Nolan Anderson

CPE/EE 323 Introduction to Embedded Computer Systems
Homework V

1 (15)	2 (25)	3 (25)	4 (20)	5 (15)	Total

Problem #1 (15 points) Exception Processing Module 6

Answer the following questions.

A. (2 points) Illustrate how the top of the stack should look like at the moment just before RETI is executed.

0x2400	TOS
0x23FE	PC
0x23FC	SR

B. (5 points) The MSP430F5529 receives interrupt requests from the watchdog timer in the interval mode (WDT) and parallel ports P1.7 and P2.2 during execution of an instruction that takes 5 clock cycles to execute. P1 is received in the 2nd clock cycle of the instruction execution, P2 is received in the 3rd clock cycle, and WDT in the 4th clock cycle. Which ISR is accepted and processed first, assuming that WDTIE=1, P1.IE=0x80, P2.IE=0x04, and GIE=1? Hint: Inspect the MSP430F5529 IVT for determining priorities. 64 FFFE

WDTIE=1
P1.IE=0x80
P2.IE=0x04
GIE=1

P1.7= 5cc, 2nd cc
P2.2 = 5cc, 3rd cc
WDT = 4th cc, ?

whatchdog goes first, and then Pin 1 and then Pin 2.

For the following questions assume that the interrupt vector table has 16 entries (0 – 15).

C. (3 points) What is the address range of the interrupt vector table (start-end)? OFFFEh -> OFFE0h

D. (3 points) What is the address of the interrupt vector with the entry number 3?

0h FFF8

E. (2 points) How does the interrupt vector table get initialized and when?

The IVT is initialized by the assembly language programmer of compiler. It is loaded at the same time the program is loaded into Flash memory.

Problem #2. (25 points) Interrupts module 6

An MSP430-based system interfaces 4 external devices (ED0, ED1, ED2, ED3), each capable of generating an interrupt request. The external devices place a request by setting the request line (a transition from a logic one to a logic zero). The request lines are connected to port 1 pins P1.3 (ED0) and P1.4 (ED1), and port 2 pins P2.3 (ED2) and P2.4 (ED3). A request line is kept active as long as the interrupt request is pending, until the request is serviced. Answer the following questions.

A (5 points) Specify the registers that need to be initialized at the beginning to configure the system for accepting the interrupts from the devices ED0-ED3. Fill in the table below. Note: to specify interrupts active on the falling edge the edge-selection bits should be set to 1.

Register	Full Name	Content after initialization [binary contnet B7… B0], b – unknown (unchanged)
P1IE	Port 1 interrupt enable	[---1 1---]
P1IES	Port 1 interrupt edgeselect	[---1 1---]
P1IFG	Port 1 interrupt flag	[---0 0---]
P2IE	Port 2 interrupt enable	[---1 1---]
P2IES	Port 2 interrupt edge select	[---0 0---]
P2IFG	Port 2 interrupt flag	[---1 1---]
SR.GIE	SR-Global interrupt enable	0x01

0000 0
0001 1
0010 2
0011 3
0100 4
0101 5
0110 6
0111 7

B. (10 points) How many ISRs are needed to process interrupts from ED0-ED3. Outline the service routine (or routines if you need multiple ones) under the following conditions. If multiple requests occur at the same time, ED0 should have the highest priority and ED3 the lowest priority. Once in the service routine, you could service more than one pending request, but the highest priority one is serviced first.

2 ISR's will be needed, one for each port.

```
0  P1_ISR
1  IF ( P1.IFG[BIT4]):
2      Clear P1.IFG[BIT4]
3      Process ED0 service
4  Else if (P1.IFG[BIT3]):
5      Clear P1.IFG[BIT1]
6      process ED1 service
7  RETI
```

```
0  P2_ISR
1  IF ( P1.IFG[BIT4]):
2      Clear P2.IFG[BIT4]
3      Process ED2 service
4  Else if (P2.IFG[BIT3]):
5      Clear P2.IFG[BIT3]
6      process ED3 service
7  RETI
```

C (10 points) Assume that processing request from each peripheral takes ~3 ms. The processor is in a low-power mode when not executing service routines. The following table describes a sequence of events in time. Fill in the table by answering what happens with the MSP430 on each relevant event if we know the following: ED2 raises an interrupt request at 13 ms, ED3 at 14 ms, and ED1 at 18 ms, and ED0 at 20 ms. The number of rows does not reflect the final solution.

Time	Event	MSP430 status
0 ms	SW initialization	Active (run initialization software)
10 ms	Go to a low-power mode	Sleep
13 ms	Request from ED2 is received, pending&accepted	P2_ISR entered (ED2 portion executed, 3 ms to go)
14 ms	Request from ED3 received	P2_ISR entered (ED2 portion still executing 2ms to go. ED3 pending.
16 ms	P2_ISR finished	main P2 interrupt pending
16ms	Enter P2_ISR	P2_ISR entered
18ms	Request from ED1 received	P2_ISR entered ED3 portion executing 1ms to go. ED1 pending
19ms	P2_ISR finished	main P1 interrupt pending
19ms	Enter P1_ISR	P1_ISR entered ED1 serviced.
20ms	Request from ED0 received	P1_ISR entered (ED1 executing 2ms to go) ED0 pending
22ms	P1_ISR finished	main (P1 interrupt pending)
22 ms	Enter P1_ISR	P1_ISR entered
25 ms	P1_ISR finished	Stop

Problem 3. (25 points, TimerB, Watchdog Timer) Module 9

Consider the following code segment that utilizes the watchdog timer in the interval mode with a period set in line 4 of the code.

```
1.  #include <msp430xG46x.h>
2.  void main(void) {
3.    int p = 0;
4.    WDTCTL = WDT_ADLY_250; // check the meaning of this constant in the include file
5.    P2DIR |= BIT2;         // Set P2.2 to output direction
6.    P2OUT &= ~BIT2;        // ? Turns off the 2.2
7.    for (;;) {             // Inf. loop
8.      if ((IFG1 & WDTIFG) == 1) {
9.        p++;                                    //increment p.
10.       IFG1 &= ~WDTIFG;
11.       if (p == 7) P2OUT ^= BIT2;              //toggles P2.2 output when p=7
12.       if (p == 15) { P2OUT ^= BIT2; p=0;}     //toggles P2.2 output when p=15
13.     }
14.   }
15. }
```

A. (5 points) What does the code segment do? What does the code in line 10 do?

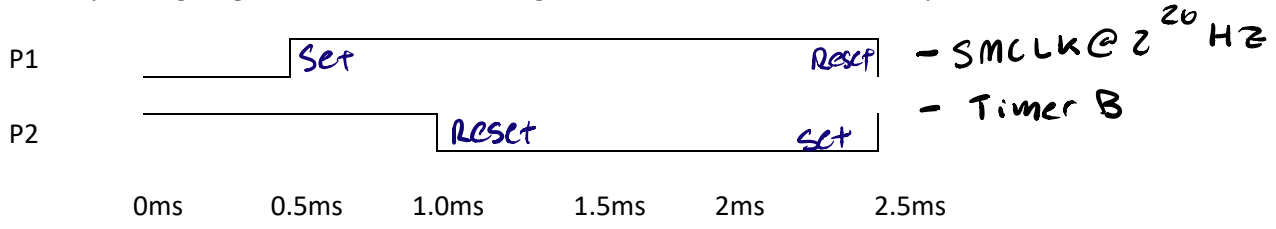
p is incremented every 250 ms. It will toggle P2.2 at 1.75s and 3.75s and remain off for 2s.

Line 10 clears WDTFG so it can return counting.

B. (10 points) How would you implement the given functionality using an interrupt service routine.

```
229  void main(void)
230  {
231      WDTCTL = WDT_ADLY_250;
232      P2DIR |= BIT2;
233      P2OUT &= ~BIT2;
234      P2IE |= BIT2;
235      P2IES |= BIT2;
236      P2IFG &= ~BIT2;
237      while(1)
238      {
239          if ((IFG1 & WDTIFG) == 1)
240          {
241              P2IFG |= BIT2;
242          }
243      }
244  }
245
246  #pragma vector = PORT2_VECTOR
247  __interrupt void PORT2_ISR(void)
248  {
249      int p = 0;
250      p++;
251      IFG1 &= ~WDTIFG;
252      P2IFG &= ~BIT2;
253      if (p == 7) P2OUT ^= BIT2;
254      else if (p == 15) { P2OUT ^= BIT2; p = 0; }
255  }
256
```

C. (10 points) You would like to generate two periodic pulse-width modulated (PWM) signals P1 and P2, with frequency of 400 Hz (one period is 2.5 ms). Assume that an 2^20 Hz clock on SMCLK is used by TimerB. Can you do this using TimerB? If yes, describe a TimerB configuration (content of control and data registers) that will carry out signal generation? Note: use English and waveforms to describe your solution.



- SMCLK@ 2^{20} Hz
- Timer B

```
Yes we can use TimerB in UP MODE to generate PWM modulated signal. For a period of 2.5 ms, the signal is
reset from 0 ms to .5 ms and set from .5 ms to 2.5 ms. We need TBCCR1 and TBCCR2 to create signals for P1 and P2.
TBCCR0 = 2.5ms / 1us = 2500
TBCCR1 = 0.5ms / 1us = 500        // Set [.5ms] / Reset [2.5ms] and repeat cycle.
TBCCR2 = 1.0ms / 1us = 1000       // Reset [1.0ms] - Set [2.5ms] and repeat cycle.
```

Problem 4. (20 points), Clocks Time, Timers Module 9

Consider the following code segment. Assume that processor clock in the active mode is set to 2,000,000 Hz.

```
1.  while(1) {
2.     int i;
3.     for(i = 800; i>0; i--);          // one loop iteration takes 5 clock cycles
4.     P3OUT |= BIT5;                     // Set P3.5
5.     for (i = 800; i>0; i--);          // Delay
6.     P3OUT &= ~BIT5;                    // Clear P3.5
7.  }
```

A. (5 points) What does the code segment do assuming that P3.5 is configured as a digital output. You may ignore delay needed to execute instructions in lines 1, 4 and 6.

- Active mode: 2,000,000 Hz
- One loop iteration : 5cc's.
- 800 x5= 4,000

4,000/2,000,000 = .002ms

On for 2 ms, off for 2ms.

B. (5 points) What will happen if you connect P3.5 to the buzzer?

This code will infinitely beep a buzzer at 4ms that is connected to p3.5.

C. (10 points) How would you implement functionality achieved by the code segment above using TimerB. Port P3.5 is multiplexed with the output signal from the capture and compare block 4 of TimerB. Give details. How would you initialize the system? What would you do in the main loop? Assume the SMCLK is used which is 2^20 =1,048,576 Hz.

$\frac{(\text{loop i})}{(1,048,576)} = .002$ loop i ≈ 2097. Hence the # in TimerB_Setup

```
200  void TimerB_setup(void)
201  {
202      TB0CCTL0 = CCIE;                // Enabled interrupt
203      TB0CCR0 = 2097;                 // 2097 / 1048756 Hz = 2ms
204      TB0CTL = TBSSEL_1 + MC_1;       // ACLK, up mode
205  }
206
207  void main(void)
208  {
209      WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
210      __enable_interrupt();       // Enable interrupts globally
211      P3DIR |= BIT5;              // P3.0 is output direction for Pin 5
212      P3REN |= BIT5;              // Enable the pull-up resistor at P3.5
213      P3OUT &= ~BIT5;             // Device is off to start.
214      TimerB_setup();             // Setup Timer B for 10 times a second
215  }
216
217  #pragma vector = TIMER0_B0_VECTOR
218  __interrupt void timerB_isr()
219  {
220      int i;
221      for(i = 800; i > 0; i--);   // Delay, same as sample code since the timer B has been setup accordingly.
222      P3OUT |= BIT5;              // Turn on the device.
223      for(i = 800; i > 0; i--);   // Delay.
224      P3OUT &= ~BIT5;             // Turn off the device.
225  }
```

Problem 5. (15 points) UART Serial Communication Module 10

Consider the following C source code (assume that P5.BIT1 is connected to a LED).

```
#pragma vector=USCIAB0RX_VECTOR ----> receive ISR
__interrupt void USCIA0RX_ISR (void)
{
  P5OUT |= BIT1;                  // Set P5.1 as output
  while(!(IFG2&UCA0TXIFG));       // Wait for transmit
  UCA0TXBUF = UCA0RXBUF;          //Echo out receive character
  while(!(IFG2&UCA0TXIFG));       // Wait for transmit
  UCA0TXBUF = UCA0RXBUF;          //Echo out receive character
  while(!(IFG2&UCA0TXIFG));       // Wait for transmit
  UCA0TXBUF = UCA0RXBUF;          //Echo out receive character
  P5OUT &= ~BIT1;                 // Turn off output
}
```

A. (5 points) What does this code segment do? USCIA0 is configured in the UART mode.

- On a received character, this ISR is called. The ISR turns on P5.1 and echos the same 3 characters to the terminal and turns off P5.1.

B. (5 points) USCI0 is configured in the UART mode to transfer 57,600 bits/second, 8-bit characters, odd parity, and two stop bits. How many processor clock cycles (MCLK = 2^20 Hz) expire during the time USCI needs to send one character over the UART?

USCI0 = 57,600 BPS

12 Bits per char

MCLK = 2^{20} Hz

$\frac{12\ BPC}{57{,}600} = .208\ ms/character$

$1/2^{20} = .000954\ ms/clock\ cycle$

$.2083 / .000954 = 219\ cc/character$

C. (5 points) Describe how to configure USCI to achieve the desired communication speed from part B using the low frequency UART mode and SMCLK as the source clock, SMCLK=MCLK, (specify values of all important control registers and individual bit fields in these registers).

```
258  void UART_setup(void)
259  {
260      P3SEL |= BIT3 + BIT4;       // Set USCI_A0 RXD/TXD to receive/transmit data
261      UCA0CTL1 |= UCSWRST;        // Set software reset during initialization
262      UCA0CTL0 = 0;               // USCI_A0 control register
263      UCA0CTL1 |= UCSSEL_2;       // Clock source SMCLK
264
265      UCA0BR0 = 0x09;             // 1048576 Hz  / 115200 lower byte
266      UCA0BR1 = 0x00;             // upper byte
267      UCA0MCTL = 0x02;            // Modulation (UCBRS0=0x01, UCOS16=0)
268
269      // Clear software reset to initialize USCI state machine
270      UCA0CTL1 &= ~UCSWRST;
271  }
```

not part of my final answer

You want to match the speed of communication from part B, you must Divide the upper byte by the lower to get the desired speed.

MCLK = 2^{20} Hz = 1048576

UCA0BR0 = $\frac{2^{20}}{X}$ = 57,600

UCA0BR0 = $x \approx 18.2$, round up to 19.

19 = 0x13 in hex

So replace line 265 with: UCA0BR0 = 0x13

-> UCA0BR1 Remains the same