SERIAL COMMUNICATIONS WITH AVR MICROCONTROLLER

■ ARUN DAYAL UDAI

Serial communication is a low-level communication between two or more devices. Normally, one of the devices is a computer, while the other device can be a modem, printer, another computer, or a scientific instrument such as an oscilloscope or a function generator. It may even be a device to device communication like that between a microcontroller and a sensor, LCD, storage memory or a different microcontroller.

In serial communication, you can send and receive data serially, one bit at a time. In a microcontroller, it can be realised in two modes of serial communication—synchronous and asynchronous.

The article covers the concept in three broad sections—synchronous serial communication, asynchronous serial communication and serial port interfacing using PC through GUI developed in Visual Basic.Net

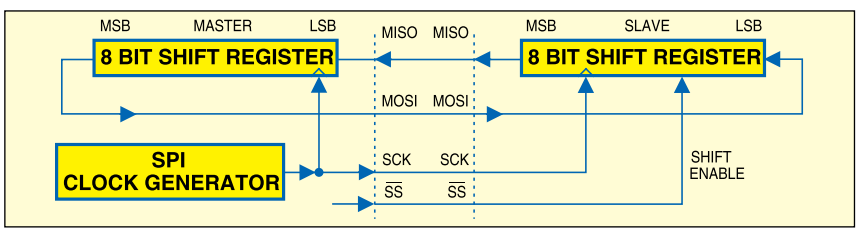

Fig. 1: SPI master-slave interconnection

Synchronous serial communication

Serial peripheral interface (SPI) is a synchronous serial data communication that includes a three-wire transmission mode, with master out slave in (MOSI), master in slave out (MISO) and synchronising clock (SCK) pins. Ground (GND) acts as a common reference and slave select ($\overline{SS}$) selects the slave which is to be driven by the master, by setting it low. $\overline{SS}$ pin may be permanently set to low if only one slave is being used. Many slaves are connected in parallel and the master selects one out of many by setting the slave's $\overline{SS}$ pin to low. MOSI and MISO forms a cyclic shift register which is moved bit-wise with the synchronising

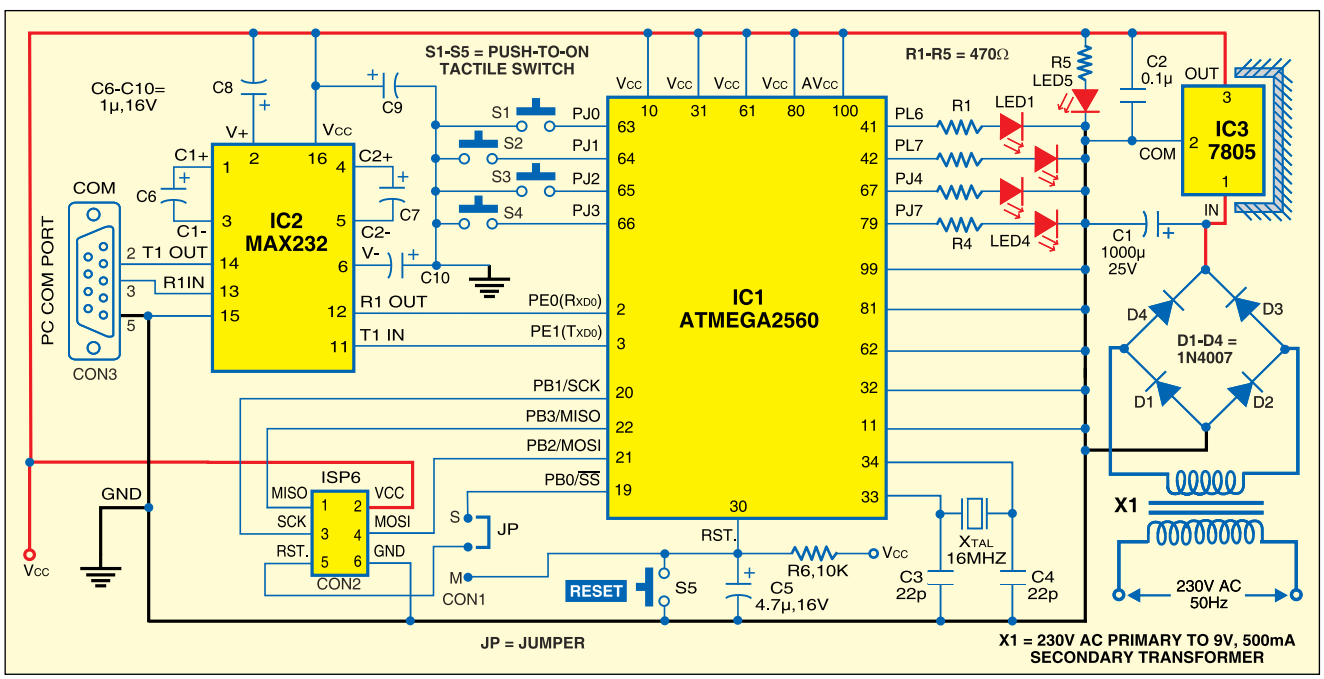

Fig. 2: Circuit of synchronous serial communication with AVR microcontroller

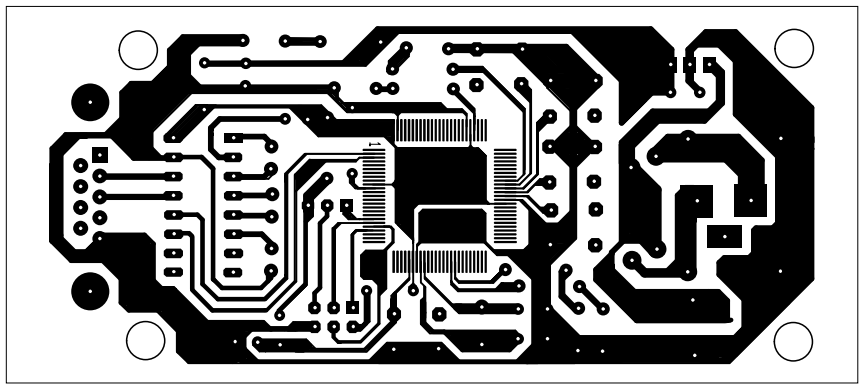

Fig. 3: An actual-size, single-side PCB for the serial communications with AVR microcontroller

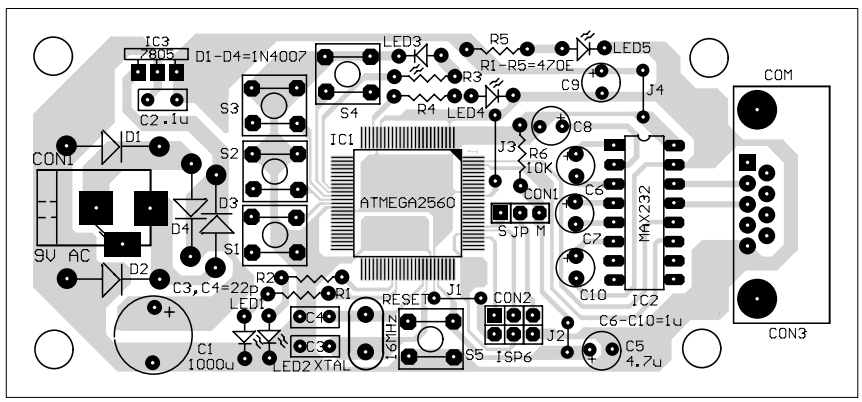

Fig. 4: Component layout for the PCB

clock signal generated by the master. Fig. 1 shows SPI master-slave interconnection.

Once the 8-bit data transfer is complete, SPI interrupt flag (SPIF) is set high in SPI status register (SPSR) and the data is available to be read in SPI data register (SPDR) of the slave. Similarly, when the data is copied to SPDR in master, it automatically starts its synchronising clock and data starts transferring cyclically through its shift register to the slave. As soon as the 8-bit data is transferred, SPIF flag in SPSR is set.

To initialise a device as master, the MOSI, SCK and SS pins are set as output by setting the corresponding bits high in data direction register (DDRx). In SPI control register (SPCR), SPI enable (SPE), SPI clock rate (SPR0-SPR1) bits are set high. Clock rate by the master should be such that low and high periods are longer than two CPU clock cycles for the slave. Master additionally drives the slave's $\overline{SS}$ pin low. To initialise a device as slave, the MISO pin in DDRx and SPE pin in SPCR are set high.

Circuit description

Fig. 2 shows the synchronous serial communication with AVR microcontroller. At the heart of the circuit is ATmega2560 (IC1), which is a low-power CMOS 8-bit microcontroller, based on the AVR enhanced RISC architecture. It features 128kB Flash memory with read-while-write capabilities, 4kB EEPROM, 8kB SRAM, 86 general-purpose input/output lines, 32 general-purpose working registers, six flexible timers/counters with compare modes, real time counter (RTC), 10-bit analogue-to-digital converter, internal and external interrupts, a serial programmable USART, a programmable watchdog timer with internal oscillator and an SPI serial port.

Let's assume that IC1 (ATmega2560) in Fig. 2 is a master device. We need another circuit same as Fig. 2 for slave device as well. Communications between master and slave devices are done through ISP6 connector as explained below.

Switches S1 through S4 are interfaced with port PJ0 though PJ3 of microcontroller IC1 respectively. Switches S1 though S4 are used as input control switches to turn off the LEDs (LED1 through LED4) connected in the slave microcontroller. LED1 through LED4 act as visual display for the device. LED1 through LED4 are interfaced with port pins PL6, PL7, PJ4 and PJ7, respectively. Switch S5 is used for manual reset. Port pins PB2 (MOSI), PB3 (MISO) and PB1 (SCK) are used for in-system programming (ISP). IC MAX232 (IC2) is used for serial communication. T_{XD} and R_{XD} pins of the microcontroller too are used for serial communication with the help of COM port. System clock plays a significant role in the operation of the microcontroller. A 16MHz quartz crystal provides basic clock to the microcontroller (IC1) at its pins 33 and 34.

The 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 9V, 500mA. The transformer output is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by IC 7805 (IC3). Capacitor C2 bypasses the ripples present in regulated supply. LED5 acts as the power indicator and resistor R5 limits the current through LED5.

An actual-size, single-side PCB for the circuit of synchronous serial communication with AVR microcontroller is shown in Fig. 3 and its component layout in Fig. 4. Assemble the circuit on a PCB to minimise time and assembly errors. Carefully assemble the components and double-check for any overlooked error. Assemble two boards—one for master and another for slave. Burn the hex codes of the C source codes listed at the end of this

PARTS LIST

Semiconductors:

IC1	- ATmega2560 microcontroller
IC2	- MAX232 RS-232 driver
IC3	- 7805, 5V regulator
D1-D4	- 1N4007 rectifier diode
LED1-LED5	- 5mm LED

Resistors (all ¼-watt, ±5% carbon)

R1-R5	- 470-ohm
R6	- 10-kilo-ohm

Capacitors:

C1	- 1000µF, 25V electrolytic
C2	- 0.1µF ceramic disk
C3, C4	- 22pF ceramic disk
C5	- 4.7µF, 16V electrolytic
C6-C10	- 1µF, 16V electrolytic

Miscellaneous:

X1	- 230V AC primary to 9V, 500mA secondary transformer
X_{TAL}	- 16MHz crystal
S1-S5	- Push-to-on tactile switch
CON3	- 9-pin D-type female connector
CON2	- 6-pin ISP male connector
CON1	- 3-pin berg strip male connector

article into the respective master and slave microcontrollers. The source codes of master and slave demonstrate SPI communication between two microcontrollers (ATmega2560) fitted in two different boards, connected through a six-pin header with pins MOSI, MISO, SCK, RST, VCC and GND. Note that jumper (JP) is not used in synchronous communication between two devices in this example. JP is used only during programming. Switches S1 through S4 on master control LED1 through LED4 of slave board respectively. This represents the communication between two microcontrollers. The six-pin header is also used to program the microcontroller using serial programmer.

Software

The software for microcontroller of master and slave are written in 'C' using the IAR Embeded Workbench integrated development environment. IAR Embedded Workbench is being developed by IAR Systems and ATMEL developers in parallel and hence it generates the optimised code which uses full 'C' coding capabilities of AVR devices. AVR development tools for embedded systems can be downloaded free from IAR website www.iar.com. For details of IAR Embedded Workbench, you may refer to 'A Beginners' Guide to ATMEL AVR Development' article published in January issue of EFY.

Communication by master. The step-by-step process is as follows:

1. Initialise master

(i) Set DDR for MOSI, SCK and $\overline{SS}$ pin as output.

(ii) Enable SPI as master by setting SPE and master/slave select (MSTR) bit in SPCR and set clock rate.

2. Master transmits data

(i) Write data to SPDR

(ii) Wait while SPIF is set in SPSR

(iii) May optionally read the data back from slave device

Communication by slave. The step-by-step process is as follows:

1. Initialise slave

(i) Set DDR for MISO pin as output

(ii) Enable SPI by setting SPE bit in SPCR

(iii) $\overline{SS}$ pin should be driven low by the master

2. Slave receives data

(i) Wait while SPIF is set in SPSR

(ii) Read SPDR

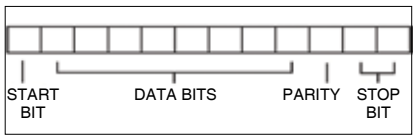

Fig. 5: Asynchronous serial communications sequence

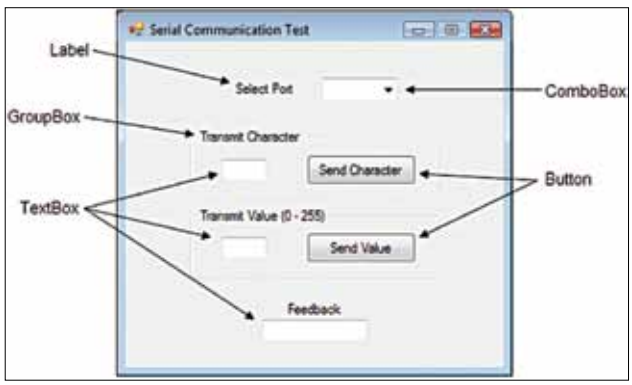

Fig. 6: GUI for serial communication using .Net

Asynchronous serial communications

Universal synchronous and asynchronous receive and transmit (USART) supported by AVR devices is normally an example of asynchronous serial communication, which can also be used for synchronous serial communications. Over the years, several standards have been set for asynchronous serial communications, of which RS-232 standard is most widely used and supported by PC and microcontrollers. It eliminates the clock signal wire and there is no cyclic register. So, we are left with only two wires, excluding the common GND wire. MOSI and MISO are represented by T_X and R_X, respectively. R_X and T_X for master and slave devices are cross-connected and each may drive the other one. That is, the T_X pin of master device is connected to R_X pin of slave device and R_X pin of master is connected to T_X pin of the slave device. Each connection sends and receives the data independently. The data-transfer speed is predetermined by a baud rate. The serial data is formatted sequentially as start bit, normally eight data bits—one parity bit and one or two stop bits (refer Fig. 5). The order of transmission is:

1. Each data transmission starts with start bit which is always a zero (0)

2. The data bits are transmitted. The first data bit corresponds to the least significant bit (LSB), while the last bit corresponds to the most significant bit (MSB)

3. The parity bit (if defined) is transmitted

4. Each data transmission ends with stop bit which always has logic level one (1)

To initialise a serial port, we need to activate the functions of the pins R_{XD0} and T_{XD0} for microcontroller ATmega2560. By default, they are enabled in asyn-

Fig. 7: HyperTerminal windows for name and icon

Fig. 8: HyperTerminal windows for COM port

chronous mode. To do so, we need to set bits—recieve enable (RXEN) and transmit enable (TXEN)—in USART control and status register B (UCSRB), with a 16-bit value of baud rate stored in USART baud rate register UBRRH and UBRRL (UBRR). Refer datasheet for UBRR values corresponding to a baud rate for a particular clock frequency. Frame format is set to 8-bit mode by setting USART register size—UCSZ0 and UCSZ1—in USART control and status register C (UCSRC), which is also the default value when we initialise USART. Stop bit is set as one and parity bit is disabled by default. Mode select (URSEL) bit should be high while writing to UCSRC. This is one of the most commonly used frame formats.

To transmit a data through serial pin T_{XD0}, data is written to USART data register (UDR) when transmit buffer is empty, which is indicated by Data Register Empty Flag (UDRE) set in USART

Fig. 9: HyperTerminal windows for selection of buad rate, data bits, parity, stop bits flow control

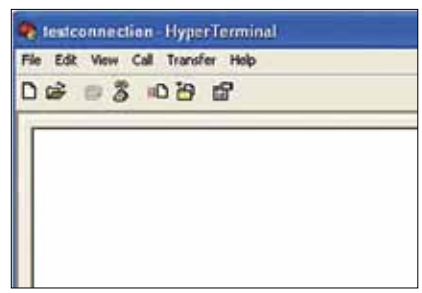

Fig. 10: Sent information on HyperTerminal

control and status register A (UCSRA). Similarly, to receive a data at serial pin R_{XD0}, UDR is read when data is received, which is indicated by receive complete RXC flag set in UCSRA.

USART communication. The step-by-step process is as follows:

1. Initialisation

(i) Put UBRR value in UBRR register based on 12-bit baud rate value in UBRRH then UBRRL

(ii) Set RXEN and TXEN in UCSRB

(iii) Set URSEL, frame format 8-bit by UCSZ1 and UCSZ0

2. USART transmits

(i) Wait while Transmit Buffer Empty, UDRE Set in UCSRA

(ii) Put data into UDR

3. USART receives

(i) Wait while data to be received, RXC set in UCSRA

(ii) Read value in UDR

The steps explained above are demonstrated by asynchronous communication as explained below. The microcontroller code (USART.C) just adds one to the value that it receives and sends it back to the PC.

Interfacing serial port in Visual Basic.Net. With Windows Vista, HyperTerminal is no longer a part of Windows like Vista. So, we are left with programming environments only to communicate with our devices like microcontrollers. Fig. 6 shows serial communication graphical user interface (GUI) window for handling and monitoring data. This GUI for serial communicaton is developed using Visual Basic.Net.

The interface demonstrates handling of ASCII text and numeric values through a PC serial port. As per our microcontroller source code (USART.C), it will send back 'B' if you send 'A' and so on. That is, if you enter 'A' under 'Transmit Character' textbox followed by clicking 'Send Character' button, you will see 'B' under 'Feedback' textbox.

Asynchronous serial communication through PC HyperTerminal. Very often we require testing our external embedded devices using serial port. Information is sent to PC through COM port using HyperTerminal program.

To open the HyperTerminal program, go to Start→Programs→Accessories→Communications→HyperTerminal. You will see windows as shown in Fig. 7 for name and icon.

Type in the desired name and click 'OK.' Select COM port (refer Fig. 8) and click 'OK.'

Select the buad rate as '115200,' data bit as '8,' parity as 'None,' and click 'OK' (refer Fig. 9).

Now click 'Transfer' menu bar and send a text file (refer Fig. 10). If the board receives character 'A' it will return 'B' on HyperTerminal. If you press any character, number or symbol then the succeeding character, number or symbol will return on HyperTerminal.

EFY note. The source codes of this article are available on www.efymag.com website.

The author is assistant professor in Department of Mechanical Engineering at Birla Institute of Technology, Mesra

MASTER.C

```
/*************************************
  Code(Master) for SPI Communication be-
tween two ATMEGA2560 Board
*************************************/

#include <ioavr.h>
#include <intrinsics.h>

#define LED1 PORTL6    //Connected to PORTL
#define LED2 PORTL7    //Connected to PORTL
#define LED3 PORTJ4    //Connected to PORTJ
#define LED4 PORTJ7    //Connected to PORTJ

#define SW1 PINJ0      //All Switches are
connected to PORTJ
#define SW2 PINJ1
#define SW3 PINJ2
#define SW4 PINJ3

#define MOSI PB2
#define SCK  PB1
#define SS   PB0

#define SETBIT(ADDRESS, BIT)      (ADDRESS
|= (1 << BIT))
#define CLEARBIT(ADDRESS, BIT)    (ADDRESS
&= ~(1 << BIT))
#define CHECKBIT(ADDRESS, BIT)  (!(ADDRESS
& (1 << BIT)))

//SPI Subroutines
//Function to Initialize SPI as Master
void SPI_MasterInit(void)
{
  //Set MOSI, SCK and SS output, all oth-
ers input
  DDRB = (1<<MOSI)|(1<<SCK)|(1<<SS);
   //Enable SPI, Master, set clock rate
fosc/16 - 115.2 kHz SPI with 3.686 Mhz Osc
  SPCR    = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
  CLEARBIT(PORTB, SS);  //Select the slave
device
}

//Function to send byte through SPI Inter-
face, ignoring the returning byte
void SPI_SendByte(unsigned char byte)
{
  SPDR = byte;
   while(!(SPSR &  (1<<SPIF)));         //
Wait for transmission complete
}

//Assuming Internal Clock of 3.686 MHz

__C_task void main(void)
{
          //Set all LEDs pins as Output
        SETBIT(DDRL, LED1);
        SETBIT(DDRL, LED2);
        SETBIT(DDRJ, LED3);
        SETBIT(DDRJ, LED4);

       //Enable Pull-ups on PORTJ Switches
         PORTJ |= (1<<PJ0)|(1<<PJ1)|(1<<P
J2)|(1<<PJ3);

        SPI_MasterInit();
          while(1)
          {
                  if  (CHECKBIT(PINJ,
SW1))
                {
                  SPI_SendByte('A');
                  SETBIT(PORTL, LED1);
                }
                else
                  CLEARBIT(PORTL, LED1);

                if (CHECKBIT(PINJ, SW2))
                {
                  SPI_SendByte('B');
                  SETBIT(PORTL, LED2);
                }
                else
                  CLEARBIT(PORTL, LED2);

                if (CHECKBIT(PINJ, SW3))
                {
                  SPI_SendByte('C');
                  SETBIT(PORTJ, LED3);
                }
                else
                  CLEARBIT(PORTJ, LED3);

                if (CHECKBIT(PINJ, SW4))
                {
                  SPI_SendByte('D');
                  SETBIT(PORTJ, LED4);
                }
                else
                  CLEARBIT(PORTJ, LED4);
          }
}
```

SLAVE.C

```
/*************************************
   Code(Slave) for SPI Communication be-
tween two ATMEGA2560 Board
*************************************/

#include <ioavr.h>
#include <intrinsics.h>

#define LED1 PORTL6    //Connected to PORTL
#define LED2 PORTL7    //Connected to PORTL
#define LED3 PORTJ4    //Connected to PORTJ
#define LED4 PORTJ7    //Connected to PORTJ

#define MISO PB3

#define SETBIT(ADDRESS, BIT)      (ADDRESS
|= (1 << BIT))
#define CLEARBIT(ADDRESS, BIT)    (ADDRESS
&= ~(1 << BIT))

//Function to Initialize SPI as Slave
void SPI_SlaveInit(void)
{
/* Set MISO output, all others input */
DDRB = (1<<MISO);
/* Enable SPI */
SPCR = (1<<SPE);
}

//Function to receive byte from the master
SPI device
unsigned char SPI_SlaveReceive(void)
{
/* Wait for reception complete */
while(!(SPSR & (1<<SPIF)));
/* Return data register */
return SPDR;
}

//Assuming External Clock of 16 MHz
__C_task void main(void)
{
          //Set all LEDs pins as Output
        SETBIT(DDRL, LED1);
        SETBIT(DDRL, LED2);
        SETBIT(DDRJ, LED3);
        SETBIT(DDRJ, LED4);

         unsigned char k;
         SPI_SlaveInit();
          while(1)
          {
          k = SPI_SlaveReceive();
          if (k == 'A')
            SETBIT(PORTL, LED1);
          else
            CLEARBIT(PORTL, LED1);

         if (k == 'B')
            SETBIT(PORTL, LED2);
         else
            CLEARBIT(PORTL, LED2);

         if (k == 'C')
            SETBIT(PORTJ, LED3);
         else
            CLEARBIT(PORTJ, LED3);

         if (k == 'D')
            SETBIT(PORTJ, LED4);
         else
            CLEARBIT(PORTJ, LED4);
        }
}
```

USART.C

```
/*
   Serial Communication Check Source for
ATmega2560 Board
*/

#include <ioavr.h>

#define SETBIT(ADDRESS, BIT)      (ADDRESS
|= (1 << BIT))
#define CLEARBIT(ADDRESS, BIT)    (ADDRESS
&= ~(1 << BIT))
#define CHECKBIT(ADDRESS, BIT)  (ADDRESS &
(1 << BIT))

#define LED1 PORTL6       //LED Connected
to PORTL

/* Prototypes */
void USART0_Init (unsigned int baud);
void USART0_Transmit (unsigned char data);
unsigned char USART0_Receive (void);

__C_task void main (void)
{
  /* Set the baudrate to 115.2 kbps using
a 16.0 MHz crystal
  Fuse bits set to handle CPU with exter-
nal Crystal Oscillator */
  unsigned char a = 0;
  USART0_Init (8);
   SETBIT(DDRL, PL6);   //Set LED pin as
output
  while (1)
    {
      a = USART0_Receive();
      if (a == '0') CLEARBIT(PORTL, LED1);
      if (a == '1') SETBIT(PORTL, LED1);
      USART0_Transmit(a + 1);
    }
}

//Initialize USART0
void USART0_Init(unsigned int baud)
{
  /* Set baud rate */
  UBRR0H = (unsigned char)(baud >> 8);
  UBRR0L = (unsigned char)(baud);
  /* Enable receiver and transmitter */
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);
   /* Set frame format: Asynchronous 8
data, 1 stop bit */
  UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
}

//Read and Write functions for USART0
void USART0_Transmit (unsigned char data)
{
  /* Wait for empty transmit buffer */
  while (!CHECKBIT(UCSR0A, UDRE0));
  /* Put data into buffer, transmits the
data */
  UDR0 = data;
}

unsigned char USART0_Receive (void)
{
  /* Wait for data to be received */
  while (!CHECKBIT(UCSR0A, RXC0));
   /* Get and return received data from
buffer */
  return UDR0;
}
```