AVR380: 8-bit Microcontroller, Simple Serial Mouse Controller

Features

- **Very Low Cost Solution**
- **Low Power Consumption**
- **Few External Components**
- **Efficient Code**
- **Interrupt Driven**
- **Uses the Economical AT90S1200**
- **Self-Powered from Serial Port**

Introduction

This application note describes a simple controller for a serial PC mouse. It takes inputs from two pairs of quadrature encoded infra-red sensors and up to three buttons, converting movement and switch data into serial data for PC control. The mouse uses the standard RS-232 port of the host PC sending movement and switch data in “Mouse Systems” format, and takes power from a signal line.

The application can be used as the basis for a more powerful mouse controller with other protocols.

Theory of Operation

The mouse has become the major input device for PCs, and various types can be found in the market place. The mouse may have its own dedicated port (Bus mouse), or more commonly connect to one of the serial ports (usually COM1). The most common type of mouse encountered plugs into the 9-pin or 25-pin D type serial port and derives its power from the same port via signal lines.

A serial mouse is made from several functional blocks:

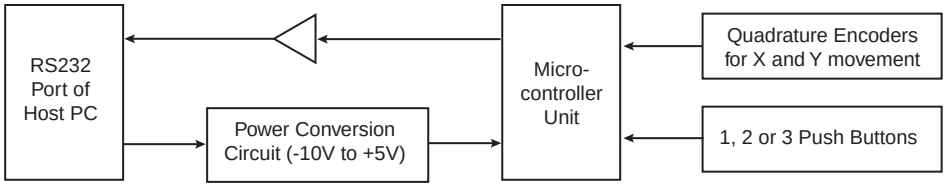
- Microcontroller Unit (MCU)
- Button detection
- Motion detection
- RS-232 signal generation
- 5v, or lower, DC supply circuit

A typical block diagram is shown in Figure 1.

Basic Principle of Operation

The serial port has no direct power connections available, consequently the first job is to derive these from a signal line. This is usually achieved by keeping the Request To Sent (RTS) line at a logic '1' (-10v or so) and converting this to the requisite positive supply rail by 'chopping' techniques.

Figure 1. Functional Blocks of Seral Mouse



Product Description

Application Note

Rev. xxxxA–02/99

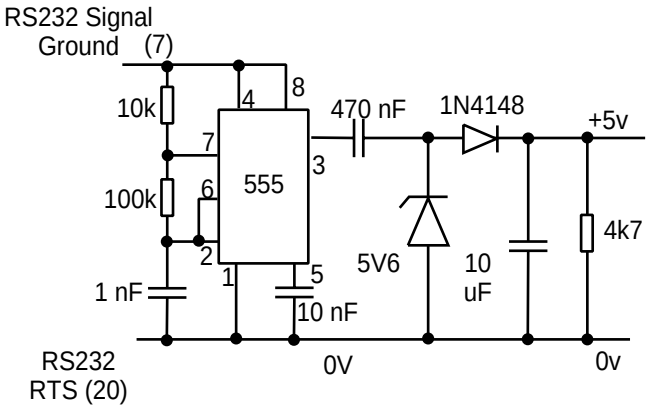
The current consumption must be kept low, typically 10 mA maximum. This places severe constraints on the electronics, requiring very low power consumption components. The AVR is ideal for this application being capable of high speed at low power.

The communication requirements for a simple serial mouse are only one way (Simplex) so only the Received Data (RXD) line is needed along with the Signal Ground.

The Baud rate is normally kept low at 1200 baud. The logic signal from the MCU to the RXD line must first be converted to RS-232 levels, i.e. 0v ('0') goes to approximately +10v (7 to 15v) and +5v ('1') goes to approximately -10v (-7 to -15v). This usually employs a commercial chip such as the MAX232 which only requires a +5v supply and a few capacitors to do the conversion, or better still, the MAX233 which requires no capacitors. These provide for two channels of send and receive so the chip is underused and relatively expensive. Another method can be seen employed in appication note AVR910 and uses bipolar transistors. The choice is largely one of economics and/or board space.

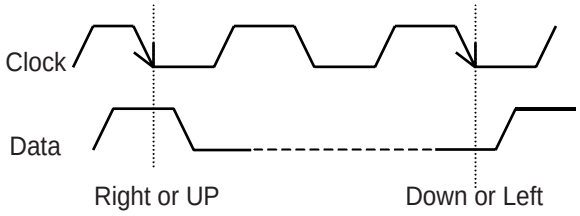
The power conversion circuit can be done with a simple (and cheap) 555 timer circuit and a few discrete components to keep the cost down. A typical circuit for this is shown in Figure 2. An alternative would be to use one of the commercial chips designed for the purpose, again the choice is one of economics.

Figure 2. Circuit Diagram of Typical Voltage Converter



The microcontroller accepts signals from the two pairs of infra-red sensors for each axis, arranged about a small slotted wheel. Each axis (X and Y) provides two signals in quadrature, commonly called CLOCK and DATA, enabling both direction and movement to be calculated by the microcontroller. Figure 3 shows how these quadrature waveforms provide the necessary information.

Figure 3. Clock and Data Quadrature Signals



The movement is calculated using the CLOCK signal to increment (or decrement) a counter and the direction is deduced from the state of the DATA line.

If we assume the CLOCK is sensed on the falling edges then we can see that if DATA is high the movement is positive (UP or RIGHT) and if low negative (DOWN or LEFT).

Sensing of each of these X or Y signals is achieved in one of two fashions :

Interrupts

Each CLOCK signal generates an interrupt which looks at the DATA line and computes the direction and amends the counter accordingly.

Polling

The CLOCK signals are continously scanned and when a change is noticed the appropriate action is taken.

The choice of which method to use is a matter of whether the interrupts are available. Using interrupts usually makes the program easier and shorter.

The mouse buttons (one, two or three) are usually simple 'micro-switch' types which pull-up a signal line normally held low by a pull-down resistor. The component count is reduced in this application by reversing the polarity and using the self contained pull-up resistors provided by the AVR port circuits for this purpose. (I.e. The switch pulls the signal line low.)

The mouse buttons are polled in the usual fashion since speed is not an important factor here. The button press response has to be slowed anyway to avoid 'bounce' problems.

This application uses the external interrupt line PD2 for X CLOCK and the analog comparator input PB1 for Y CLOCK, with PB0 taken to a half supply point, fixed by a potential divider. This trick provides the necessary two external interrupt inputs, since the AT90S1200 only has one external interrupt line. Both external interrupt and analog comparator inputs are set for falling edge trigger.

Having computed the movement and switch data it then has to be sent to the host system in RS-232 Serial format. The most common format, consequently used in this application, is 1200 Baud, eight data bits, no parity and two stop

bits. The data is sent according to an agreed protocol. Two common protocols are 'Mouse System' and 'Microsoft Format'. These are illustrated in Table 1 and Table 2.

L, M, R refer to the Left, Middle and Right buttons respectively and are active high, i.e. '1' = pressed and '0' = released. X7-0 and Y7-0 are the X and Y movement data.

Table 1. Mouse System Format

Bit	7	6	5	4	3	2	1	0
Byte 1	1	0	0	0	0	L	M	R
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Byte 4	X7	X6	X5	X4	X3	X2	X1	X0
Byte 5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

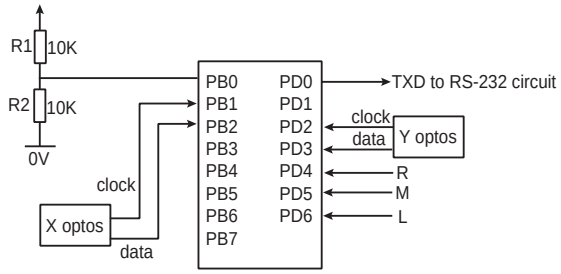
Bytes 2 and 4 are identical copies of the X movement and bytes 3 and 5 are identical copies of the Y movement.

Table 2. Microsoft Format

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	L	R	Y7	Y6	X7	X6
Byte 2	0	0	X5	X4	X3	X2	X1	X0
Byte 3	0	0	Y5	Y4	Y3	Y2	Y1	Y0

The application uses Mouse System format and has been tested on a PC using a 'Logitech' driver and found working satisfactory .

Figure 4. Port Allocation of AVR (AT90S1200)



The main program polls the buttons, sets flags to indicate the change and then sends the data to the serial port, from data registers, if a change is flagged up. The serial routine is entered every time but data is sent only if a change has occurred. The time taken is used as the bounce time. Moving the mouse generates an interrupt and updates the X and Y data.

The diagram shown in figure 4 gives the port allocations for the application. Circuit details are left to the user since you will have your own preferred methods of interfacing to the RS-232 port and for voltage generation.

A 4 Mhz resonator, with built-in capacitors, was used for the clock source, giving an instruction cycle of 250 ns.

The application was tested using a 555 timer circuit for supply voltage derivation and a MAX 232 chip for serial level conversion.

Implementation

The firmware comprises :

- An initialization/reset routine, which sets up ports and interrupts, a continuous loop is then entered to run the main program loop consisting of two sections:
  - A pushbutton polling routine, which scans the pushbuttons and updates flags accordingly.
  - A communication routine, which sends out data to the host computer if any change has occurred.
- Two similar interrupt service routines which update X and Y counters and set flags to tell the foreground process that a change has occurred.

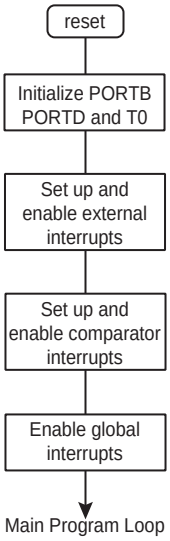
Reset/Initialization Function

This routine is entered once on power up or reset. Since it is unusual for a mouse to have a reset button this will be entered on power up. The port pins are set up as inputs/outputs as required and the pull-ups turned on for the push buttons and encoder inputs. Unused pins (6 free I/O) are configured as outputs to reduce power consumption and avoid noise pick-up. Timer 0 is set up as clock/64 to give 16 ms/count for the serial routine. The two interrupt sources are then set up and enabled such that they interrupt on the falling edge of the encoder CLOCK inputs.

For the external interrupt input line (PD2) this is done by setting bit 1 (ISC01) of the MCUCR to set falling edge interrupts and setting the INT0 bit in GIMSK to enable it.

For the analog comparator bit 1 (ACIS1) of ACSR is set to trigger interrupts on the falling edge and enabled by setting bit 3 (ACIE). Figure 5 illustrates the program flow.

Figure 5. Flow chart for 'Reset' Routine



Main Program Loop

Global interrupts are now enabled to allow the mouse positons to be updated by the X and Y background functions. These automatically update the coordinate bytes for the foreground function to send and set the flags. The pushbuttons are now interrogated and, if one is pressed, the appropriate bits in byte 1 of the word to be sent are updated. It also sets a flag to indicate to the serial function that a change has occurred.

The serial function is now entered for each byte. Since the coordinate bytes are updated by the interrupt function these will also be sent.

The flowchart in Figure 6 shows this flow.

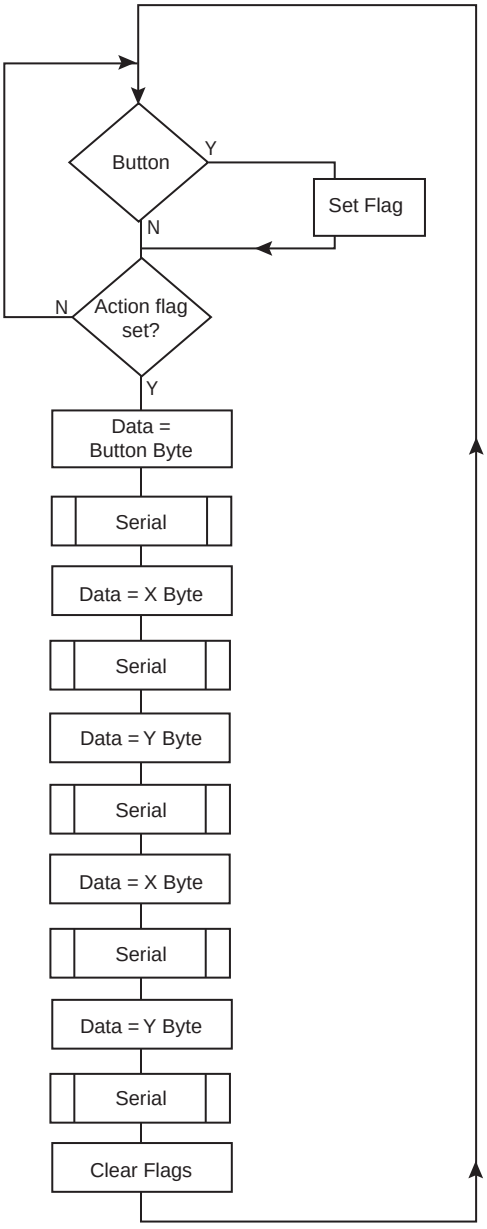
Serial Function

The serial function organizes the 1200 baud timing using timer 0 as a time reference by polling TCNT0. The byte of data is only sent if the flag is set indicating that a change has occurred. The time taken for this function is used as the debounce time for the button press. Since the time reference is in hardware and the interrupt routines are very quick, minimal error will be introduced in the serial timing by moving the mouse. Timer 0 has the prescaler switched in during the initialization stage to divide the clock by 64 giving 16 µs per count. For 1200 Baud the requisite 1/1200 second is thus achieved at a count of 52 in TCNT0 (52 * 16 µs= 832 µs a -0.2 % error).

Figure 7 shows the flow chart for the serial function. The time taken around this loop, even with no flags set, is used as the debounce time for the pushbuttons.

The send processes employ a small function called 'baud' to generate the 1/1200S (833 µs). This function cleans Timer0, starts it and waits until it reaches 52 counts.

Figure 6. Flowchart of Main Program Loop.



Interrupt Service Routines

These are two similar routines reacting to the external interrupt input (INT0) for the Y clock and the analog comparator interrupt for the X clock.

They both interrogate the respective data inputs and increment or decrement the coordinate bytes accordingly and then set the flag.

Figure 7 shows the program flow for one function. The other is identical in structure, the coordinate byte effected is the only difference.

Figure 7. Flow Chart of Serial Function

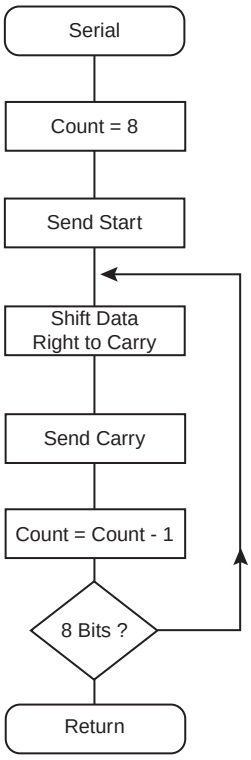


Figure 8. Flow Chart for Encoder Interrupt Service Routines

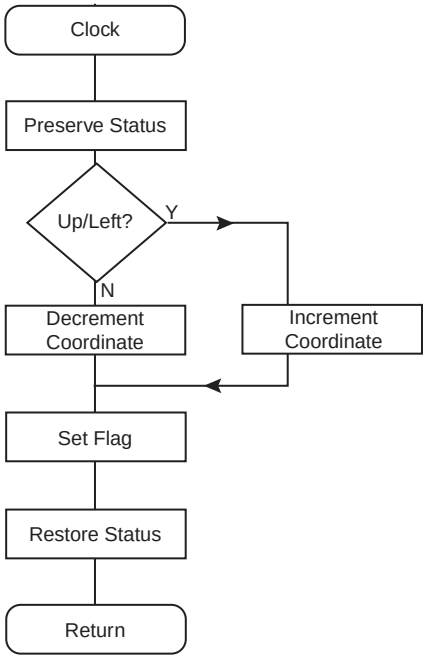


Table 3. Resources

Function	Code Size	Cycles	Register Usage	Interrupt	Description
Reset	21 words	21 cycles	R16, R18, R19, R20, R21	-	Initialization
Main	18 words	29 typical	R17, R18, R19, R20, R21, R23	-	Main program loop
Serial	17 words	30 typical	R17, R22, R23	-	Sends serial data
Baud	6 words	3,332 cycles	R16	-	833 us bit delay for serial
Xupdate	10 words	12 cycles	R0, R18, R20, R23	ANA_COMP	X encoder interrupt routine
Yupdate	10 words	12 cycles	R0, R19, R21, R23	INT_0	Y encoder interrupt routine
Total	82 words	-	R0, R16, R17, R18, R19, R20, R21, R22, R23	-	

Table 4. Peripheral Usage

Peripheral	Description	Interrupts
PD2 (INT0)	Y encoder service routine	External Interrupt 0 (falling edge triggered)
PB1 (ANA_COMP)	X encoder service routine	Analogue comparator interrupt (falling edge)
Port B 3 I/O pins	X encoder opto. input	-
Port D 6 I/O pins	Y encoder opto input and pushbutton inputs	-

```
;**** A P P L I C A T I O N   N O T E   A V R  3 8 0 ***********************
;*
;* Title:          Simple Serial Mouse Controller
;* Version:        1.0
;* Last Updated:   98.10.23
;* Target:         AT90S1200
;*
;* Support E-mail: avr@atmel.com
;*
;* DESCRIPTION
;* This Application note covers a program to provide a simple serial
;* mouse controller powered from the RS-232 serial port of the host
;* computer and providing X, Y and three mouse buttons.
;* The quadrature encoders of the mouse mechanism are interrupt driven
;* to simplify programming and improve performance.  The serial routine
;* runs at 1200 Baud and uses timer 0 for the reference frequency.
;* The application uses the economical AT90S1200 and shows how this
;* device, which has only one external interrupt line, can be adapted to
;* have two, using the analogue comparator inputs.
;* A 4 MHz clock (crystal or resonator) is assumed for the timing.
;*****************************************************************************

;***** Registers used by all programs
;******Global variables used by routines

.def    temp    =r16                            ;temporary store
.def    data    =r17                            ;byte 1 storage for mouse buttons and data for serial
.def    x_data  =r18                            ;X data
.def    y_data  =r19                            ;Y data
.def    x_datas =r20                            ;X data to be transmited
.def    y_datas =r21                            ;Y data to be transmited
.def    bitcnt  =r22                            ;bit counter for serial routine
.def    flag    =r23                            ;flag byte for passing action to serial routine

;*****7      6     5     4     3     2         1     0
;  Global    X     X     X     X     L     M   R
;  change          unused bits       Switch data
.equ    action     = 7                          ;action flag is top bit

;Port B pins
```

```
.equ    xclock  =1                              ;X encoder clock input (ana.interrupt)
.equ    xdata   =2                              ;X encoder data input

;Port D pins

.equ    TXD     =1                              ;Transmit data line for serial output
.equ    yclock  =2                              ;Y encoder clock input (ext. interrupt)
.equ    ydata   =3                              ;Y encoder data input
.equ    R       =4                              ;Right switch (active low)
.equ    M       =5                              ;Middle switch (active low)
.equ    L       =6                              ;Left switch (active low)

.include "1200def.inc"

;***** Registers used by interrupt service routines

.def    status =r0                              ;low register to preserve status register


;****Source code*************************************************
.cseg                                           ;CODE segment
.org 0
  rjmp    reset                                 ;Reset handler
  rjmp    yupdate                               ;ext. interrupt
  reti                                          ;unused timer counter overflow
  rjmp    xupdate                               ;analog interrupt

;*** Reset handler ***********************************************
;*** to provide initial port, timer and interrupt setting up

reset:
  ;    LDI     R16,low(RAMEND); Initialization of  Stack pointer
  ;    OUT     SPL,R16
  ;    LDI     R16,high(RAMEND); Commented out since 1200 has hardware stack
  ;    OUT     SPH,R16

  ldi     temp,0xF8;
  out     DDRB,temp                             ;initialize port B
  ldi     temp,0x03
  out     DDRD,temp                             ;initialize port D
  ldi     temp,0x06
  out     PORTB,temp                            ;turn on pull ups for X encoder opto
  ldi     temp,0x7f                             ;turn on pull ups for Y encoder opto
  out     PORTD,temp                            ;and mouse switches. TXD high (no start)
  ldi     temp,0x03                             ;timer prescalar clock/64 (62.5 kHz)
  out     TCCR0,temp                            ;for serial baud rate
  ldi     temp,0x02                             ;set falling edge trigger
  out     MCUCR,temp                            ;for Y encoder clock input
  ldi     temp,0x40                             ;enable external interrupts
```

```
  out     GIMSK,temp
  ldi     temp,0x0A                       ;set up falling edge trigger on analog
  out     ACSR,temp                       ;and enable X encoder clock interrupt
  clr     x_data                          ;initialise data bytes
  clr     y_data
  sei                                     ;enable global interrupts

;*********Main program loop - with interrupts enabled************
main:

  sbis    PIND,L                          ;Left switch pressed?
  sbr     flag,0x84                       ;yes set L flag and global
  sbis    PIND,M                          ;Middle switch pressed?
  sbr     flag,0x82                       ;yes set M flag and global
  sbis    PIND,R                          ;Left switch pressed?
  sbr     flag,0x81                       ;yes set R flag and global

  sbrs    flag,action                     ;Send data frame if required
  rjmp    main                            ;If not, check buttons again

  mov     data,flag                       ;Move flag to serial databuffer.
  mov     x_datas,x_data                  ;Make copys og Xdata and Ydata
  mov     y_datas,y_data                  ;to keep them form changing during serial-transmition
  clr     flag                            ;Clear flag

  ; Send frame

  rcall   serial                          ;send it
  mov     data,x_datas                    ;get X data
  rcall   serial                          ;and send it
  mov     data,y_datas                    ;get Y data
  rcall   serial                          ;and send it
  mov     data,x_datas                    ;get X data
  rcall   serial                          ;and send it
  mov     data,y_datas                    ;get Y data
  rcall   serial                          ;and send it

  rjmp    main                            ;repeat loop


;***************************************************************************
;*
;* "putchar" (from Atmel appnote AVR305)
;*
;* This subroutine transmits the byte stored in the "Txbyte" register
;* The number of stop bits used is set with the sb constant
;*
;* Number of words      :14 including return
;* Number of cycles     :Depens on bit rate
;* Low registers used   :None
```

```
;* High registers used   :2 (bitcnt,data)
;* Pointers used         :None
;*
;***************************************************************************
.equ     sb    =1                                ;Number of stop bits (1, 2, ...)

serial:
putchar:
  ldi     bitcnt,9+sb                           ;1+8+sb (sb is # of stop bits)
  com     data                                  ;Inverte everything
  sec                                           ;Start bit

putchar0:
  brcc    putchar1                              ;If carry set
  cbi     PORTD,TxD                             ; send a '0'
  rjmp    putchar2                              ;else

putchar1:
  sbi     PORTD,TxD                             ; send a '1'
  nop

putchar2:
  rcall   baud                                  ;One bit delay

  lsr     data                                  ;Get next bit
  dec     bitcnt                                ;If not all bit sent
  brne    putchar0                              ;send next
                                                ;else
  ret                                           ;return

;*********Baud rate timing routine*******************************
;****generates 1/Baud rate delay (833 us for 1200 Baud)**********

baud:
  clr     temp                                  ;clear timer 0
  out     TCNT0,temp                            ;and start counting
tagain:
  in      temp,TCNT0                            ;read timer 0
  cpi     temp,52                               ;is it 832 us?
  brlo    tagain                                ;no try again
  ret                                           ;yes return

;***********Analog comparator interrupt service routine **********
;***updates X coordinates when the mouse is moved left/right/down***

xupdate:
  in      status,SREG                           ;preserve status register
  sbic    PINB,xdata                            ;moving  right?
  rjmp    right                                 ;yes increase X coordinate
  dec     x_data                                ;no decrement X coordinate
```

```
xexit:
  sbr     flag,0x80                       ;set global flag
  out     SREG,status                     ;restore status register
  reti                                    ;and return to main loop

right:
  inc     x_data                          ;increment Y coordinate
  rjmp    xexit                           ;and return

;***********External interrupt service routine ********************
;***updates Y coordinates when the mouse is moved up/down***********

yupdate:
  in      status,SREG                     ;preserve status register
  sbic    PIND,ydata                      ;Moving up?
  rjmp    up                              ;yes increase Y coordinate
  dec     y_data                          ;no decrement Y coordinate
yexit:
  sbr     flag,0x80                       ;set global flag
  out     SREG,status                     ;restore status register
  reti                                    ;and return to main loop

up:
  inc     y_data                          ;increment Y coordinate
  rjmp    yexit                           ;and return
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686677
FAX (44) 1276-686697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road
Tsimshatsui East
Kowloon, Hong Kong
TEL (852) 27219778
FAX (852) 27221369

Japan
Atmel Japan K.K.
Tonetsu Shinkawa Bldg., 9F
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4 42 53 60 00
FAX (33) 4 42 53 60 01

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

xxxxA–02/99/xM