Question 1: USART Basics (10pts)

a) Sketch the logic waveform appearing at the output of the USART when it transmits a character ‘T’ at a baud rate of 9,600. The sketch should show the bit durations in microseconds, in addition to the waveform. The frame format is of 1 start bit, 8 data bits, an odd parity bit, and 2 stop bits. [7 pts]

ASCII ‘T’ = 0x54 =0b0101 0100

Note: UART transmits the least significant bit first.

Gading Note: -1pts for incorrect bit order

From datasheet an Odd Parity bit is computed as:

$$P_{\text{odd}} = d_7 \oplus d_6 \oplus d_5 \oplus d_4 \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

$$= 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1$$

$$= 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1$$

$$= 0 \oplus 1 \oplus 1$$

$$= 0$$

$$P_{\text{odd}} = 0$$

Bit duration(Bit Period) in microseconds = $1/9600 = 104.17us$
b) What is the data rate of the USART configuration given in part a?

Data rate is the fraction of the Baud Rate for transmitting data bits. 
Number of bits in a frame: 1 start + 8 data + 1 parity + 2 stop = 12 bits

Data Rate = BAUD_RATE * (8 data_bits / 12 frame_bits)  
Data Rate = 9600 * 8/12 = 9600 * .6667 = 6400 data bits / second

Question 2: USART/ADC Programming (20pts)

You are assigned to a programming task as follows. Recall that the ATmega128’s ADC has eight channels. In this application, each channel is connected to a different analog sensor, and USART0 of the ATmega128 is connected to a PC. A program running on the PC periodically sends a channel number to the ATmega128 over USART0. Program the ATmega128 such that it receives this input from USART0, starts an ADC conversion for the specified channel, and then sends back the ADC conversion result; and then waits to receive a channel request.

The channel number is transmitted in an 8-bit raw format (i.e. not ASCII). For example, channel 0 is encoded as 0b00000000, 0x00, 0. The ADC conversion result is transmitted back in a raw format as well, but only the most significant 8 bits are sent; the other two bits are discarded. Recall that an ADC conversion result is 10-bit.

Assume the ATmega128 runs at 16MHz. You are NOT allowed to use interrupts. The following is the main function of the program:

```c
#include <avr/io.h>

void serial_init();
void adc_init();
void poll_and_exec_cmd();

int main()
{
    adc_init();
    serial_init();

    while (1)
    {
        poll_and_exec_cmd();
    }

    return 0;
}
```
a. [6] Write the serial_init() function to initialize USART0 as follows:

- Asynchronous communication
- Receive and transmit
- 57,600 baud rate
- 8 data bits
- Even parity
- 2 stop bit
- All interrupt disabled.

You may initialize unrelated control bits as you wish.

```c
void serial_init()
{
    // Set baud rate for USART0 to 57600
    // For signal data rate: UBRR = 16
    // For double data rate: UBRR = 34
    UBRR0H = 0;
    UBRR0L = 16; // 34 for double data rate

    // Bits 7 - 2: Zero is fine for all
    // Bits 1: U2X double data rate: 0  (disabled based on UBRR)
    // Bits 0: Zero is fine
    UCSR0A = 0;

    // Bits 7 - 5: RXCIE TXCIE UDRIE:000 (disable UART interrupts)
    // Bits 4 - 3: RXEN TXEN : 11
    // Bits 2: UCSZ2: 0 (bit 2 for defining data size)
    // Bits 1-0: Zeros fine
    UCSR0B = 0b0001 1000; // 0x18

    // Bits 7: Zero is fine
    // Bits 6: UMSEL: 0 (Asynchronous Operation)
    // Bits 5-4: UPM1 UPM0: 10 (Even parity)
    // Bits 3: USBS: 1 (2 stop bits)
    // Bits 2-1: UCSZ1 UCSZ0: 11 (8 data bits)
    // Bits 0: Zero is fine
    UCSR0C = 0b0010 1110; // 0x2E
}
```
b. [7] Write the ADC_init() function to configure the ADC as follows:

- Interrupts disabled
- One-shot mode
- ADC clock rate in the range of 50K-200K Hz
- Reference voltage is 2.56v
- No differential input
- ADCW left adjusted

You may initialize unrelated bits as you wish.

```c
void ADC_init()
{
    //Bits 7-6: REFS1 REFS0: 11 (Vref = 2.56V)
    //Bits 5  : ADLAR : 1 (left adjusted)
    //Bits 4-0: 00000 (initialize to channel Zero)
    ADMUX = 0b1110 0000;  // 0xE0

    //Bits 7  : ADEN: 1 (Enable ADC)
    //Bits 6  : Zero fine
    //Bits 5  : ADFR: 0 (One-shot mode)
    //Bits 4  : Zero fine
    //Bits 3  : ADIE: 0 (Disable ADC interrupts)
    //Bits 2-0: ADPS2 ADPS1 ADPS0: = 111 (16MHz/128=125KHz)
    ADCSRA = 0b1000 0111; //0x87
}
```
c. [7] Write the poll_and_exec_cmd() function. You may assume that the channel input is always in the correct range, i.e. from 0 to 7 (Reminder this is a raw 8-bit encoding NOT ASCII).

```c
void poll_and_exe_cmd()
{
    unsigned char chan;  // Store channel

    // Wait for new data from USART0
    // Check UART Receive Complete (RXC) status register
    // RXC status register located at position 7
    while (!(UCSR0A & 0b1000 0000))
    {
    }
    chan = UDR0;  // Get the channel number

    // Set ADC channel
    ADMUX = (ADMUX & 0b11100000) | chan;

    // Start Conversion
    ADCSRA |= 0b01000000;  // ADSC is bit position 6

    // Wait for ADC conversion to complete
    while (ADCSRA & 0b01000000) // ADCS is bit pos 6
    {
    }

    // Wait for the USART0 to be ready for transmitting
    // Check for UART Data Transmit Reg (UDRE) empty
    // UDRE status register located at position 5
    while (! (USCR0A & 0b0010 0000))
    {
    }

    // Send the data out through USART0(8 most significant bits)
    UDR0 = ADCH; // Left justified so ADCH has top 8 bits
                  // ADC >> 8 also fine
}
```
Question 3: ADC Programming, Interrupt Based (20pts)

In this programming assignment, the ADC is connected to a thermal sensor by ADC channel 0. You are asked to program the ATMega128 such that the ADC runs in the free running mode and sends out every conversion result through USART0. You **MUST** use interrupt-based I/O programming style so that the CPU is not fully occupied by this task.

The ADC should be configured as follows:

- Interrupt enabled
- Free running mode
- ADC clock rate in the range of 50K-200K Hz
- Reference voltage is 2.56v
- No differential input
- ADCW left adjusted

The USART0 should be initialized as follows:

- Asynchronous communication
- Transmit only, **NO** receive
- 57,600 baud rate
- 8 data bits
- Even parity
- 2 stop bits

You may initialize unrelated control bits as you wish.

The ADC conversion results are 10-bit. For each result, send the most significant eight bits and discard the two least significant bits.
a. [10] Write the serial_init() and ADC_init() functions by revising your code for Question 2. Note: The ADC_init() function has to start the first ADC conversion.

```c
void serial_init(){
    // Set baud rate for USART0 to 57600
    // For signal data rate: UBRR = 16
    // For double data rate: UBRR = 34
    UBRR0H = 0;
    UBRR0L = 16;  // 34 for double data rate

    // Bits 7 - 2: Zero is fine for all
    // Bits 1: U2X double data rate: 0 (disabled based on UBRR)
    // Bits 0: Zero is fine
    UCSR0A = 0;

    // Bits 7 - 5: RXCIE TXCIE UDRIE:000 (disable UART interrupts)
    // Bits 4 - 3: RXEN TXEN : 01 (Disabled Receiving)
    // Bits 2: UCSZ2: 0 (bit 2 for defining data size)
    // Bits 1-0: Zeros fine
    UCSR0B = 0b0000 1000;  // 0x08

    // Bits 7: Zero is fine
    // Bits 6: UMSEL: 0 (Asynchronous Operation)
    // Bits 5-4: UPM1 UPM0: 10 (Even parity)
    // Bits 3: USBS: 1 (2 stop bits)
    // Bits 2-1: UCSZ1 UCSZ0: 11 (8 data bits)
    // Bits 0: Zero is fine
    UCSR0C = 0b0010 1110;  // 0x2E
}

void ADC_init(){
    //Bits 7-6: REFS1 REFS0: 11 (Vref = 2.56V)
    //Bits 5 : ADLAR : 1 (left adjusted)
    //Bits 4-0: 00000 (initialize to channel Zero)
    ADMUX = 0b1110 0000;  // 0xE0

    //Bits 7 : ADEN: 1 (Enable ADC)
    //Bits 6 : ADSC: 1 (Start Conversion)
    //Bits 5 : ADFR: 1 (Free running mode)
    //Bits 4 : Zero fine
    //Bits 3 : ADIE: 1 (Enable ADC interrupts)
    //Bits 2-0: ADPS2 ADPS1 ADPS0: = 111 (16MHz/128=125KHz)
    ADCSRA = 0b1110 1111; //0xEF
}
```
b. [10] Write the corresponding ISR. You may assume that USART0 is never overrun by the ADC, so when an ADC conversion is completed the USART0 is always ready for transmitting.

Hints: 1) Which ISR should you write? 2) You may find the right ISR vector code in the datasheet, and specific Vector names can be found here: http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

ISR (ADC_vect) // Fill the ISR vector code name
{
    UDR0 = ADCH; // Or ADC >> 8;
    // (Grading Note: -2pts if not shifted correctly)
}