Name: Lab Section:

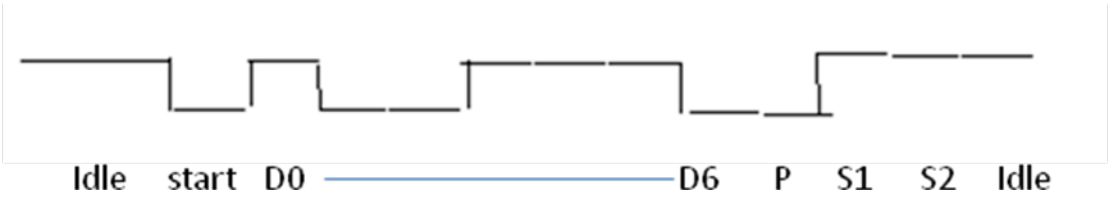
CprE 288 Fall 2012 – Homework 6
Due Thu. Oct. 11 in the class

Notes:

- **Start early on homework.**
- Homework answers must be typed using a word editor. Hand in a hard copy in the class.
- Late homework is accepted within three days from the due date. **E-mail late homework to both of the grading TAs, Min Sang Yoon (my222@iastate.edu) and Zhen Chen (zchen@iastate.edu)**. *Late penalty is 10% per day (counting from 10:45am of the due date if you are in the morning class or 2:10pm if you are in the afternoon class).*

Question 1: USART Basics (10pts)

Sketch the waveform appearing at the output of the USART when it transmits a character '9' at 9,600 baud rate (see the top half of slide 8 of week 6). The sketch should show the bit durations in microseconds, in addition to the waveform. The frame format is of 1 start bit, 7 data bits, an even parity bit, and 2 stop bits.



The length of one bit is 1/9,600 second = ~104 microseconds.

Question 2: USART Programming (20pts)

You are asked to program an ATmega128 so that it plays as a USART reply node. Its USART0 unit connects to a USART channel #0, and its USART1 unit connects to another USART channel #1. Any character from channel #0 should be copied to channel #1, and any character from channel #1 should be copied to channel #0. The program must be interrupt-based, so the microcontroller can do other work.

a. [10] Assume the ATmega128 runs at 16MHz. Write a USART_init() function to initialize both USART units as follows:

- Two-way asynchronous communication
- 125,000 baud rate
- 6 Data bits
- Odd Parity

- 1 Stop bits
- An interrupt enable bit is set if and only if that is necessary.

You may initialize unrelated control bits as you wish.

```
void USART_init()
   {
        #define baud_setting (16000000/(16*125000)-1)
        UBRR0H = baud_setting >> 8;
        UBRR0L = (unsigned char) baud_setting;
        UBRR0H = baud_setting >> 8;
        UBRR0L = (unsigned char) baud_setting;

        // enable receive, transmit and interrupt on receive
        // RXEN <= 1, TXEN <= 1, RXCIE <=1
        UCSR0B = UCSR1B = _BV(RXEN) | _BV(TXEN) | _BV(RXCIE);

        // async, 6 data, 1 stop bit, odd parity
        // UMSEL <= 0, UCSZ2:0 <= 001, USBS <= 0,
        // UPM1:0 <= 11; note UCSZ2 is in UCSRxB
       UCSR0C = UCSR1C = _BV(UCSZ0) | _BV(UPM1) | _BV(UPM0);
   }
```

b. [10] Write the interrupt handlers that do the copying. You may assume that, since the two USART units are running at the same baud rates, overflow should not happen. However, if it does happen, i.e. a USART unit is not ready for transmitting when a new character comes, then the new character should be discarded. Your code should also check for frame error and parity error, and discard any character that causes any of those errors.

Note: Reading the UDR*n* will clear the RXC flag and consequently the interrupt signal (if the character size is under 8 bits).

```
   ISR (USART0_RXC_vect)
   {
       char ch = UDR0;      // receive from USART0

       // check for frame error and parity error
       if (UCSR0A & (_BV(FE) | _BV (UPE)))
           return;

       // send to USART1 if it's ready; otherwise, it's lost
       if (USCAR1A & _BV(UDRE1))
```

Name: Lab Section:

```
        UDR1 = ch;        // send to USART1
}

ISR (USART1_RXC_vect)
{
   char ch = UDR1;        // receive from USART1

   // check for frame error and parity error
   if (UCSR1A & (_BV(FE) | _BV (UPE)))
      return;

   // send to USART1 if it's ready; otherwise, it's lost
   if (USCAR0A & _BV(UDRE0))
       UDR0 = ch;              // send to USART0
}
```

Question 3: ADC Design Principle (10 pts)

Suppose that an ATmega128 is used with a pressure censor to monitor the gas pressure exerted on a valve. The pressure sensor measures pressure in range of 20.0psi (pounds per square inch) to 220.0psi and convert it proportionally to an electrical signal in voltage range from 0V to 2.4V. The reference voltage of 2.56V is used.

a. If the gas pressure is 100.0 psi, what is the voltage level at the sensor's output? What is the digital reading from the ATmega128's ADC?

(Solution revised)
Let V'max = 2.4V, and Vmax=2.56
From gas pressure to voltage:
v/V'max = (100.0psi-20.0psi)/(220.0psi-20.0psi) = 0.4, v=V'max*0.4 = 0.96V
From voltage to ADC reading:
d/M = v/Vmax = 0.96V/2.56V, M=1024, d = 384.

b. If the digital reading is 240, what is the range of the analog value? Note that each digital number corresponds to a step (a small range in the span) of the analog value.

(Solution revised)
From ADC reading to voltage
d/M = v/Vmax, here v is the low end of the range.
240/1024 = v/Vmax, v = 240/1024*2.56 = 0.6V,
step size = 2.56V/1024 = 0.0025V, voltage range is 0.6V-0.6025V
From voltage to pressure
(p-20.0psi)/(220.0psi-20.0psi) = 0.6V/2.4V, p = 70.0psi

Name: Lab Section:

Step size = 0.0025V/2.4V * (220psi-20psi) = 0.2083psi, range is 70-70.2083psi
Note: If one gives one answer for either voltage or pressure, it's OK.

Question 4: ADC Design Principle (10 pts)

A 4-bit ADC (of 16 steps in the analog range) uses the Successive Approximation implementation. The input voltage range is 0V-16V. If the input is 12V, how does the ADC work out each bit of the digital encoding? Fill the following table to show the steps (as did in the class). The first step is given.

Step	Range	DN_Mid	AV_Mid(V)	Input >= AV_Mid
0	xxxx	1000	8	1
1	1xxx	1100	12	1
2	11xx	1110	14	0
3	110x	1101	13	0
4	1100			

The digital value is __1100___ (binary) and __12___ (decimal).

Question 5: ATmega128 ADC Programming (5 pts)

Describe how to use the ADSC bit in ADCSRA to start an ADC conversion, and to wait for the ADC conversion result to be ready.

Question 6: ATmega128 ADC Programming (10 pts)

Assume ATmega128 platform with the system clock configured to **4MHz**. Write two C statements to configure the ADC as follows:

- Interrupt enabled
- One-shot mode
- ADC clock rate in the range of 50K-200K Hz
- Reference voltage is 2.56V
- No differential input
- ADCW right adjusted
- The initial input channel is 0

Name: Lab Section:

```
ADC_init()
{
  // REFS=11, ADLR=0, MUX=00000
  ADMUX = BV(REFS1)|BV(REFS0) ;

  // ADEN=1, ADSC=0, ADFR=0, ADIF=0, ADIE=1, ADPS=101
  ADCSRA = _BV(ADEN)|_BV(ADIE)|_BV(ADPS2)|_BV(ADPS0);
}
```

Alternatively (not preferred):

```
ADC_init()
{
  // REFS=11, ADLR=0, MUX=00000
  ADMUX = 0b11000000;

  // ADEN=1, ADSC=0, ADFR=0, ADIF=0, ADIE=1, ADPS=101
  ADCSRA = 0b10001101;
}
```