Example 10.6

```
#define USART_TRANSMIT_BUFFER_SIZE   80

unsigned char g_transmitBuffer[USART_TRANSMIT_BUFFER_SIZE];
unsigned char g_transmitHead;
unsigned char g_transmitTail;
```

As in the receiver case, the `g_transmitBuffer` is reserved memory used to store the outbound data stream, and `g_transmitHead` and `g_transmitTail` are indices used to write/read to/from the buffer. The following steps are used to transmit a stream of bytes.

1. Write each new byte at `g_transmitHead` index, and then increment the head to the next byte in the buffer.
2. After incrementing `g_transmitHead`, check to make sure the value is not greater-than or equal-to the maximum value defined by the buffer array size. If it is, reset it back to 0.
3. Once the entire message is written to the buffer, turn on the transmitter by enabling the proper interrupt. Once the interrupt is enabled, the *TX* ISR will be called which will start the process of copying the transmit buffer bytes to the serial port transmitter.

These steps are implemented as in the following function.

Example 10.7

```
#define UCSR0B_ADDR                (unsigned char *) 0xC1

#define UCSR0B_TXEN_MASK           0x08
#define UCSR0B_TXEN_ON             0x08
#define UCSR0B_TXEN_OFF            0x00

void TransmitString (const char *bytes, unsigned char numberOfBytes)
{
   unsigned char *portUSARTControlAndStatusRegisterB;
   unsigned char shadow;
   unsigned char i;

   for (i = 0; i < numberOfBytes; i++)
   {
      g_transmitBuffer[g_transmitHead++] = (unsigned char) bytes[i];

      if (g_transmitHead >= USART_TRANSMIT_BUFFER_SIZE)
      {
         g_transmitHead = 0;
      }
   }
```

```
   /* Turn on the TX interrupt. */
   portUSARTControlAndStatusRegisterB = UCSR0B_ADDR;

   shadow = *portUSARTControlAndStatusRegisterB;
   shadow &= ~(UCSR0B_TXEN_MASK);
   shadow |= (UCSR0B_TXEN_ON);
   *portUSARTControlAndStatusRegisterB = shadow;
}
```

The final component to manage the serial port transmitter is the ISR. When the transmit interrupt occurs, a check is made to see if any more bytes exist in the buffer. If no more bytes are to be sent, the *TX* interrupt is turned off. However, if more bytes need to be transmitted, the ISR copies the next buffer byte into the transmit data register and the *TX* interrupt is reenabled. The AVR-GCC ISR could look something like the following.

Example 10.8

```
ISR(USART_UDRE_vect)
{
   unsigned char *portUSARTDataRegister;
   unsigned char *portUSARTControlAndStatusRegisterB;
   unsigned char shadow;

   portUSARTDataRegister = UDR0_ADDR;
   portUSARTControlAndStatusRegisterB = UCSR0B_ADDR;

   shadow = *portUSARTControlAndStatusRegisterB;
   shadow &= ~(UCSR0B_TXEN_MASK);

   if (g_transmitHead != g_transmitTail)
   {
      *portUSARTDataRegister = g_transmitBuffer[g_transmitTail++];

      if (g_transmitTail >= USART_TRANSMIT_BUFFER_SIZE)
      {
         g_transmitTail = 0;
      }

      shadow |= (UCSR0B_TXEN_ON);
   }
   else
   {
      shadow |= (UCSR0B_TXEN_OFF);
   }

   *portUSARTControlAndStatusRegisterB = shadow;
```

```
}
```

One important modification can be made to the code presented in order to make the ISR functions more efficient. Consider making the length of each buffer array a power of 2. Then managing the indices can be made quicker by using a bit-mask as in the following example.

Example 10.9

```
#define USART_TRANSMIT_BUFFER_SIZE   32
#define USART_TRANSMIT_BUFFER_MASK   0x1F

ISR(USART_UDRE_vect)
{
   if (g_transmitHead != g_transmitTail)
   {
      *portUSARTDataRegister = g_transmitBuffer[g_transmitTail++];

      g_transmitTail &= USART_TRANSMIT_BUFFER_MASK;
   }
}
```

This concept can be taken one step further if the microcontroller has enough memory by letting each buffer be 256 bytes in length. Then, because an `unsigned char` is and 8-bit entity, the bit-mask is unnecessary as in the following example.

Example 10.10

```
#define USART_TRANSMIT_BUFFER_SIZE   256

ISR(USART_UDRE_vect)
{
   if (g_transmitHead != g_transmitTail)
   {
      *portUSARTDataRegister = g_transmitBuffer[g_transmitTail++];
   }
}
```

Now, when the head or tail is sitting at 255 and increased by one, it will naturally wrap around to 0. Personally, I normally don't like this kind of code as it represents poor maintainability; that is, given to a software developer, they would not automatically recognize that the wrap around will occur. However, because we are dealing with ISRs, sometimes the low-level functionality is more important than the maintainability of the software. Certainly this would be a valid location for a nice comment stating the fact that the wrap around is intentional.

10.2 PERTINENT REGISTER DESCRIPTIONS

The information presented in this section was taken from ATMEL (2009).

10.2.1 TWBR - TWI BIT RATE REGISTER

Bit	7	6	5	4	3	2	1	0
0xB8	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- TWBR: TWI Bit Rate. TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes.

$$f_{\text{SCL}} = \frac{16\ \text{MHz}}{16 + 2(\text{TWBR})(\text{PrescalerValue})}$$

10.2.2 TWCR - TWI CONTROL REGISTER

Bit	7	6	5	4	3	2	1	0
0xBC	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W
Default	0	0	0	0	0	0	0	0

- TWEA: TWI Enable Acknowledge Bit. The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met.

  1. The device's own slave address has been received.
  2. A general call has been received, while the TWGCE bit in the TWAR is set.
  3. A data byte has been received in Master Receiver or Slave Receiver mode.

  By writing the TWEA bit to zero, the device can be virtually disconnected from the 2-wire Serial Bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- TWSTA: TWI START Condition Bit. The application writes the TWSTA bit to one when it desires to become a Master on the 2-wire Serial Bus. The TWI hardware checks if the bus is available and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected and then generates a new START condition to claim the bus Master status. TWSTA must be cleared by software when the START condition has been transmitted.

- TWSTO: TWI STOP Condition Bit. Writing the TWSTO bit to one in Master mode will generate a STOP condition on the 2-wire Serial Bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In Slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed Slave mode and releases the SCL and SDA lines to a high impedance state.

- TWWC: TWI Write Collision Flag. The TWWC bit is set when attempting to write to the TWI Data Register (TWDR) when TWINT is low. This flag is cleared by writing the TWDR Register when TWINT is high.

- TWEN: TWI Enable Bit. The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- See Ch. 9 for interrupt-related bit descriptions.

10.2.3 TWSR - TWI STATUS REGISTER

Bit	7	6	5	4	3	2	1	0
0xB9	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0
Read/Write	R	R	R	R	R	R	R/W	R/W
Default	1	1	1	1	1	0	0	0

- TWS7-3: TWI Status. These 5 bits reflect the status of the TWI logic and the 2-wire Serial Bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit pre-scaler value. The application designer should mask the pre-scaler bits to zero when checking the Status bits. This makes status checking independent of pre-scaler setting.

- TWPS1-0: TWI Pre-scaler Bits. These bits can be read and written, and control the bit rate pre-scaler.

10.2.4 TWDR - TWI DATA REGISTER

Bit	7	6	5	4	3	2	1	0
0xBB	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	1	1	1	1	1	1	1	1

Table 10.1: TWI Bit Rate Pre-scaler

TWPS1-0	Pre-scaler Value
00	1
01	4
10	16
11	64

- TWD7-0: TWI Data. In Transmit mode, `TWDR` contains the next byte to be transmitted. In Receive mode, the `TWDR` contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI Interrupt Flag (`TWINT`) is set by hardware. Note that the Data Register cannot be initialized by the user before the first interrupt occurs. The data in `TWDR` remains stable as long as `TWINT` is set. While data is shifted out, data on the bus is simultaneously shifted in. `TWDR` always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

10.2.5 TWAR - TWI SLAVE ADDRESS REGISTER

Bit	7	6	5	4	3	2	1	0
0xBA	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	1	1	1	1	1	1	1	0

- TWA6-0: TWI Slave Address. The `TWAR` should be loaded with the 7-bit Slave address to which the TWI will respond when programmed as a Slave Transmitter or Receiver, and not needed in the Master modes. In multi master systems, `TWAR` must be set in masters which can be addressed as Slaves by other Masters.

- TWGCE: TWI General Call Recognition Enable Bit. This bit is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

10.2.6 TWAMR - TWI SLAVE ADDRESS MASK REGISTER

Bit	7	6	5	4	3	2	1	0
0xBD	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	-
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
Default	0	0	0	0	0	0	0	0

- TWAM6-0: TWI Address Mask. The TWAMR can be loaded with a 7-bit Slave Address mask. Each of the bits in TWAMR can mask (disable) the corresponding address bits in the TWI Address Register (TWAR). If the mask bit is set to one, then the address match logic ignores the comparison between the incoming address bit and the corresponding bit in TWAR.

10.2.7 SPCR - SPI CONTROL REGISTER

Bit	7	6	5	4	3	2	1	0
0x4C	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- SPE: SPI Enable. When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.
- DORD: Data Order. When the DORD bit is written to one, the LSB of the data word is transmitted first. When the DORD bit is written to zero, the MSB of the data word is transmitted first.
- MSTR: Master/Slave Select. This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.
- CPOL: Clock Polarity. When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle.
- CPHA: Clock Phase. The settings CPHA determine if data is sampled on the leading (first) or trailing (last) edge of SCK. When CPHA is written to zero, data is sampled on the leading edge. When CPHA is written to one, data is sampled on the trailing edge.
- SPR1-0: SPI Clock Rate Select Bits. These two bits control the SCK rate of the device configured as a Master. SPR1:0 have no effect on the Slave. The relationship between SCK and the Oscillator Clock frequency f_{osc} is shown in Table 10.2.
- See Ch. 9 for interrupt-related bit descriptions.

Table 10.2: Relationship Between SCK and f_{osc}.

SPI2X	SPR1-0	SCK Frequency
0	00	$f_{osc}/4$
0	01	$f_{osc}/16$
0	10	$f_{osc}/64$
0	11	$f_{osc}/128$
1	00	$f_{osc}/2$
1	01	$f_{osc}/8$
1	10	$f_{osc}/32$
1	11	$f_{osc}/64$

10.2.8 SPSR - SPI STATUS REGISTER

Bit	7	6	5	4	3	2	1	0
0x4D	SPIF	WCOL	-	-	-	-	-	SPI2X
Read/Write	R	R	R	R	R	R	R	R/W
Default	0	0	0	0	0	0	0	0

- WCOL: Write COLlision Flag. The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit is cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- SPI2X: Double SPI Speed Bit. When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode. This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at $f_{osc}/4$ or lower.

- See Ch. 9 for interrupt-related bit descriptions.

10.2.9 SPDR - SPI DATA REGISTER

Bit	7	6	5	4	3	2	1	0
0x4E	SPD7	SPD6	SPD5	SPD4	SPD3	SPD2	SPD1	SPD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	-	-	-	-	-	-	-	-

- SPD7-0: SPI Data. The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

10.2.10 UDR0 - USART0 I/O DATA REGISTER

Bit	7	6	5	4	3	2	1	0
0xC6	UD7	UD6	UD5	UD4	UD3	UD2	UD1	UD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- UD7-0: USART0 Data. The USART0 Transmit Data Buffer Register and USART0 Receive Data Buffer Registers share the same I/O address referred to as USART0 Data Register or UDR0. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR0 Register location. Reading the UDR0 Register location will return the contents of the Receive Data Buffer Register (RXB).

  For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

  The transmit buffer can only be written when the UDRE0 Flag in the UCSR0A Register is set. Data written to UDR0 when the UDRE0 Flag is not set, will be ignored by the USART0 Transmitter. When data is written to the transmit buffer and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD0 pin.

  The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS) since these also will change the state of the FIFO.

10.2.11 UCSR0A - USART0 CONTROL AND STATUS REGISTER A

Bit	7	6	5	4	3	2	1	0
0xC0	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
Read/Write	R	R/W	R	R	R	R	R/W	R/W
Default	0	0	1	0	0	0	0	0

- FE0: Frame Error. This bit is set if the next character in the receive buffer had a Frame Error when received, i.e., when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDR0) is read. The FE0 bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSR0A.

- DOR0: Data OverRun. This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDR0) is read. Always set this bit to zero when writing to UCSR0A.

- UPE0: USART0 Parity Error. This bit is set if the next character in the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (`UPM01 = 1`). This bit is valid until the receive buffer (`UDR0`) is read. Always set this bit to zero when writing to `UCSR0A`.

- U2X0: Double the USART0 Transmission Speed. This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

  Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication.

- MPCM0: Multi-processor Communication Mode. This bit enables the Multi-processor Communication mode. When the `MPCM0` bit is written to one, all the incoming frames received by the USART0 Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the `MPCM0` setting.

- See Ch. 9 for interrupt-related bit descriptions.

10.2.12 UCSR0B - USART0 CONTROL AND STATUS REGISTER B

Bit	7	6	5	4	3	2	1	0
0xC1	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Default	0	0	0	0	0	0	0	0

- RXEN0: USART0 Receiver Enable. Writing this bit to one enables the USART0 Receiver. The Receiver will override normal port operation for the `RxD0` pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the `FE0`, `DOR0`, and `UPE0` Flags.

- TXEN0: USART0 Transmitter Enable. Writing this bit to one enables the USART0 Transmitter. The Transmitter will override normal port operation for the `TxD0` pin when enabled. Disabling the Transmitter will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the `TxD0` port.

- UCSZ02: USART0 Character Size. The `UCSZ02` bit combined with the `UCSZ01:0` bits in `UCSR0C` sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

- RXB80: USART0 Receive Data Bit 8. `RXB80` is the ninth data bit of the received character when operating with serial frames with nine data bits. It must be read before reading the low bits from `UDR0`.

- TXB80: USART0 Transmit Data Bit 8. TXB80 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. It must be written before writing the low bits to UDR0.

- See Ch. 9 for interrupt-related bit descriptions.

10.2.13 UCSR0C - USART0 CONTROL AND STATUS REGISTER C

Bit	7	6	5	4	3	2	1	0
0xC2	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	1	1	0

- UMSEL01-0: USART0 Mode Select Bits. These bits select the mode of operation of the USART0 as shown in Table 10.3.

Table 10.3: UMSEL0 Bit Settings.

UMSEL01-0	Mode
00	Asynchronous USART
01	Synchronous USART
10	Reserved
11	Master SPI (MSPIM)

- UPM01-0: USART0 Parity Mode. These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPM0 setting. If a mismatch is detected, the UPE0 Flag in UCSR0A will be set.

Table 10.4: UPM0 Bit Settings.

UPM01-0	Parity Mode
00	Disabled
01	Reserved
10	Enabled, Even Parity
11	Enabled, Odd Parity

- USBS0: USART0 Stop Bit Select. This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting. If USBS0 is cleared, there is 1 stop bit. If USBS0 is set, there are 2 stop bits.

- UCSZ01-0: USART0 Character Size. The UCSZ01:0 bits combined with the UCSZ02 bit in UCSR0B sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

Table 10.5: UCSZ0 Bit Settings.

UCSZ02-0	Character Size
000	5-bit
001	6-bit
010	7-bit
011	8-bit
100	Reserved
101	Reserved
110	Reserved
111	9-bit

- UCPOL0: USART0 Clock Polarity. This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOL0 bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK0).

Table 10.6: UCPOL0 Bit Settings.

UCPOL0	Transmitted Data Changed	Received Data Sampled
0	Rising XCK0 Edge	Falling XCK0 Edge
1	Falling XCK0 Edge	Rising XCK0 Edge

10.2.14 UBRR0H AND UBRR0L - USART0 BAUD RATE REGISTERS

Bit	7	6	5	4	3	2	1	0
0xC5	-	-	-	-	UBRR0[11:8]			
0xC4	UBRR0[7:0]							
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- UBRR011-0: USART0 Baud Rate Bits. This is a 12-bit register which contains the USART0 baud rate. The UBRR0H contains the four most significant bits, and the UBRR0L contains the eight least significant bits of the USART0 baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing UBRR0L will trigger an immediate update of the baud rate pre-scaler.

Table 10.7: UBRR0 Bit Settings for 16 MHz System Clock.

Baud Rate (bps)	U2X0 = 0, UBRR0	U2X0 = 1, UBRR0
2400	416	832
4800	207	416
9600	103	207
14400	68	138
19200	51	103
28800	34	68
38400	25	51
57600	16	34
76800	12	25
115200	8	16
230400	3	8
250000	3	7
500000	1	3
1000000	0	1

PROBLEMS

10.1 Determine the appropriate bit settings for `UCSR0A`, `UCSR0B`, `UCSR0C`,and `UBRR0` to manage a serial interface using the following specific details:

- use normal transmission speed (i.e., disable the x2 speed),
- disable the multi-processor communication mode,
- turn on the *RX* complete interrupt, turn off the *TX* complete interrupt, turn on the data register empty interrupt,
- turn on the receiver, turn off the transmitter,
- set the character size to 8 bits,
- use the asynchronous USART mode,
- use no parity,
- use 1 stop bit,
- set the baud rate to 115200 bits per second.

10.2 Create a function that initializes the USART based on the values determined in problem 10.1.

10.3 Create an ISR for both the *RX* and *TX* interrupts.

10.4 Create a program that utilizes the USART to send and receive bytes via the USB converter chip to and from the host computer terminal program. Use a switch statement to send a