

Abstract

AVR 2006 Design Contest
Entry registration number: AT3283
Entry title: Positional Sensor Interface
Product: Atmega16

Description

This project implements and demonstrates an interface which supports up to 3 moderately precise non-contact XY sensors, 3 low resolution rotational sensors, and 9 switches using a simple protocol on a single communications channel. It was developed primarily to provide low cost and easily implemented sensors for homebrew robotics projects. The project includes 2 simple demonstration applications which utilize the interface.

The project uses the **ATmega16** and the **STK500 AVR Starter Kit** because the author had both on hand. Any of several other AVR controllers would have sufficed, but the author chose not to sacrifice his AVR Butterfly because of its inherent cuteness.

The sensors which the interface controls are components in one to three inexpensive (under \$10 each) optical scroll mice. Each mouse has 3 switches, a scroll wheel, and an optical XY sensor. A typical use envisioned for this interface might have the mouse (or its extracted innards) in a stationary mounting, with some external surface passing in front of its optics to provide the relative motion which is detected and reported.

It is assumed that this interface would most often be used with another microcontroller as its host, and for that reason the communications channel would logically be at digital signal levels. However, because the STK500 includes an easily accessible spare RS-232 capability, this method was chosen for the communications link to the host. RS-232 also had the advantage of simplifying debugging, and all features of the finished interface may be fully utilized via terminal software.

The mice are wired directly to the AVR, so the interface consists of no external components. In this project the mice are powered with an external 5v supply since it isn't clear what it takes to fry the STK500 power capabilities. Implementing the interface by itself would be the cost of the AVR and the RS232 level converter (unless a digital I/O scheme were used instead), plus power, reset, etc.

A simple command language is implemented which allows the host to control and interrogate the interface and unused AVR I/O pins (mice use 2 pins each). Mouse button events can also be recognized by the interface and mapped to the I/O pins (in a variety of formats) without the need of host intervention. For example, the

interface by itself could light an LED or signal other hardware when a switch is closed.

To demonstrate some of the features of the interface, two devices have been cobbled together along with some simple PC host software. These demonstrations, while possibly short on practicality, should give a flavor of the types of uses the interface may find.

The **MouseMobile** demonstrates two mice in fairly conventional use. Built in the guise of a car, its host software displays a graphic trace of its position as it moves and rotates. Acting as a digitizer and distance measurer, it can also locate the midpoint of an arbitrarily drawn curve.

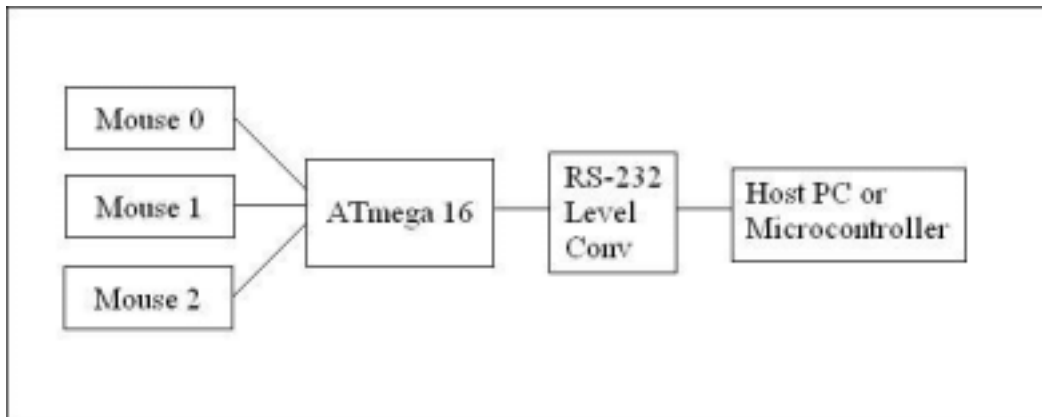
The **Mousitizer** is a polar coordinate digitizer and demonstrates how the interface can use an XY sensor to detect rotational motion and the rotational sensor to detect linear motion.

Project Photograph



The AVR STK500 Starter Kit with a block of four PS/2 connectors scavenged from an old KVM switch. The power supply is used to power the mice via the PS/2 connectors.

Block Diagram



The interface supports 3 mice.

The Batmobile Wannabe demonstration uses 2 unmodified mice.

The Mousitizer demonstration uses 1 mouse which has been disassembled and parts strewn asunder.

Software Code Sample

The entire interface (AVR code) was written in assembly language using AVR Studio 4. All code is original. The host PC programs for the two demonstrations were written in Visual Basic.

Part of the mouse interrupt handler:

```
; =====  
;      mint2 - mouse interrupt handler, part deux  
;  
;      mouse number will be in r0 (0, 1, or 2)  
;      r16 may be used without save/restore (caller does it)  
;      end with ret, not reti  
;  
;      status: 0 - idle (initial and after read or write)  
;              1 - sending setup in progress (ignore interrupts)  
;              2 - sending data  
;              3 - receiving data  
;              4 - received stop bit (ok for new receive, but  
;                  if sendmouse sees this it has to wait for  
;                  the line to go high  
;              5 - waiting ack for send  
;  
mint2:  push    ZH  
        push    ZL  
  
        rcall   moff2z  
        ldd    r16,Z+mo_sts    ; current line status  
        push   r16             ; it's arg to jumtable  
        rcall   jumtable  
        rjmp   strtrecv       ; 0 - this int's a start bit  
        rjmp   ignore         ; 1 - ignore (sendmouse did it)  
        rjmp   sendbits       ; 2 - we're clocking bits out  
        rjmp   recvbits       ; 3 - we're clocking bits in
```

```

rjmp      strtrecv          ; 4 - it's a start bit
rjmp      waitack          ; 5 - it's the send ack

;
; -----
; the line is idle or we recvd a stop bit on the last clock (4),
; and we've had a falling edge. the mouse is sending us a start bit
strtrecv:
rcall     startofbyte      ; packet mgmt
ldi      r16,3            ; should wrap up in a couple ms
rcall     starttimer
clr      r16
std      Z+mo_r,r16      ; received byte = 0
std      Z+mo_rc,r16     ; received bit count = 0
std      Z+mo_rp,r16     ; received parity = 0
ldi      r16,3            ; status: receive in progress
std      Z+mo_sts,r16

ignore:   pop      ZL
          pop      ZH
          ret

;
; -----
; reading bits from the mouse (data, parity, stop)
recvbits:
ldd      r16,Z+mo_rc      ; # bits received so far
inc      r16              ; count this one
std      Z+mo_rc,r16

          cpi      r16,9
          breq     pbit    ; if 9, this is parity bit
          brcs     dbit    ; if <9, it's a data bit

          ldd      r16,Z+mo_r ; doggies, it's the stop bit
          push     XH
          push     XL
          rcall    mhinc2x   ; point to buffer head w/ inc
          st      X,r16     ; put our byte away
          pop      XL
          pop      XH
          rcall    endofbyte ; packet mgmt

```

A piece of the main command processor:

```

;
; -----
cmd_3:   rcall     chkmouse   ; 'B' 1 byte: read switches
          breq     sendnak   ; mouse not initialized

          rcall     moff2z
          ldd      r16,Z+mo_but ; buttons and overflows
          mov      r1,r16     ; save. YX---MRL is order
          lsr      r16       ; building ----LMR
          andi     r16,0x03   ; keep MR
          sbrc     r1,0
          sbr      r16,0x04   ; put L back in

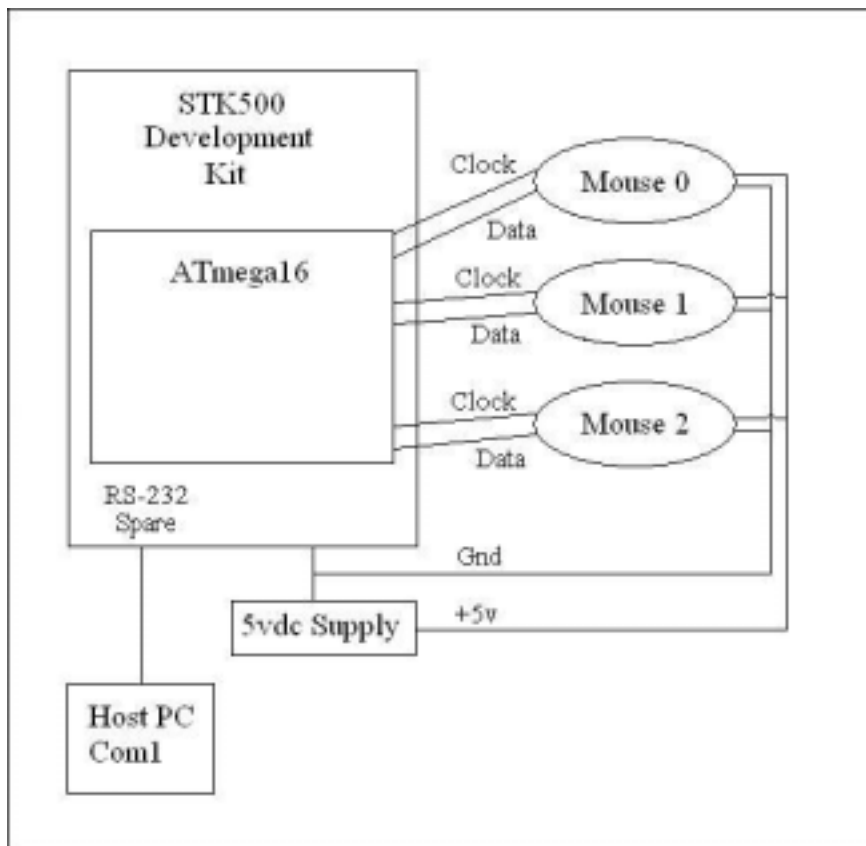
          rcall     bin2hex
          push     r16        ; arg to xmit_put
          rcall     xmit_put
          rjmp     sendack

;
; -----
; for output, set DDR <- 1
cmd_4:   rcall     prepport   ; 'D' 2 bytes: DPort - port to output
          breq     sendnak4   ; zero set means port error

          lds      XL,pindef+w_ddr ; DDR address for this port
          clr      XH         ; they're small addresses
          ser      r16
          st      X,r16     ; all bits set
          rjmp     sendack

```

Schematic



Circuitry added to STK500:

- 3 mice
- External 5vdc power supply

Mice connect to the Atmega16 through the STK500 port connections:

- Mouse 0 clock.....Port D, bit 2 (Int 0)
- Mouse 0 data.....Port D, bit 4
- Mouse 1 clock.....Port D, bit 3 (Int 1)
- Mouse 1 data.....Port D, bit 5
- Mouse 2 clock.....Port B, bit 2 (Int 2)
- Mouse 2 data.....Port B, bit 0

Other STK500 connections:

- Spare RS232 Tx.....Port D, pin 0 (Tx D)
- Spare RS232 Rx.....Port D, pin 1 (Rx D)
- LEDs.....Port C
- Switches.....Port A