CmpE213 – Digital Systems Design

Homework 10
Interrupts & timers

For any programs below, please email me a copy of your completed code. Send them as attachments to an email with subject line "HW10 - program". Please send all programs for this assignment in a single email (as opposed to a separate email for each program).

1. Create a timer-driven interrupt to produce a pulse-width modulated signal on P1^3. The on and off time for this pulse-width modulated signal will be stored in global variables char high_time and char low_time, respectively. Variable high_time specifies how many microseconds the signal should be high and low_time specifies how many microseconds it should be low. Give your program the following characteristics:

   - **FOREGROUND**. In the main function (in the foreground),:
     (a) Initialize the timer
     (b) Initialize a timer-based interrupt
     (c) Initialize high_time and low_time to some appropriate values (say 0x42 and 0x2A).
     (d) Execute an infinite loop (it doesn't really have to do anything of importance)
   - **BACKGROUND**. Create a timer-driven interrupt which operates in the background. This interrupt will:
     (a) Reload the timer to cause an interrupt at the next appropriate time. (i.e. if beginning
     (b) set the output bit P1^3 high or low, depending on if the PWM is beginning a high-state or low-state.
     (c) Quit – Return control to the task operating in the foreground.

   For example, let's say we want a PWM signal with period of 100uS and a duty cycle of 25%. From main(), we might make high_time=25 and low_time=75 (period=25+75=100 machine cycles which equals 100uS with a 12MHz clock. Duty-cycle=25/100=25%). The first time the interrupt is called, it a) will set the timer to overflow in 25 machine cycles, b) will set P1^3 high, and c) will exit. In 25 machine cycles (25uS with a 12MHz clock), the interrupt will occur again. This time, the interrupt a) will set the timer to overflow in 75 machine cycles, b) will set P1^3 low, and c) will exit. In 75uS when the interrupt occurs, it will again set the timer to overflow in high_time machine cycles, will set P1^3 high, and will exit. And so-forth.

Simulate your code using Keil uVision. Test the operation of the PWM output by watching P1^3 - check for both duty-cycle and period.

Email me: Your code and anything else I need to quickly compile and test your program.

Hand in: A hardcopy of your code.

2. Write ASM code equivalent of the C-program in problem 1. Be sure to set aside space for and initialize your stack. Email me the ASM code and turn in a hard-copy.

3. Write a short C-program which writes 1 byte of data to the serial port. You do not have to simulate this program, but it does have to compile.

   - Send 8 bits of data and a 9th "parity" bit (Look in PSW for parity).
   - Send data at 5,280 baud assuming a 4.2MHz clock
   - Send a single byte, 0x79

4. For the above problem, show the signal that would result on TxD. Please calculate the time-interval, T, over which each bit is sent and show this on your plot. Remember that the PSW calculates even parity – if the ACC has an odd number of "1"s, parity will be 1, making total number of "1"s an even number.

5. I was going to also have you write a program which read and wrote data through the serial-port using an interrupt, but decided you already have enough to do. Make sure you learn this for the test, though. (Don't turn this problem in!)

   The problem would have looked something like this: Write an interrupt which sends and receives data over the serial port. Give it the following characteristics:

   - The interrupt should determine if a send or receive interrupt caused it to be activated (The same interrupt is generated on transmit and receive).
   - When a receive interrupt comes in, write received data to a global array called rdata. Keep track of where the next data should be placed in the array using a global variable rbytes.
   - When a transmit interrupt comes in, send bytes of data from a global array called tdata. Keep track of your current position in tdata using a global variable tbytes. For this exercise, you can specify the contents of tdata in code as a constant.
   - Send/receive data at 19,200 baud. (Set the appropriate mode for both the serial port and for the appropriate timer).
   - Use 8 data bits

   For now, do nothing in main. In a real program, you would read received data from rdata (changing rbytes appropriately when you did so). You would write data you wished to transmit to tdata and would then trigger the interrupt to start the transmission (by setting TI in SCON), if the 8051 was not already transmitting data.