a little longer, and it would vary slightly. The reason is that TCNT keeps incrementing after the compare action occurs, and so its value on line 3 is a little bigger than what is in TC0.

(c) Describe the output signal if the programmer forgot to enable interrupts (the instruction on line 14).
The ISR would never be executed. However, the pin would still automatically toggle every time the timer register (TCNT) equaled whatever was in the compare register. This would happen every 65536 counts, or 65536*0.042 us = 0.00273 seconds. So a square wave would still be generated, but its period would be 2*0.00273 = 0.00546 seconds.

3. A program is to use the RTI system to periodically schedule a measurement of an external signal. Assume that you have to take a measurement about once per minute. Below is the RTI interrupt service routine. What is the value you would program into the RTICTL register, and what is the value of N, to achieve the desired interval? Assume an 8 MHz oscillator.

```c
void interrupt VectorNumber_Vrti rti_isr(void) {
    CRGFLG = 0x80;  // Clear flag
    if (RTIcount == N) {
        takeMeasurement();
        RTIcount = 0;
    } else
        RTIcount++;
}
```

Solution: Looking at the table of RTI interrupt periods (see Table 6.4 in the textbook), the longest period is 16x2^16 counts, or 2^20 = 1048576 counts (in units of OSCCLK cycles). To get this period, we need to program RTICTL with the value 0x7f.

Using an OSCLK of 8 MHz, each clock tick is 0.125 us. So the period is 1048576*0.125 us = 0.1311 seconds. In one minute (or 60 seconds), we would have 60/0.1311 = 457.76 timeouts. Rounding off to the nearest integer, N = 458.

4. Write C language instructions to set the SCI system on the HCS12 for the following configuration:
- Enable transmitter and receiver
- Baud rate = 115.2 K
- 8 bit characters, no parity bit
- No interrupts

Solution: In C, the code would be

```c
#define BusFreq 24000000
SCIBDH = 0;
SCIBDL = (BusFreq/16)/115200;
SCICR2 = 0x0c;  // Rx and Tx on, no interrupts
// The default value for SCICR1 is 0x00. This configures it for 8 bit characters, no parity bit
```

Note: the compiler knows that SCIBD is a 16 bit integer, composed of SCIBDH and SCIBDL. So you can do SCIBD = (BusFreq/16)/115200;
5. You are to place an I/O device in the memory map of a microcomputer with a 16-bit address bus. The memory map has unused blocks between locations $4000..4FFF$ and $6000..6FFF$. Design an address decoder to place this device somewhere in the memory map using the fewest number of address bits.

Solution: We only need to decode the first hex digit of the address. The address decoder needs to recognize either a 4 (binary 0100) or a 6 (binary 0110). Therefore, the decoder is

\[ /CS = (A15' * A14 * A12') \]

A read or write to anywhere in the unused block will activate the device.

6. The following figure shows an address decoder for an 8-bit address A7..A0. For what addresses will the output signal GO be asserted?

Solution: When A7:A6:A5 = 110, then Q6 of the first decoder is asserted. When A4:A3:A2 = 011, then the output Q3 of the second decoder is asserted. A1 and A0 don’t matter. So the addresses are %11001100 through %11001111, or $CC$ through $CF$.

7. You are to design a program which detects an input pulse and outputs a pulse as quickly as possible after that. Should you use interrupts or polling for the fastest possible response time, and why? (Assume that the program doesn’t have to do anything else.)

Solution: You should use polling, because the ISR has a delay due to saving the registers and fetching the vector.

8. The following is a voltage versus time waveform of an RS-232 signal. Assume the waveform is the transmission of two ASCII characters. What characters are being transmitted? Assume odd parity, with one start bit and one stop bit. Each time tick represents one bit cell.
Solution: The bits are 0 10110000 1 0 01010001 1, where start (0) and stop (1) bits frame each character. Since LSB is transmitted first, the actual data characters are 0000 1101 and 1000 1010. We ignore the leading odd parity bit. Thus we get the data codes $0D and $0A. These are the ASCII codes for CR and LF, respectively.

9. Draw the voltage versus time waveform of the RS-232 signal for the transmission of the ASCII characters “51”. Assume 19200 baud, odd parity, with one start bit and two stop bits. Assume 7 data bits for the ASCII characters (so that a total of 11 bits are transmitted, counting the parity bit, and start and stop bits). Indicate voltage levels and times on your sketch.

Solution: The ascii codes for "5" and "1" are 10110101 and 00110001, with odd parity. Transmit the LSB's first. Each time tick represents 1/19200 = 0.052 ms.

\[+10V\]
\[-10V\]

\[\text{Start} \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad \text{Stop} \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad \text{Stop}\]

\[+10V\]
\[-10V\]

\[\text{Start} \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad \text{Stop} \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad \text{Stop}\]

\[+10V\]
\[-10V\]

\[\text{Start} \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad \text{Stop} \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad \text{Stop}\]
10. The following C statements are used to initialize the free-running counter (TCNT) timer system.

\[
\begin{align*}
TSCR1 &= 0x80; \\
TSCR2 &= 0x04;
\end{align*}
\]

(a) If these statements are executed, what is the duration of one count in the TCNT register? Assume a 24 MHz E-clock.

Solution:

\[
\begin{align*}
TSCR1 &= 0x80; \quad // \text{turn on timer system} \\
TSCR2 &= 0x04; \quad // \text{divide E clock by 16 to prescale for TCNT}
\end{align*}
\]

\[
24 \text{ MHz/16} = 1.5 \text{ MHz}. \quad \text{So the period of one count is 0.667 microseconds.}
\]

(b) The following C function can be used to time a short delay.

\[
\text{void wait(} \text{unsigned int delay)}\{ \\
\quad \text{unsigned int startTime;} \\
\quad \text{startTime} = \text{TCNT}; \\
\quad \text{while}((\text{TCNT} - \text{startTime}) \leq \text{delay})\{}
\}
\]

What parameter should you pass in to function \text{wait()} in order to implement a delay of 5 milliseconds?

Solution: \(5 \text{ ms}/0.667 \text{ us} = 7500\)

(c) What is the longest time you can delay using function \text{wait()}? Assume that you still initialize the timer in the same way.

The longest time is achieved by passing in the largest value for the parameter, which is 65535. This would produce a delay of \(65535 \times 0.667 \text{ us} = 0.04369 \text{ seconds}\).

11. A transducer is to be used to measure temperature from –10 to 70 degrees C. We need to display the temperature to a resolution of plus or minus 1 degree. The transducer produces a voltage from 0 to 5 volts over this temperature range, with 5 millivolts of noise. Specify the number of bits in the A/D converter (a) based on the dynamic range and (b) based on the required resolution.

Solution:

(a) The dynamic range is the span divided by the noise, or \((5 \text{ V})/(0.005 \text{ V}) = 1000\). Thus, we need at least 1000 steps, which we can get with an 10 bit A/D converter (this would actually give us 1024 steps).

(b) The required resolution is 80 degrees / 1 degree = 80. A 7 bit A/D converter would meet this requirement.

12. What value should be written into TCTL1 to toggle the voltage on the PT5 pin on successful output compares?

You should write a \%----01--\, or a \$04

13. Say that you use the input capture system to measure the frequency of an input signal. What is the lowest frequency that you can measure, assuming that the crystal frequency is 24 Mhz, and there is no scaling of the timer counter?
With no scaling of the timer, you can have a period of up to 65536x0.042 us, or 2.73 ms. This corresponds to a frequency of 366 Hz.

14. An 8-bit analog-to-digital converter has a sample time (i.e., the time that it is looking at the input signal) of 0.1 µs. What is the maximum frequency of the input signal, such that the digitization error is no more than ±1/2 LSB?

\[ t_{AP} = \frac{1}{(2 \pi f_{max} 2^N)} \]

\[ f_{max} = \frac{1}{(2 \pi t_{AP} 2^N)} = \frac{1}{(2 \pi 0.1 \text{ us} * 2^8)} = 6217 \text{ Hz} \]

15. A microcontroller can be used to measure the capacitance of an unknown capacitor, using a resistor of known value, and its A/D converter. Assume you have the connections as shown.

Describe in detail the program you could write to determine the capacitance (recall the exponential decay formula for an RC circuit, \( V = V_0 e^{-t/RC} \)). Just describe the program in words or give pseudocode; it is not necessary to write C or assembly language instructions.

Solution:

1. The MCU outputs a logic high on PT0 to charge up the capacitor. Keep it high a sufficiently long time
2. The MCU outputs a logic low on PT0 to discharge the capacitor, and starts a timer. For example it could capture the value of TCNT, and count the number of overflows.
3. The MCU measures the voltage on AN0. When it falls to a given level, note the time elapsed. For the level, use say, 5V * e^{-1}.
4. Compute the capacitance using \( C = \frac{t}{R} \)
   Or you could let it go down to 1 volt, and then determine C via
   \[ C = -t / R \ln(1 \text{ volt} / 5\text{volts}) \]

Note – it may also be possible to calculate C by generating an AC signal, and looking at the complex impedance.