Head Mounted Display Tracker

Adam Thompson
Nick Sorenson

http://www.eng.utah.edu/~adamt/HMD.html

December 8, 2006

Introduction

Small but high quality LCD monitors have recently been incorporated into goggles and eye glasses in order to give consumers a greater feeling of emersion. Doctors, for example, use a single eye monitor to project the image from an internal scope while they perform surgery. NASA is also working on a virtual reality device that projects an image into both eyes. Combined with a robotic arm, the goal is for an earth bound surgeon to be able to perform remote surgery on an orbiting astronaut. This advancement in virtual reality, however, adds little value.

The one eye approach interferes with depth perception, while an image projected into both eyes prevents the user from being able to simultaneously interact with the camera controls and the surgical arm. Because the mental demands of surgery are already extreme, interaction with the remote device should be entirely intuitive. The device should also provide doctors with a level of emersion that will allow them to feel as if they are directly operating on the patient. The purpose of this project, then, is to add motion tracking to a two eyed head mounted display. In this way, a doctor will be able to control a remote camera with head movements alone.

Project Functionality

For the end user, the *HMD Tracker* is nearly a plug and play device. The user must simply attach the output of the *Tracker* to any USB port on their computer. Note, video and audio connections will depend on what sort of HMD the user purchases. Once the USB connection is made, the tracker will be automatically detected by any windows XP/Vista computer. The HMD Tracker will be detected as a joystick.

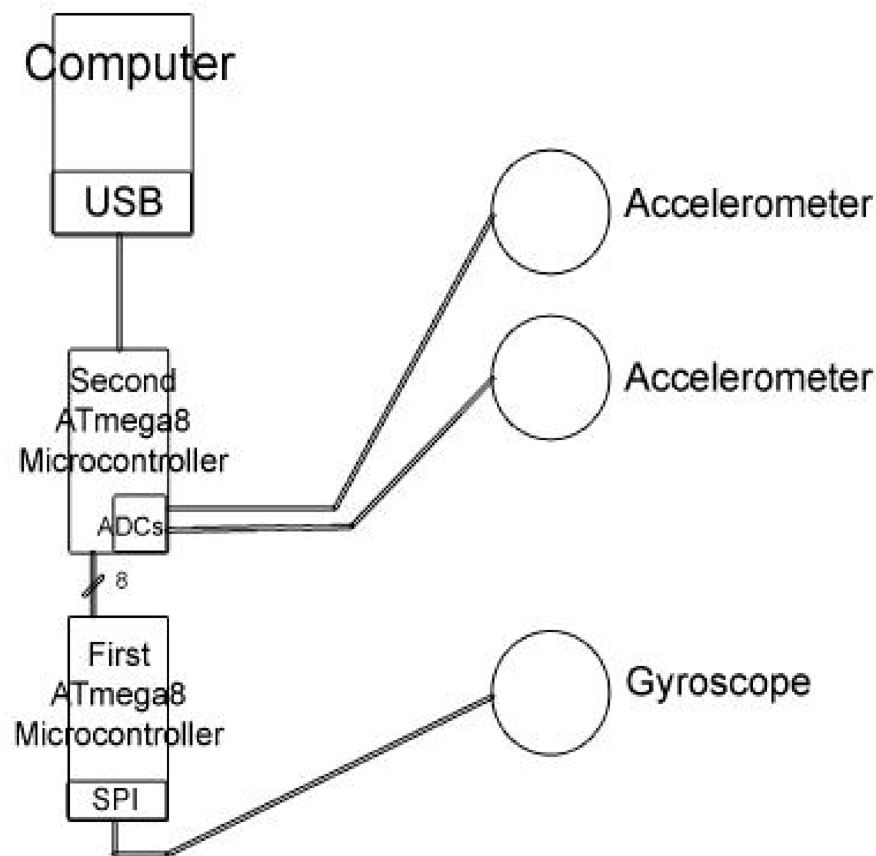
Before operation, the user will need to calibrate the Tracker for the individuals unique head range of motion. This calibration is done using Windows default calibration software. Access the calibration software by first clicking on the *Start* button, then *Control Panel*. Select *Game Controllers* and a new window will open. Choose the *HMD Tracker* device, and click on properties. A second window will open that will allow you to test the current calibration of the device. To change the calibration, use the *Settings* tab and click on calibration.

Before you begin the calibration process, be sure you are comfortably seated where you will be when you use the device. The calibration software will ask you to move your head in circles to test the range of motion for looking up and down, as well as right and left. Move your head in complete circles and at a comfortable speed. Be careful not to turn your body when you rotate your head as this motion will be included in the final calibration range. Thus, you will be unable to reach the outermost regions of the *Trackers* range without also moving your body. Finally, the software will ask you to set the range of motion of the z axis. This is done by tilting your head to the right and left, touching your ears to your shoulders.

Technical Specifications

The *HMD Tracker* includes the following components: a gyroscope, two accelerometers, two microcontrollers, and a head mounted display. While the prototype has only the gyroscope and accelerometers mounted on the HMD, a retail version would have all of the parts attached inside the plastic casing. The gyroscope is attached to one of the Atmega8 microcontrollers through a serial peripheral interface (SPI). For the prototype, this first microcontroller is running on a development board. The retail HMD Tracker will have the accelerometer mounted on a PCB. The development board was also used for programming the microcontrollers, though there are alternative methods for programming the ATMega8 chip.

The two microcontrollers are connected directly through eight of their input/output pins. As an alternative, the two accelerometers could be combined. This, however, would lead to a significant software rewrite that would likely slow the output speed, and thus reduce the *Tracker's* accuracy. The accelerometers are attached to the second microcontroller. They output an analog signal, which is connected to the microcontroller's onboard analog to digital converters (ADCs). This second microcontroller is then connected to the USB port of a computer. The video glasses RCA input is connected through an adaptor to the S-video out port of the computer's video card. The image below illustrates the design.



The programmable gyroscope reads the rotational velocity around its axis. It then performs several computations, and stores these in its internal registers. One of these computations (the one that we use in our design) is the angle out value. This is a sixteen bit number which represents how far and in what direction the gyroscope rotated since this register was last reset. This data, then, is the integration of the typical gyro out data. The gyroscope is connected to the microcontroller through the standard SPI interface, using four wires (MOSI, MISO, ~CS, and clock). This first microcontroller is configured as the master device in the SPI setup.

The output of the angle out, then, is obtained by reading the gyroscopes 0x0E and 0x0F registers out over the MISO line. The microcontroller acquires the angle output

register value by transmitting either of the angle offset register addresses. As the register is a sixteen bit and only 8 bits are transmitted at a time, 0x0F will be sent twice. This returns the sixteen bit return value from the last request we made. For example, if we transmit 0x0F 0x0F(first attempt), then transmit 0x0F 0x0F(second attempt), after the last transmission we will have received the sixteen bit response to our first request. This process is illustrated in the gyroscope's datasheet on page 11.

The microcontroller then takes this sixteen bit number (held in two eight bit registers), and outputs bits 11 through 4 on its port D. The code for this microcontroller can be found in Appendix B. The reason for outputting bits 11 through 4 was two fold. First, the users head movements are inherently noisy. Thus, the lower bits were dropped to get an accurate, but not overly sensitive response. Second, the microcontroller has a limited number of available.

The eight port D pins are then wired to eight pins of the second microcontroller. The first microcontroller uses the development board's internal crystal, running at 3.69 MHz. The development kit's built in software (AVR Studio) allows the user to set clock speed via internal fuses. For this microcontroller, the fuses were left with their default values. The development kit and first microcontroller and be seen in figure 1.

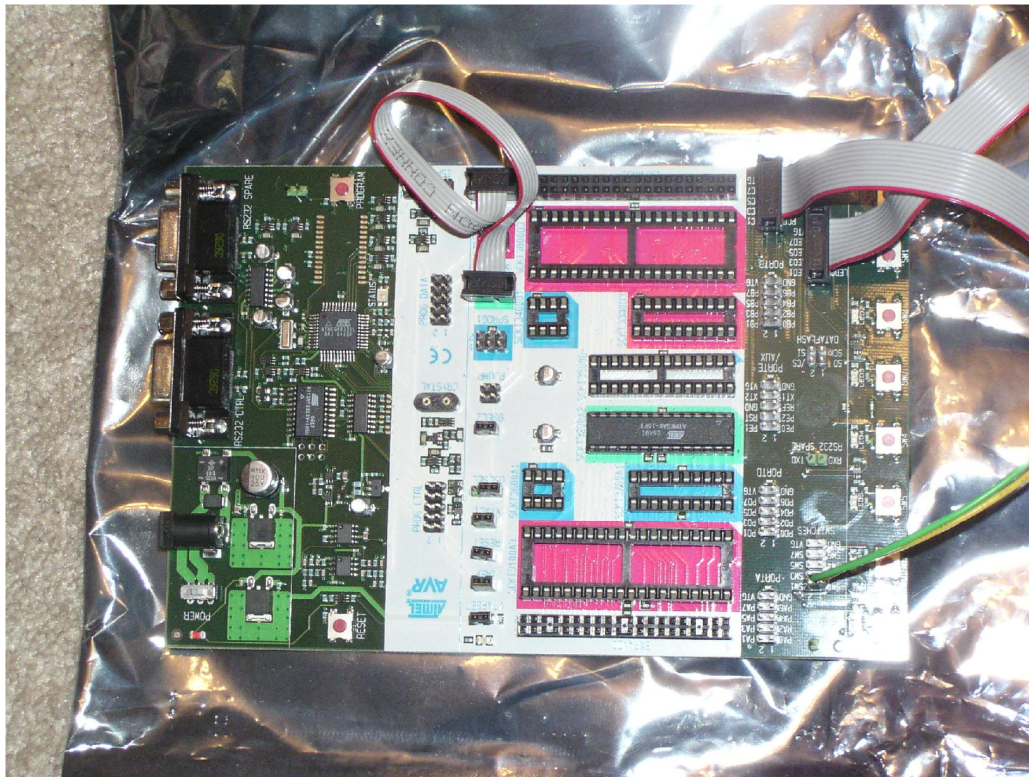


Figure 1 - Development Kit and Microcontroller

The ADXL203 is a dual axis accelerometer with an analog output. Their sensitivity is approximately 1000mV/g. The accelerometers were mounted on the glasses with one axis parallel to the earth, as shown in figure 2 below. Since the measured axis is parallel to the acceleration of gravity, they read 0g. This results in an analog output of approximately 2.5V. As the user tilts their head downward, the accelerometer mounted above the right ear begins to measure a negative acceleration, since it's no longer parallel to the earth. The maximum value it will read is -1g, if it's perpendicular to the earth. This results in a decrease in the 2.5V output. If the user looks up, the measured acceleration becomes positive, and the output increases.

A similar principle is used with the accelerometer mounted on the front of the glasses. As the user rolls their head (moving their ear towards their shoulder), the

measured acceleration varies from -1g to +1g. These two analog output voltages are connected to the second microcontroller's onboard ten bit ADCs.

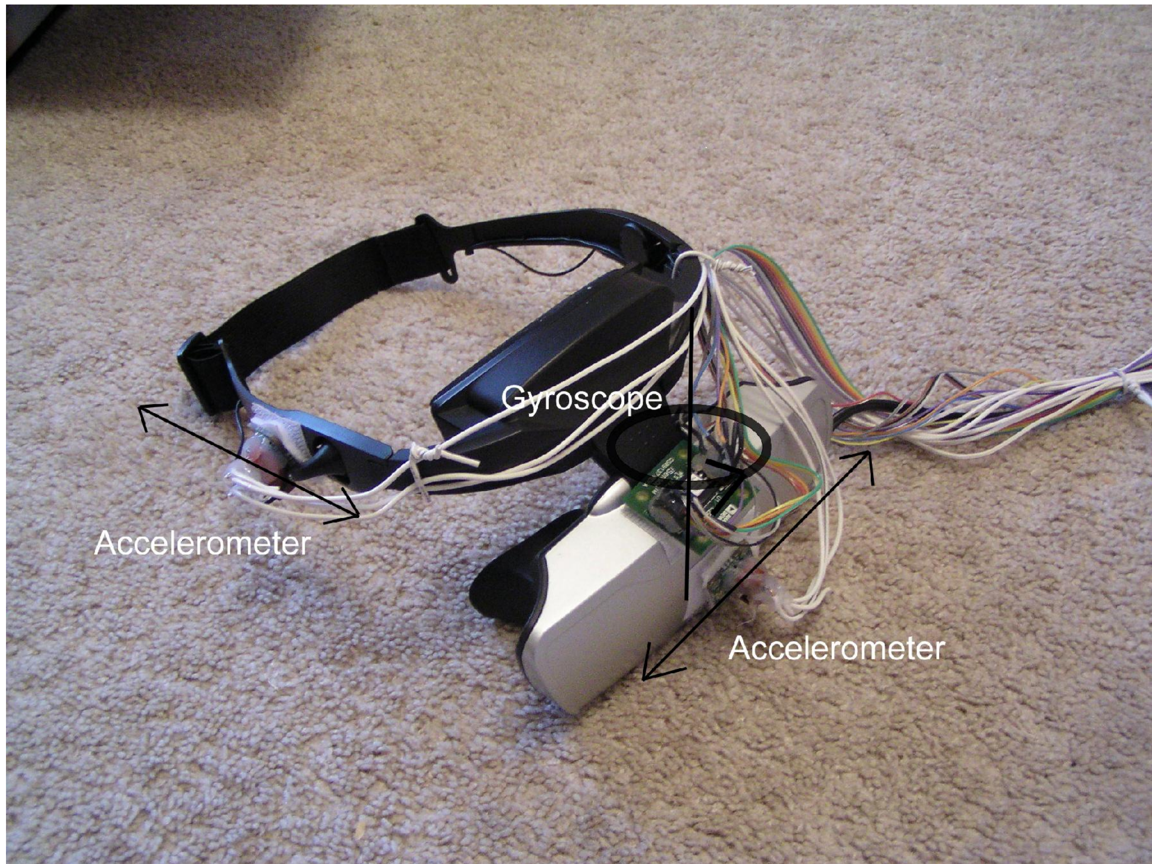


Figure 2 – Video Glasses with Accelerometers and Gyroscope

For prototyping, the second ATmega8 microcontroller is setup on a breadboard. Its crystal is running at 12 MHz. As before, certain fuses must be set in the programmer to allow the chip to run at this frequency. Page 29 of the datasheet describes the fuse options. For this speed, set CKSEL3..0 to 1000, and SUT1..0 to 10. This instructs the microcontroller to run from an external 8-12 MHz crystal. The second microcontroller reads output of the gyroscope from its pins which are connected to the first microcontroller, as described above. It ranges from -127 to 127, with zero being in the center of the screen, with the default calibration.

The second Microcontroller also reads in and converts the analog accelerometer data into a ten bit number. The ATmega8 microcontroller has six pins that can be configured to use the onboard ADCs. These can be read by first setting up the appropriate registers specifying which ADCs to use. This setup is explained in the Atmega8 datasheet beginning on page 196. It's also illustrated in the code for this microcontroller, which is found in Appendix A. The data from the accelerometer mounted above the right ear is used for the Y axis. And, the data from the accelerometer mounted on the front is used for the Z axis. Since these are ten bit values, they range from -512 to 511. The third input source for this microcontroller comes from two simple buttons. The microcontroller simply reads from these two pins, PD0 and PD1, to acquire their current status.

In order to communicate with the computer across the USB connection, the circuit is built with a configuration of resistors and diodes across the USB data lines (D- and D+). When the USB cable is initially plugged into the computer, the computer's USB controller sees these voltages on the D- and D+ lines and sends a message requesting the device's descriptors. If the device isn't currently up and running, no descriptor will be sent, and the computer will come up with an error message saying there is a problem with the device. The code running on the second microcontroller goes into an interrupt handler when it receives this initial request. It then prepares, and sends the appropriate descriptors. One of these descriptors (the report descriptor) informs the computer of the type of device that is attached, and the data that will be sent. The *HMD Tracker* is setup as a joystick with the following data: a ten bit Z axis, a ten bit Y axis, an eight bit X axis, and 2 one bit buttons. After the setup process, the second microcontroller periodically

reads from its inputs, and transmits this data to update the joysticks current state on the computer

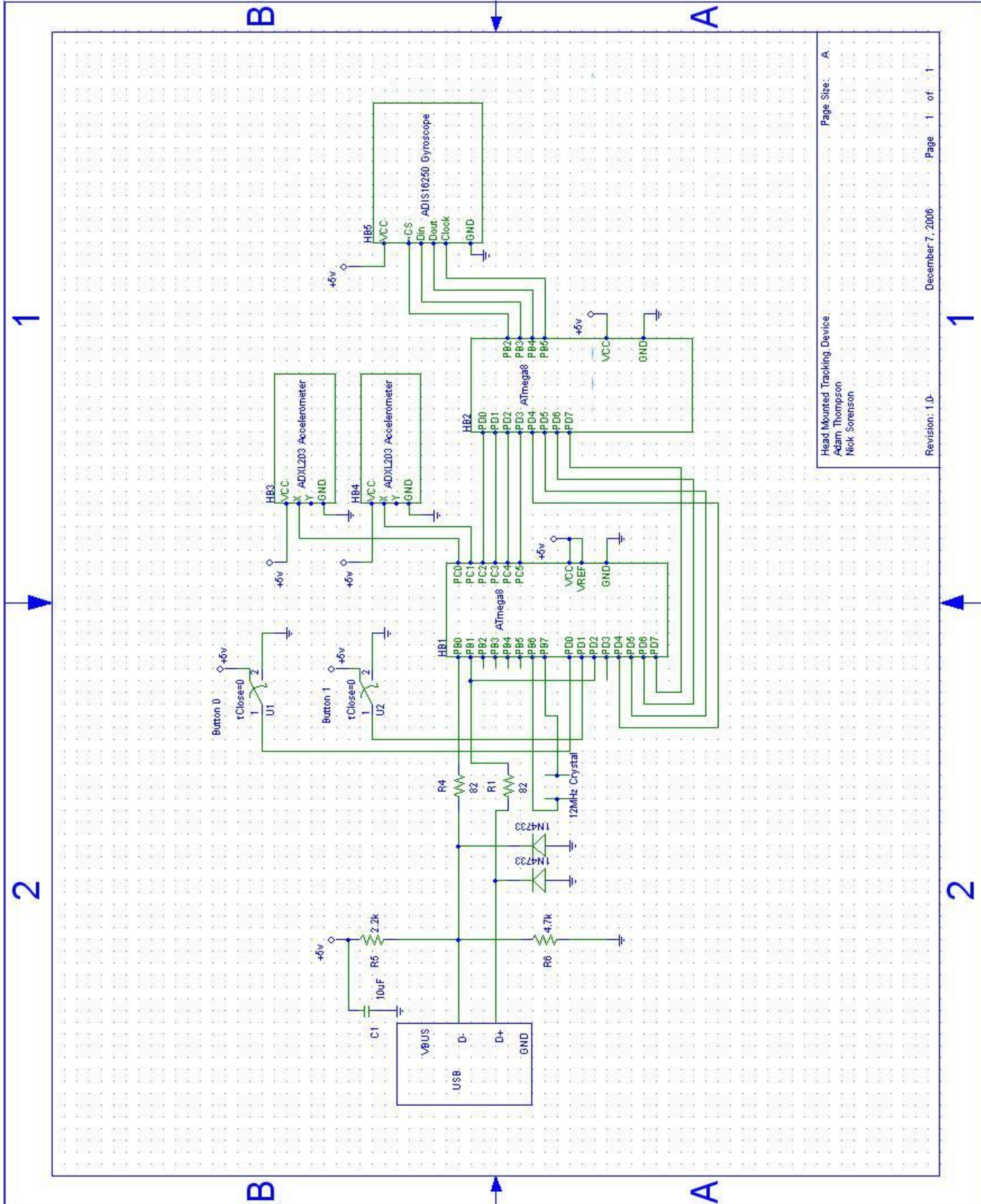


Figure 3 – Schematic of Completed Design

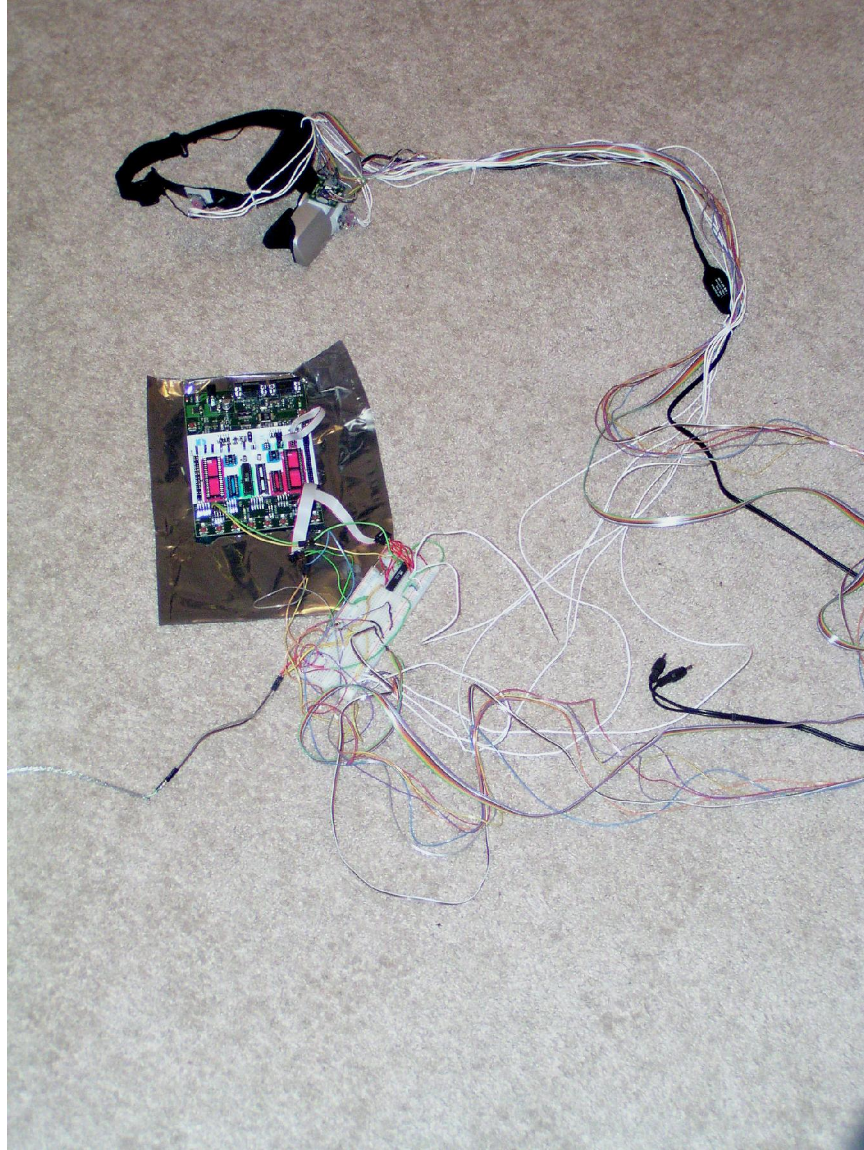


Figure 4 – The Complete Design

Conclusion

The primary difficulty we came across with our project was calculating the amount the user panned their head. Our initial design was to use the accelerometers to calculate this, but after many unsuccessful attempts we decided to include the gyroscope. This proved to be a much better and easier way to determine the panning angle. One thing that came as a bit of a surprise, was the complexity involved in programming the USB controller on our microcontroller. But thanks to several individuals that utilized this same microcontroller for USB communication, and several good guides, we were able to implement this. Through the course of this project, we were able to learn a great deal in many areas, ranging from USB specifications to how to work with accelerometers and gyroscopes.

Appendix A

```
;***************************************************************************
;* U S B   S T A C K   F O R   T H E   A V R   F A M I L Y
;*
;* Date                  :12.07.2006
;* Version               :1.0
;* Target MCU            :ATmega8
;* AUTHOR               :Adam Thompson, Nick Sorenson
;* Based on code of:
;*                                  Mindaugas Milasauskas
;*                                  Lithuania
;*                                  mindaug@mindaugas.com
;*                                  http://www.mindaugas.com
;*
;*                                  and
;*
;*                                  Ing. Igor Cesko
;*                                  Slovakia
;*                                  cesko@internet.sk
;*                                  http://www.cesko.host.sk
;*
;*
;* DESCRIPTION:
;*  USB protocol implementation into MCU with noUSB interface:
;*
;* The timing is adapted for 12 MHz crystal
;*
;*
;***************************************************************************
.include "m8def.inc"

;RSEG CODE ; This code is relocatable, RSEG
.equ    UBRR                  =UBRRL
.equ    EEAR                  =EEARL
.equ    E2END                 =127
.equ    RAMEND128             =96+127

.equ    inputport             =PINB
.equ    outputport            =PORTB
.equ    USBdirection          =DDRB
.equ    DATAplus              =1              ;signal D+ on PB1
.equ    DATAminus             =0              ;signal D- on PB0 - give on this pin pull-up
1.5kOhm
.equ    USBpinmask            =0b11111100    ;mask low 2 bit (D+,D-) on PB
.equ    USBpinmaskDplus               =~(1<<DATAplus)        ;mask D+ bit on PB
.equ    USBpinmaskDminus      =~(1<<DATAminus);mask D- bit on PB


.equ    SOPbyte               =0b10000000    ;Start of Packet byte
.equ    DATA0PID              =0b11000011    ;PID for DATA0 field
.equ    DATA1PID              =0b01001011    ;PID for DATA1 field
.equ    OUTPID                =0b11100001    ;PID for OUT field
.equ    INPID                 =0b01101001    ;PID for IN field
.equ    SOFPID                =0b10100101    ;PID for SOF field
.equ    SETUPPID              =0b00101101    ;PID for SETUP field
.equ    ACKPID                =0b11010010    ;PID for ACK field
.equ    NAKPID                =0b01011010    ;PID for NAK field
.equ    STALLPID              =0b00011110    ;PID for STALL field
.equ    PREPID                =0b00111100    ;PID for FOR field

.equ    nSOPbyte              =0b00000001    ;Start of Packet byte - reverse order
.equ    nDATA0PID             =0b11000011    ;PID for DATA0 field - reverse order
.equ    nDATA1PID             =0b11010010    ;PID for DATA1 field - reverse order
.equ    nOUTPID               =0b10000111    ;PID for OUT field - reverse order
.equ    nINPID                =0b10010110    ;PID for IN field - reverse order
.equ    nSOFPID               =0b10100101    ;PID for SOF field - reverse order
.equ    nSETUPPID             =0b10110100    ;PID for SETUP field - reverse order
```

```
.equ    nACKPID              =0b01001011    ;PID for ACK field - reverse order
.equ    nNAKPID              =0b01011010    ;PID for NAK field - reverse order
.equ    nSTALLPID            =0b01111000    ;PID for STALL field - reverse order
.equ    nPREPID              =0b00111100    ;PID for FOR field - reverse order

.equ    nNRZITokenPID        =~0b10000000   ;PID mask for Token packet
(IN,OUT,SOF,SETUP) - reverse order NRZI
.equ    nNRZISOPbyte         =~0b10101011   ;Start of Packet byte - reverse order NRZI
.equ    nNRZIDATA0PID        =~0b11010111   ;PID for DATA0 field - reverse order NRZI
.equ    nNRZIDATA1PID        =~0b11001001   ;PID for DATA1 field - reverse order NRZI
.equ    nNRZIOUTPID          =~0b10101111   ;PID for OUT field - reverse order NRZI
.equ    nNRZIINPID           =~0b10110001   ;PID for IN field - reverse order NRZI
.equ    nNRZISOFPID          =~0b10010011   ;PID for SOF field - reverse order NRZI
.equ    nNRZISETUPPID        =~0b10001101   ;PID for SETUP field - reverse order NRZI
.equ    nNRZIACKPID          =~0b00100111   ;PID for ACK field - reverse order NRZI
.equ    nNRZINAKPID          =~0b00111001   ;PID for NAK field - reverse order NRZI
.equ    nNRZISTALLPID        =~0b00000111   ;PID for STALL field - reverse order NRZI
.equ    nNRZIPREPID          =~0b01111101   ;PID for FOR field - reverse order NRZI
.equ    nNRZIADDR0           =~0b01010101   ;Address = 0 - reverse order NRZI

                                            ;status bytes - State
.equ    BaseState            =0             ;
.equ    SetupState           =1             ;
.equ    InState              =2             ;
.equ    OutState             =3             ;
.equ    SOFState             =4             ;
.equ    DataState            =5             ;

                                            ;Flags of action
.equ    DoNone                              =0
.equ    DoReceiveOutData                    =1
.equ    DoReceiveSetupData                  =2
.equ    DoPrepareOutContinuousBuffer        =3
.equ    DoReadySendAnswer                   =4
.equ    DoPrepareJoystickAnswer                     =5
.equ    DoReadySendJoystickAnswer                   =6

                        ; Joystick flags
.equ    JoystickDataRequest          =1
.equ    JoystickDataRequestBit       =0
.equ    JoystickDataReady            =2
.equ    JoystickDataReadyBit         =1
.equ    JoystickDataProcessing =4
.equ    JoystickDataProcessingBit    =2
.equ    JoystickLastDataPID          =8
.equ    JoystickLastDataPIDBit       =3
.equ    JoystickReportID             =0b00010000
.equ    JoystickReportIDBit                 =4


;.equ   CRC5poly             =0b00101               ;CRC5 polynomial
;.equ   CRC5zvysok           =0b01100               ;CRC5 remainder after successful CRC5
.equ    CRC16poly            =0b1000000000000101    ;CRC16 polynomial
;.equ   CRC16zvysok          =0b1000000000001101    ;CRC16 remainder after successful
CRC16

.equ    MAXUSBBYTES          =14                    ;maximum bytes in USB input message
.equ    NumberOfFirstBits    =10                    ;how many first bits allowed be
longer
.equ    NoFirstBitsTimerOffset =256-12800*12/1024   ;Timeout 12.8ms (12800us) to
terminate after firsts bits

.equ    InputBufferBegin     =RAMEND128-127                        ;compare of receiving
shift buffer
.equ    InputShiftBufferBegin =InputBufferBegin+MAXUSBBYTES        ;compare of receiving
buffera

.equ    OutputBufferBegin    =RAMEND128-MAXUSBBYTES-2      ;compare of transmitting
buffer
.equ    AckBufferBegin       =OutputBufferBegin-3   ;compare of transmitting buffer Ack
.equ    NakBufferBegin       =AckBufferBegin-3      ;compare of transmitting buffer Nak
```

```
.equ    ConfigByte            =NakBufferBegin-1      ;0=unconfigured state
.equ    AnswerArray           =ConfigByte-8          ;8 byte answer array
.equ    JoystickBufferBegin   = AnswerArray - MAXUSBBYTES
.equ    JoyOutputBufferLength = JoystickBufferBegin - 1
.equ    JoyOutBitStuffNumber = JoyOutputBufferLength - 1
.equ    BkpOutputBufferLength = JoyOutBitStuffNumber - 1
.equ    BkpOutBitStuffNumber = BkpOutputBufferLength - 1
.equ    JoyVal = BkpOutBitStuffNumber - 1


.equ    StackBegin            =JoyVal-1              ;low reservoir (stack is big cca 54
byte)


.def    JoystickFlags         =R1            ; Endpoint 1 interrupt status flags for
joystick reports
.def    backupbitcount        =R2            ;backup bitcount register in INT0
disconnected
.def    RAMread               =R3            ;if reading from SRAM
.def    backupSREGTimer               =R4            ;backup Flag register in Timer
interrupt
.def    backupSREG            =R5            ;backup Flag register in INT0 interrupt
.def    ACC                   =R6            ;accumulator
.def    lastBitstufNumber     =R7            ;position in bitstuffing
.def    OutBitStuffNumber     =R8            ;how many bits to send last byte -
bitstuffing
.def    BitStuffInOut         =R9            ;if insertion or deleting of bitstuffing
.def    TotalBytesToSend      =R10           ;how many bytes to send
.def    TransmitPart          =R11           ;order number of transmitting part
.def    InputBufferLength     =R12           ;length prepared in input USB buffer
.def    OutputBufferLength    =R13           ;length answers prepared in USB buffer
.def    MyUpdatedAddress      =R14           ;my USB address for update
.def    MyAddress             =R15           ;my USB address


.def    ActionFlag            =R16           ;what to do in main program loop
.def    temp3                 =R17           ;temporary register
.def    temp2                 =R18           ;temporary register
.def    temp1                 =R19           ;temporary register
.def    temp0                 =R20           ;temporary register
.def    bitcount              =R21           ;counter of bits in byte
.def    ByteCount             =R22           ;counter of maximum number of received bytes
.def    inputbuf              =R23           ;receiver register
.def    shiftbuf              =R24           ;shift receiving register
.def    State                 =R25           ;state byte of status of state machine
input/output
.def    ROMBufptrZ            =R30           ;ZL register - pointer to buffer of ROM data


;requirements on descriptors
.equ    GET_STATUS            =0
.equ    CLEAR_FEATURE         =1
.equ    SET_FEATURE           =3
.equ    SET_ADDRESS           =5
.equ    GET_DESCRIPTOR        =6
.equ    SET_DESCRIPTOR        =7
.equ    GET_CONFIGURATION     =8
.equ    SET_CONFIGURATION     =9
.equ    GET_INTERFACE         =10
.equ    SET_INTERFACE         =11
.equ    SYNCH_FRAME           =12

; Class requests
.equ    GET_REPORT            =1
.equ    GET_IDLE              =2
.equ    GET_PROTOCOL   =3
.equ    SET_REPORT            =9
.equ    SET_IDLE              =10
.equ    SET_PROTOCOL   =11

;Standard descriptor types
```

```
.equ    DEVICE                 =1
.equ    CONFIGURATION          =2
.equ    STRING                 =3
.equ    INTERFACE              =4
.equ    ENDPOINT               =5

; Class Descriptor Types
.equ    CLASS_HID              =0x21
.equ    CLASS_Report   =0x22
.equ    CLASS_Physical =0x23

;databits
.equ    DataBits5              =0
.equ    DataBits6              =1
.equ    DataBits7              =2
.equ    DataBits8              =3

;parity
.equ    ParityNone             =0
.equ    ParityOdd              =1
.equ    ParityEven             =2
.equ    ParityMark             =3
.equ    ParitySpace            =4

;stopbits
.equ    StopBit1               =0
.equ    StopBit2               =1

;user function start number
.equ    USER_FNC_NUMBER                =100


;------------------------------------------------------------------------------------------
--
;*********************************************************************
;* ;* Interrupt table
;*********************************************************************
.cseg
;------------------------------------------------------------------------------------------
--
.org 0                                        ;after reset
               rjmp    reset
;------------------------------------------------------------------------------------------
--
.org INT0addr                                 ;external interrupt INT0
               rjmp    INT0handler
;------------------------------------------------------------------------------------------
--

;------------------------------------------------------------------------------------------
--
;*********************************************************************
;*  Init program
;*********************************************************************
;------------------------------------------------------------------------------------------
--
reset:                 ;initialization of processor and variables to right values

               ldi     temp0,StackBegin       ;initialization of stack
               out     SPL,temp0

               clr     XH                             ;RS232 pointer
               clr     YH                             ;USB pointer
               clr     ZH                             ;ROM pointer
               clr     JoystickFlags
               ldi             temp0,JoystickDataReady
               or              JoystickFlags,temp0


               clr     MyUpdatedAddress       ;new address USB -  non-decoded
               rcall   InitACKBufffer         ;initialization of ACK buffer
```

```
		rcall	InitNAKBufffer		;initialization of NAK buffer
		rcall	InitJoystickBufffer		;initialization of Joystick buffer

		rcall	USBReset			;initialization of USB addresses

		ldi	temp0,0b00000100	;set pull-up on PORTB
		out	PORTB,temp0
		ldi	temp0,0b11000000	;set pull-up on PORTC
		out	PORTC,temp0
		ldi	temp0,0b01111011	;set pull-up on PORTD
		out	PORTD,temp0

		ldi	temp0,0b00000000	;set input on PORTD
		out	DDRD,temp0

		clr	temp0			;
		out	EEARH,temp0		;zero EEPROM index

	; prepare ADC
	;	ldi	temp0, 0b10000101	;ADC enable ( /32 )
							; no start yet
							; Single Conversion mode
							; no interrupts
							;set ADC prescaler to 12MHz /
32 = 375kHz
		ldi	temp0, 0b10000110	;ADC enable
							; no start yet
							; Single Conversion mode
							; no interrupts
							;set ADC prescaler to 12MHz /
64 = 187.5kHz
		out ADCSRA,temp0

		ldi	temp0, 0b01000000	;AVCC refernce, PortC 0 - X
							; clear ADLAR for 10 bit
conversion
		out ADMUX, temp0


		ldi	temp0,0x0F		;INT0 - respond to leading edge
		out	MCUCR,temp0		;
		ldi	temp0,1<<INT0		;enable external interrupt INT0
		out	GIMSK,temp0

;------------------------------------------------------------------------------------------
--
;*********************************************************************
;*  Main program
;*********************************************************************
		sei					;enable interrupts globally
Main:
		sbis	inputport,DATAminus	;waiting till change D- to 0
		rjmp	CheckUSBReset		;and check, if isn't USB reset

		cpi	ActionFlag,DoReceiveSetupData
		breq	ProcReceiveSetupData
		cpi	ActionFlag,DoPrepareOutContinuousBuffer
		breq	ProcPrepareOutContinuousBuffer
		sbrc	JoystickFlags,JoystickDataRequestBit
		rjmp	ProcJoystickRequest
		rjmp	Main

CheckUSBReset:
		ldi	temp0,255		;counter duration of reset (according to
specification is that cca 10ms - here is cca 100us)
WaitForUSBReset:
		sbic	inputport,DATAminus	;waiting till change D+ to 0
		rjmp	Main
		dec	temp0
		brne	WaitForUSBReset
		rcall	USBReset
```

```
                rjmp    Main

ProcPrepareOutContinuousBuffer:
                rcall   PrepareOutContinuousBuffer    ;prepare next sequence of answer to
buffer
                ldi     ActionFlag,DoReadySendAnswer
                rjmp    Main
ProcReceiveSetupData:
                ldi     USBBufptrY,InputBufferBegin   ;pointer to begin of receiving buffer
                mov     ByteCount,InputBufferLength   ;length of input buffer
                rcall   DecodeNRZI            ;transfer NRZI coding to bits
                rcall   MirrorInBufferBytes   ;invert bits order in bytes
                rcall   BitStuff              ;removal of bitstuffing
                ;rcall  CheckCRCIn            ;rcall  CheckCRCIn            ;check CRC
                rcall   PrepareUSBOutAnswer   ;prepare answers to transmitting buffer
                ldi     ActionFlag,DoReadySendAnswer
                rjmp    Main



;*********************************************************************
;*  Joystick Requests Processing Routine
;*********************************************************************

ProcJoystickRequest:
        ; clear ready flag to avoid data conflict
                mov             temp0, JoystickFlags
                ldi             temp0,0xFF
                andi            temp0,~JoystickDataReady
                andi    temp0,~JoystickDataRequest ; clear request flag to avoid call on
next cycle
                and             JoystickFlags,temp0

                sbrc    JoystickFlags,JoystickReportIDBit
                rjmp    ProcessJoystickReport_2

ProcessJoystickReport_1:
                ;save report ID
                ldi             temp0, 1
                sts             JoystickBufferBegin+2,temp0
;;;;;;;;;;;;;;;;;;;;;;;;
; Read value from ADC X channel
                ldi             temp0, 0       ;PortC 0 - X
                rcall   ReadADC_10

        ; Store X value
;               andi    temp0,0xFC     ;Drop lower 2 bits
                sts             JoystickBufferBegin+4,temp0
                sts             JoystickBufferBegin+5,temp1
                mov             temp3, temp1   ; store to combine with Y


;;;;;;;;;;;;;;;;;
; Read value from ADC Y channel
                ldi             temp0, 1       ;PortC 1 - Y
                rcall   ReadADC_10


                ldi             temp2, 0
                add             temp2, temp0
        ; shift for Y value
;               andi    temp0,0xFC     ;Drop lower 2 bits
                clc
                rol             temp0
                rol             temp1
                rol             temp0
                rol             temp1
                or              temp3, temp0
                ori     temp1, 0b11000000
                sts             JoystickBufferBegin+5,temp3
                sts             JoystickBufferBegin+6,temp1
```

```
;;;;;;;;;;;;;;;;;
; Read value from Dial ADC channel
		;lds		temp0, JoystickBufferBegin+4	;read in deah
	;PortC 5 - Dial

		cbi	DDRC,2
		cbi	DDRC,3
		cbi	DDRC,4
		cbi	DDRC,5

		in		temp0, PINC
		in		temp1, PIND
		andi	temp0,0b00111100
		lsr		temp0
		lsr		temp0
		andi	temp1,0b11110000
		or		temp0,temp1


		sts		JoystickBufferBegin+3,temp0

		sbi	DDRC,2
		sbi	DDRC,3
		sbi	DDRC,4
		sbi	DDRC,5

		ldi		temp3,JoystickReport1Size		;Joystick report size
		rjmp	SendJoystickReport

ProcessJoystickReport_2:
		;save report ID
		ldi		temp0, 2
		sts		JoystickBufferBegin+2,temp0	; write Report ID


Buttons:
		in		temp3,PIND
		;Invert the button input
		com		temp3
		sts		JoystickBufferBegin+3,temp3

		ldi		temp3,JoystickReport2Size		;Joystick report size

SendJoystickReport:
		ldi		temp0, JoystickReportID		; flip report ID
		eor		JoystickFlags, temp0


	; simulate call to AddCRCOut
	; simply push point of return onto stack
		ldi		temp0, low(AddCRCOutReturn)	;ROMpointer to descriptor
		push	temp0
		ldi		temp0,  high(AddCRCOutReturn)
		push	temp0


		ldi	USBBufptrY,JoystickBufferBegin

		ldi	ByteCount,2		;length of output buffer (only SOP and PID)
		add	ByteCount,temp3		;+ number of bytes
		push	USBBufptrY
		push	ByteCount
	rjmp	AddCRCOut_2		;addition of CRC to buffer

;ReadButtonsRow:
;		nop					; for synchronization
;		in		temp0,PIND
;		andi	temp0,0b01111011	;mask out PD2 and PD7 which is INT0 and
Column 4
```

```
;               in              temp1,PINB
;               andi    temp1,0b00000100       ;extract bit PB2
;               or              temp0,temp1                    ;merge values
;               com             temp0
;               andi    temp0,0b01111111
;               ret


ReadADC_10: ; Input:   temp0 contains number of ADC input to read
                        ; Return:      temp0 contains bits 0-7
                        ;                       temp1 contains bits 8-9
                andi temp0, 0b00000111
                ori      temp0, 0b01000000     ;AVCC refernce, clear ADLAR for 10 bit
conversion
                                                                ;
                out ADMUX, temp0
                sbi             ADCSRA, ADSC   ; start conversion
WaitForADC_10:
                sbic    ADCSRA, ADSC
                rjmp    WaitForADC_10

                in      temp0,ADCL
                in      temp1,ADCH

                cpi     temp0,0
                brne    Subtract512
                cpi     temp1,0
                brne    Subtract512
                ldi     temp0,1
Subtract512:
        ; subtract 512 to get values in range (-511, 511)
                subi    temp0, 0x00
                sbci    temp1, 0x02
                andi    temp1, 0x03

                ret

ReadADC_8: ; temp0 contains number of ADC input to read
; Read value from ADC X channel
                andi temp0, 0b00000111
                ori      temp0, 0b01100000     ;AVCC refernce, set ADLAR for 8 bit
conversion
                out ADMUX, temp0
                sbi             ADCSRA, ADSC   ; start conversion
WaitForADC_8:
                sbic    ADCSRA, ADSC
                rjmp    WaitForADC_8

                in      temp0,ADCH

Subtract128:
        ; subtract 128 to get values in range (-127, 127)
                subi    temp0, 0x80
                ret


TenBitto8:      ; Input:       temp0 contains bits 0-7
                        ;                       temp1 contains bits 8-9
                        ; Return:      temp0 contains bits 2-9
                andi temp0,0xFC        ;Drop lower 2 bits
                lsr temp0
                lsr     temp0
                lsl     temp1
                lsl     temp1
                lsl     temp1
                lsl     temp1
                lsl     temp1
                lsl     temp1
                or      temp0,temp1
                ret
```

```
AddCRCOutReturn:
		inc	ByteCount		;length of output buffer + CRC16
		inc	ByteCount


	; Backup Control pipe buffer pointers to save Control pipe state
		mov	temp0, OutputBufferLength
		sts BkpOutputBufferLength,temp0
		mov temp0, OutBitStuffNumber
		sts BkpOutBitStuffNumber,temp0


		inc	BitStuffInOut			;transmitting buffer - insertion of
bitstuff bits
		ldi	USBBufptrY,JoystickBufferBegin	;to transmitting buffer
		rcall	BitStuff
;		mov	OutputBufferLength,ByteCount	;length of answer store for
transmiting
		clr	BitStuffInOut			;receiving buffer - deletion of
bitstuff bits


	; copy to Joystick buffer
		sts JoyOutputBufferLength, ByteCount
		sts JoyOutBitStuffNumber, OutBitStuffNumber


	; Restore Control pipe buffer pointers
		lds		temp0, BkpOutputBufferLength
		mov		OutputBufferLength, temp0
		lds		temp0, BkpOutBitStuffNumber
		mov		 OutBitStuffNumber,temp0



	; set joystick data ready flag
		ldi		temp0,JoystickDataReady
		or		JoystickFlags,temp0
;TestpointEnd -------------

		rjmp	Main


;*********************************************************************
;*  Main program END
;*********************************************************************
;------------------------------------------------------------------------------------------
--
;*********************************************************************
;*  Interrupt0 interrupt handler
;*********************************************************************
INT0Handler:					;interrupt INT0
		in	backupSREG,SREG
		push	temp0
		push	temp1

		ldi	temp0,3			;counter of duration log0
		ldi	temp1,2			;counter of duration log1
		;waiting for begin packet
CheckchangeMinus:
		sbis	inputport,DATAminus	;waiting till change D- to 1
		rjmp	CheckchangeMinus
CheckchangePlus:
		sbis	inputport,DATAplus	;waiting till change D+ to 1
		rjmp	CheckchangePlus
DetectSOPEnd:
		sbis	inputport,DATAplus
		rjmp	Increment0		;D+ =0
```

```
Increment1:
                ldi     temp0,3                 ;counter of duration log0
                dec     temp1                   ;how many cycles takes log1
                nop
                breq    USBBeginPacket          ;if this is end of SOP - receive packet
                rjmp    DetectSOPEnd
Increment0:
                ldi     temp1,2                 ;counter of duration log1
                dec     temp0                   ;how many cycles take log0
                nop
                brne    DetectSOPEnd            ;if there isn't SOF - continue
                rjmp    EndInt0HandlerPOP2
EndInt0Handler:
                pop     ACC
                pop     R26
                pop     temp3
                pop     temp2
EndInt0HandlerPOP:
                pop     USBBufptrY
                pop     ByteCount
                mov     bitcount,backupbitcount         ;restore bitcount register
EndInt0HandlerPOP2:
                pop     temp1
                pop     temp0
                out     SREG,backupSREG
                ldi     shiftbuf,1<<INTF0       ;zero interruptu flag INTF0
                out     GIFR,shiftbuf
                reti                            ;otherwise finish (was only SOF - every
millisecond)

USBBeginPacket:
                mov     backupbitcount,bitcount         ;backup bitcount register
                in      shiftbuf,inputport      ;if yes load it as zero bit directly to
shift register
USBloopBegin:
                push    ByteCount               ;additional backup of registers (save of
time)
                push    USBBufptrY
                ldi     bitcount,6              ;initialization of bits counter in byte
                ldi     ByteCount,MAXUSBBYTES   ;initialization of max number of received
bytes in packet
                ldi     USBBufptrY,InputShiftBufferBegin        ;set the input buffer
USBloop1_6:
                in      inputbuf,inputport
                cbr     inputbuf,USBpinmask     ;unmask low 2 bits
                breq    USBloopEnd              ;if they are zeros - end of USB packet
                ror     inputbuf                ;transfer Data+ to shift register
                rol     shiftbuf
                dec     bitcount                ;decrement bits counter
                brne    USBloop1_6              ;if it isn't zero - repeat filling of shift
register
                nop                             ;otherwise is necessary copy shift register
to buffer
USBloop7:
                in      inputbuf,inputport
                cbr     inputbuf,USBpinmask     ;unmask low 2 bits
                breq    USBloopEnd              ;if they are zeros - end of USB packet
                ror     inputbuf                ;transfer Data+ to shift register
                rol     shiftbuf
                ldi     bitcount,7              ;initialization of bits counter in byte
                st      Y+,shiftbuf             ;copy shift register into buffer and
increment pointer to buffer
USBloop0:                                       ;and start receiving next byte
                in      shiftbuf,inputport      ;zero bit directly to shift register
                cbr     shiftbuf,USBpinmask     ;unmask low 2 bits
                breq    USBloopEnd              ;if they are zeros - end of USB packet
                dec     bitcount                ;decrement bits counter
                nop                             ;
                dec     ByteCount               ;if not reached maximum buffer
                brne    USBloop1_6              ;then receive next
```

```
                rjmp    EndInt0HandlerPOP     ;otherwise repeat back from begin

USBloopEnd:
                cpi     USBBufptrY,InputShiftBufferBegin+3   ;if at least 3 byte not
received
                brcs    EndInt0HandlerPOP     ;then finish
                lds     temp0,InputShiftBufferBegin+0 ;identifier of packet to temp0
                lds     temp1,InputShiftBufferBegin+1 ;address to temp1
                brne    TestDataPacket        ;if is length different from 3 - then this
can be only DataPaket
TestIOPacket:

                andi    temp1,0xFE            ;MMM mask out bit 0 of address to avoid
conflict with endpoint 1

                cp      temp1,MyAddress               ;if this isn't assigned (address) for
me
                brne    TestDataPacket        ;then this can be still DataPacket
TestSetupPacket:                      ;test to SETUP packet
                cpi     temp0,nNRZISETUPPID
                brne    TestOutPacket         ;if this isn't Setup PID - decode other
packet
                ldi     State,SetupState
                rjmp    EndInt0HandlerPOP     ;if this is Setup PID - receive consecutive
Data packet
TestOutPacket:                        ;test for OUT packet
                cpi     temp0,nNRZIOUTPID
                brne    TestInPacket          ;if this isn't Out PID - decode other packet
                ldi     State,OutState
                rjmp    EndInt0HandlerPOP     ;if this is Out PID - receive consecutive
Data packet
TestInPacket:                         ;test on IN packet
                cpi     temp0,nNRZIINPID
                brne    TestDataPacket        ;if this isn't In PID - decode other packet
                rjmp    AnswerToInRequest
TestDataPacket:                       ;       test for DATA0 and DATA1 packet
                cpi     temp0,nNRZIDATA0PID
                breq    Data0Packet           ;if this isn't Data0 PID - decode other
packet
                cpi     temp0,nNRZIDATA1PID
                brne    NoMyPacked            ;if this isn't Data1 PID - decode other
packet
Data0Packet:
                cpi     State,SetupState      ;if was state Setup
                breq    ReceiveSetupData      ;receive it
                cpi     State,OutState        ;if was state Out
                breq    ReceiveOutData        ;receive it
NoMyPacked:
                ldi     State,BaseState               ;zero state
                rjmp    EndInt0HandlerPOP     ;and receive consecutive Data packet

AnswerToInRequest:
                push    temp2                 ;backup next registers and continue
                push    temp3
                push    R26
                push    ACC


; this might be  Endpoint1 interrupt query
                lds             temp1,InputShiftBufferBegin+1 ;address to temp1
                lds             temp2,InputShiftBufferBegin+2 ;endpoint and CRC to temp2

                ror             temp1 ; move bit 0 to carry
                ror             temp2 ; bring bit 7 to carry
                swap    temp2
                sbrs    temp1, 0       ; check bit 1 (6) of address
                rjmp    AddrBit6Zero
                com             temp2
AddrBit6Zero:
                andi    temp2, 0x0F
                cpi             temp2, 0x0A
```

```
                breq   ProcessEndpoint0
                cpi            temp2, 0x05
                breq   ProcessEndpoint1
                rjmp   EndInt0Handler


ProcessEndpoint0:
                cpi            ActionFlag,DoReadySendAnswer  ;if isn't prepared answer
                brne   NoReadySend           ;then send NAK
                rcall  SendPreparedUSBAnswer ;transmitting answer back
                and            MyUpdatedAddress,MyUpdatedAddress    ;if is
MyUpdatedAddress nonzero
                brne   SetMyNewUSBAddress_2  ;then is necessary to change USB address
                ldi            State,InState
                ldi            ActionFlag,DoPrepareOutContinuousBuffer
                rjmp   EndInt0Handler        ;and repeat - wait for next response from
USB
ReceiveSetupData:
                push   temp2                 ;backup next registers and continue
                push   temp3
                push   R26
                push   ACC
                rcall  SendACK               ;accept Setup Data packet
                rcall  FinishReceiving               ;finish receiving
                ldi            ActionFlag,DoReceiveSetupData
                rjmp   EndInt0Handler
ReceiveOutData:
                push   temp2                 ;backup next registers and continue
                push   temp3
                push   R26
                push   ACC
                cpi            ActionFlag,DoReceiveSetupData ;if is currently in process
command Setup
                breq   NoReadySend           ;then send NAK
                rcall  SendACK               ;accept Out packet
                clr            ActionFlag
                rjmp   EndInt0Handler
NoReadySend:
                rcall  SendNAK               ;still I am not ready to answer
                rjmp   EndInt0Handler        ;and repeat - wait for next response from
USB




SetMyNewUSBAddress_2:
                rjmp SetMyNewUSBAddress


;-- ENDPOINT 1

ProcessEndpoint1:            ; on Endpoint1 In we have interrupt handler which is
sending reports of joystick data


        ; Check if we have joystick data ready
                sbrs   JoystickFlags, JoystickDataReadyBit
                rjmp   NoJoystickDataReady

        ; Backup Control pipe buffer pointers to save Control pipe state
                mov    temp0, OutputBufferLength
                sts BkpOutputBufferLength,temp0

                mov temp0, OutBitStuffNumber
                sts BkpOutBitStuffNumber,temp0

        ; Retrieve Joystick buffer parameters
                lds    temp0,JoyOutBitStuffNumber
                mov    OutBitStuffNumber, temp0
```

```
		lds		ByteCount,JoyOutputBufferLength		;length of
answer
		ldi		USBBufptrY,JoystickBufferBegin		;pointer to
begin of transmitting buffer
		rcall	SendUSBBuffer


	; Restore Control pipe buffer pointers
		lds		temp0, BkpOutputBufferLength
		mov		OutputBufferLength, temp0
		lds		temp0, BkpOutBitStuffNumber
		mov		 OutBitStuffNumber,temp0

		; flip data PID
		lds		temp0, JoystickBufferBegin + 1
		cpi		temp0, DATA0PID
		breq	FlipToDATA1PID
		ldi		temp0, DATA0PID
		rjmp	FlipDone
FlipToDATA1PID:
		ldi		temp0, DATA1PID
FlipDone:
		sts		JoystickBufferBegin + 1,temp0

		ldi		temp0,JoystickDataRequest
		or		JoystickFlags,temp0 ;JoystickDataRequest	; request new
joystick data

		rjmp	EndInt0Handler		;and complete

NoJoystickDataReady:
	;	rcall	SendNAK			;still I am not ready to answer
		rjmp	EndInt0Handler		;and repeat - wait for next response from
USB


;--END--ENDPOINT1






;------------------------------------------------------------------------------------------
--
SetMyNewUSBAddress:		;set new USB address in NRZI coded
		clr	MyAddress		;original answer state - of my nNRZI USB
address
		ldi	temp2,0b00000001	;mask for xoring
		ldi	temp3,8			;bits counter
SetMyNewUSBAddressLoop:
		mov	temp0,MyAddress			;remember final answer
		ror	MyUpdatedAddress	;to carry transmitting bit LSB (in direction
firstly LSB then MSB)
		brcs	NoXORBit		;if one - don't change state
		eor	temp0,temp2		;otherwise state will be changed according
to last bit of answer
NoXORBit:
		ror	temp0			;last bit of changed answer to carry
		rol	MyAddress		;and from carry to final answer to the LSB
place (and reverse LSB and MSB order)
		dec	temp3			;decrement bits counter
		brne	SetMyNewUSBAddressLoop	;if bits counter isn't zero repeat
transmitting with next bit
		clr	MyUpdatedAddress	;zero addresses as flag of its next
unchanging

		; mask out bit 0 to avoid conflict with endpoints
		mov	temp2, MyAddress
		andi	temp2,0xFE
```

```
                mov MyAddress, temp2

                rjmp    EndInt0Handler
;------------------------------------------------------------------------------------------
--
FinishReceiving:              ;corrective actions for receive termination
                cpi     bitcount,7            ;transfer to buffer also last not completed
byte
                breq    NoRemainingBits               ;if were all bytes transfered, then
nothing transfer
                inc     bitcount
ShiftRemainingBits:
                rol     shiftbuf              ;shift remaining not completed bits on right
position
                dec     bitcount
                brne    ShiftRemainingBits
                st      Y+,shiftbuf           ;and copy shift register bo buffer - not
completed byte
NoRemainingBits:
                mov     ByteCount,USBBufptrY
                subi    ByteCount,InputShiftBufferBegin-1     ;in ByteCount is number of
received bytes (including not completed bytes)

                mov     InputBufferLength,ByteCount           ;and save for use in main
program
                ldi     USBBufptrY,InputShiftBufferBegin      ;pointer to begin of
receiving shift buffer
                ldi     R26,InputBufferBegin+1        ;data buffer (leave out SOP)
                push    XH                                    ;save RS232BufptrX Hi index
                clr     XH
MoveDataBuffer:
                ld      temp0,Y+
                st      X+,temp0
                dec     ByteCount
                brne    MoveDataBuffer

                pop     XH                                    ;restore RS232BufptrX Hi
index
                ldi     ByteCount,nNRZISOPbyte
                sts     InputBufferBegin,ByteCount            ;like received SOP - it is
not copied from shift buffer
                ret
;------------------------------------------------------------------------------------------
--
;*********************************************************************
;*  Other procedures
;*********************************************************************
;------------------------------------------------------------------------------------------
--
USBReset:               ;iinitialization of USB state engine
                ldi     temp0,nNRZIADDR0      ;initialization of USB address
                mov     MyAddress,temp0
                clr     State                 ;initialization of state engine
                clr     BitStuffInOut
                clr     OutBitStuffNumber
                clr     ActionFlag
                clr     RAMread               ;will be reading from ROM
                sts     ConfigByte,RAMread    ;unconfigured state
                ret
;------------------------------------------------------------------------------------------
--
SendPreparedUSBAnswer: ;transmitting by NRZI coding OUT buffer with length
OutputBufferLength to USB
                mov     ByteCount,OutputBufferLength          ;length of answer
SendUSBAnswer: ;transmitting by NRZI coding OUT buffer to USB
                ldi     USBBufptrY,OutputBufferBegin          ;pointer to begin of
transmitting buffer
SendUSBBuffer: ;transmitting by NRZI coding given buffer to USB
                ldi     temp1,0               ;incrementing pointer (temporary variable)
                mov     temp3,ByteCount               ;byte counter: temp3 = ByteCount
                ldi     temp2,0b00000011      ;mask for xoring
```

```
               ld      inputbuf,Y+            ;load first byte to inputbuf and increment
pointer to buffer
                                              ;USB as output:
               cbi     outputport,DATAplus    ;down DATAPLUS : idle state of USB port
               sbi     outputport,DATAminus   ;set DATAMINUS : idle state of USB port
               sbi     USBdirection,DATAplus  ;DATAPLUS as output
               sbi     USBdirection,DATAminus ;DATAMINUS as output

               in      temp0,outputport       ;idle state of USB port to temp0
SendUSBAnswerLoop:
               ldi     bitcount,7             ;bits counter
SendUSBAnswerByteLoop:
               nop                            ;delay because timing
               ror     inputbuf               ;to carry transmiting bit (in direction
first LSB then MSB)
               brcs    NoXORSend              ;if that it is one - don't change USB state
               eor     temp0,temp2            ;iotherwise state will be changed
NoXORSend:
               out     outputport,temp0       ;send out to USB
               dec     bitcount               ;decrement bits counter - according to carry
flag
               brne    SendUSBAnswerByteLoop  ; if bits counter isn't zero - repeat
transmiting with next bit
               sbrs    inputbuf,0             ;if is transmiting bit one - don't change
USB state
               eor     temp0,temp2            ;otherwise state will be changed
NoXORSendLSB:
               dec     temp3                  ;decrement bytes counter
               ld      inputbuf,Y+            ;load next byte and increment pointer to
buffer
               out     outputport,temp0       ;transmit to USB
               brne    SendUSBAnswerLoop      ;repeat for all buffer (till temp3=0)

               mov     bitcount,OutBitStuffNumber    ;bits counter for bitstuff
               cpi     bitcount,0             ;if not be needed bitstuff
               breq    ZeroBitStuf
SendUSBAnswerBitstuffLoop:
               ror     inputbuf               ;to carry transmiting bit (in direction
first LSB then MSB)
               brcs    NoXORBitstuffSend      ;if is one - don't change state on USB
               eor     temp0,temp2            ;otherwise state will be changed
NoXORBitstuffSend:
               out     outputport,temp0       ;transmit to USB
               nop                            ;delay because of timing
               dec     bitcount               ;decrement bits counter - according to carry
flag
               brne    SendUSBAnswerBitstuffLoop     ;if bits counter isn't zero - repeat
transmiting with next bit
               ld      inputbuf,Y             ;delay 2 cycle
ZeroBitStuf:
               nop                            ;delay 1 cycle
               cbr     temp0,3
               out     outputport,temp0       ;transmit EOP on USB

               ldi     bitcount,5             ;delay counter: EOP shouls exists 2 bits (16
cycle at 12MHz)
SendUSBWaitEOP:
               dec     bitcount
               brne    SendUSBWaitEOP

               sbi     outputport,DATAminus   ;set DATAMINUS : idle state on USB port
               sbi     outputport,DATAminus   ;delay 2 cycle: Idle should exists 1 bit (8
cycle at 12MHz)
               cbi     USBdirection,DATAplus  ;DATAPLUS as input
               cbi     USBdirection,DATAminus ;DATAMINUS as input
               cbi     outputport,DATAminus   ;reset DATAMINUS : the third state on USB
port
               ret
;------------------------------------------------------------------------------------------
--
ToggleDATAPID:
```

```
                lds     temp0,OutputBufferBegin+1     ;load last PID
                cpi     temp0,DATA1PID                ;if last was DATA1PID byte
                ldi     temp0,DATA0PID
                breq    SendData0PID                  ;then send zero answer with DATA0PID
                ldi     temp0,DATA1PID                ;otherwise send zero answer with
DATA1PID
SendData0PID:
                sts     OutputBufferBegin+1,temp0     ;DATA0PID byte
                ret
;------------------------------------------------------------------------------------------
--
ComposeZeroDATA1PIDAnswer:
                ldi     temp0,DATA0PID                ;DATA0 PID - in the next will be
toggled to DATA1PID in load descriptor
                sts     OutputBufferBegin+1,temp0     ;load to output buffer
ComposeZeroAnswer:
                ldi     temp0,SOPbyte
                sts     OutputBufferBegin+0,temp0     ;SOP byte
                rcall   ToggleDATAPID                 ;change DATAPID
                ldi     temp0,0x00
                sts     OutputBufferBegin+2,temp0     ;CRC byte
                sts     OutputBufferBegin+3,temp0     ;CRC byte
                ldi     ByteCount,2+2                 ;length of output buffer (SOP and PID
+ CRC16)
                ret
;------------------------------------------------------------------------------------------
--

;------------------------------------------------------------------------------------------
--
InitJoystickBufffer:

;               ldi     ZH, high(JoystickAnswer<<1)   ;ROMpointer to answer
;               ldi     ZL,  low(JoystickAnswer<<1)
                ldi     temp0,JoystickReport1Size                          ;number of my
bytes answers to temp0

                mov TotalBytesToSend, temp0

                ldi     temp0,SOPbyte
                sts     OutputBufferBegin+0,temp0             ;SOP byte
                ldi     temp0,DATA0PID
                sts     OutputBufferBegin+1,temp0             ;DATA0PID byte


                mov     temp3,TotalBytesToSend ;otherwise send only given number of bytes
                mov ByteCount,TotalBytesToSend;

                ldi     USBBufptrY,OutputBufferBegin+2       ;to transmitting buffer
LoadDescriptorFromROM_2:
                lpm                           ;load from ROM position pointer to R0 <-
(ZH:ZL)
                st      Y+,R0                 ;R0 save to buffer and increment buffer (Y)
<- R0, Y++
                adiw    ZH:ZL,1               ;increment index to ROM ; Z++
                dec     ByteCount             ;till are not all bytes
                brne    LoadDescriptorFromROM_2       ;then load next

                ldi     ByteCount,2           ;length of output buffer (only SOP and PID)
                add     ByteCount,temp3               ;+ number of bytes
        rcall   AddCRCOut             ;addition of CRC to buffer
                inc     ByteCount             ;length of output buffer + CRC16
                inc     ByteCount


                inc     BitStuffInOut                 ;transmitting buffer - insertion of
bitstuff bits
                ldi     USBBufptrY,OutputBufferBegin  ;to transmitting buffer
        rcall   BitStuff
                mov     OutputBufferLength,ByteCount  ;length of answer store for
transmiting
```

```
                clr     BitStuffInOut                  ;receiving buffer - deletion of
bitstuff bits



        ; copy to Joystick buffer
                sts JoyOutputBufferLength, OutputBufferLength
                sts JoyOutBitStuffNumber, OutBitStuffNumber

                mov     ByteCount, OutputBufferLength
        ;       ldi     ZH, high(OutputBufferBegin<<1)       ;pointer to answer
                clr ZH
                ldi     ZL,  OutputBufferBegin
                ldi     USBBufptrY,JoystickBufferBegin       ;to transmitting buffer
CopyToJoyBufferLoop:
                ld      R0, Z+                         ;load from RAM position pointer to R0
<- (ZH:ZL)
                st      Y+,R0                  ;R0 save to buffer and increment buffer (Y)
<- R0, Y++
;               adiw    ZH:ZL,1                ;increment index to ROM ; Z++
                dec     ByteCount              ;till are not all bytes
                brne    CopyToJoyBufferLoop    ;then load next

                ret
;------------------------------------------------------------------------------------------
--
InitACKBufffer:
                ldi     temp0,SOPbyte
                sts     ACKBufferBegin+0,temp0        ;SOP byte
                ldi     temp0,ACKPID
                sts     ACKBufferBegin+1,temp0        ;ACKPID byte
                ret
;------------------------------------------------------------------------------------------
--
SendACK:
                push    USBBufptrY
                push    bitcount
                push    OutBitStuffNumber
                ldi     USBBufptrY,ACKBufferBegin     ;pointer to begin of ACK buffer
                ldi     ByteCount,2                   ;number of transmit bytes (only SOP
and ACKPID)
                clr     OutBitStuffNumber
                rcall   SendUSBBuffer
                pop     OutBitStuffNumber
                pop     bitcount
                pop     USBBufptrY
                ret
;------------------------------------------------------------------------------------------
--
InitNAKBufffer:
                ldi     temp0,SOPbyte
                sts     NAKBufferBegin+0,temp0        ;SOP byte
                ldi     temp0,NAKPID
                sts     NAKBufferBegin+1,temp0        ;NAKPID byte
                ret
;------------------------------------------------------------------------------------------
--
SendNAK:
                push    OutBitStuffNumber
                ldi     USBBufptrY,NAKBufferBegin     ;pointer to begin of NACK buffer
                ldi     ByteCount,2                   ;number of transmited bytes (only SOP
and NAKPID)
                clr     OutBitStuffNumber
                rcall   SendUSBBuffer
                pop     OutBitStuffNumber
                ret
;------------------------------------------------------------------------------------------
--
ComposeSTALL:
                ldi     temp0,SOPbyte
                sts     OutputBufferBegin+0,temp0     ;SOP byte
```

```
               ldi     temp0,STALLPID
               sts     OutputBufferBegin+1,temp0     ;STALLPID byte
               ldi     ByteCount,2                   ;length of output buffer (SOP and
PID)
               ret
;------------------------------------------------------------------------------------
--
DecodeNRZI:    ;encoding of buffer from NRZI code to binary
               push    USBBufptrY            ;back up pointer to buffer
               push    ByteCount             ;back up length of buffer
               add     ByteCount,USBBufptrY  ;end of buffer to ByteCount
               ser     temp0                 ;to ensure unit carry (in the next rotation)
NRZIloop:
               ror     temp0                 ;filling carry from previous byte
               ld      temp0,Y               ;nload received byte from buffer
               mov     temp2,temp0           ;shifted register to one bit to the right
and XOR for function of NRZI decoding
               ror     temp2                 ;carry to most significant digit bit and
shift
               eor     temp2,temp0           ;NRZI decoding
               com     temp2                 ;negate
               st      Y+,temp2              ;save back as decoded byte and increment
pointer to buffer
               cp      USBBufptrY,ByteCount  ;if not all bytes
               brne    NRZIloop              ;then repeat
               pop     ByteCount             ;restore buffer length
               pop     USBBufptrY            ;restore pointer to buffer
               ret                           ;otherwise finish
;------------------------------------------------------------------------------------
--
BitStuff:      ;removal of bitstuffing in buffer
               clr     temp3                 ;counter of omitted bits
               clr     lastBitstufNumber     ;0xFF to lastBitstufNumber
               dec     lastBitstufNumber
BitStuffRepeat:
               push    USBBufptrY            ;back up pointer to buffer
               push    ByteCount             ;back up buffer length
               mov     temp1,temp3           ;counter of all bits
               ldi     temp0,8               ;sum all bits in buffer
SumAllBits:
               add     temp1,temp0
               dec     ByteCount
               brne    SumAllBits
               ldi     temp2,6               ;initialize counter of ones
               pop     ByteCount             ;restore buffer length
               push    ByteCount             ;back up buffer length
               add     ByteCount,USBBufptrY  ;end of buffer to ByteCount
               inc     ByteCount             ;and for safety increment it with 2 (because
of shifting)
               inc     ByteCount
BitStuffLoop:
               ld      temp0,Y               ;load received byte from buffer
               ldi     bitcount,8            ;bits counter in byte
BitStuffByteLoop:
               ror     temp0                 ;filling carry from LSB
               brcs    IncrementBitstuff     ;if that LSB=0
               ldi     temp2,7               ;initialize counter of ones +1 (if was zero)
IncrementBitstuff:
               dec     temp2                 ;decrement counter of ones (assumption of
one bit)
               brne    NeposunBuffer         ;if there was not 6 ones together - don't
shift buffer
               cp      temp1,lastBitstufNumber       ;
               ldi     temp2,6               ;initialize counter of ones (if no
bitstuffing will be made then must be started again)
               brcc    NeposunBuffer         ;if already was made bitstuffing - don't
shift buffer

               dec     temp1   ;
               mov     lastBitstufNumber,temp1       ;remember last position of
bitstuffing
```

```
                cpi     bitcount,1              ;for pointing to 7-th bit (which must be
deleted or where to insert zero)
                brne    NoBitcountCorrect
                ldi     bitcount,9      ;
                inc     USBBufptrY              ;increment pointer to buffer
NoBitcountCorrect:
                dec     bitcount
                bst     BitStuffInOut,0
                brts    CorrectOutBuffer        ;if this is Out buffer - increment buffer
length
                rcall   ShiftDeleteBuffer       ;shift In buffer
                dec     temp3                   ;decrement counter of omission
                rjmp    CorrectBufferEnd
CorrectOutBuffer:
                rcall   ShiftInsertBuffer       ;shift Out buffer
                inc     temp3                   ;increment counter of omission
CorrectBufferEnd:
                pop     ByteCount               ;restore buffer length
                pop     USBBufptrY              ;restore pointer to buffer
                rjmp    BitStuffRepeat          ;and restart from begin
NeposunBuffer:
                dec     temp1                   ;if already were all bits
                breq    EndBitStuff             ;finish cycle
                dec     bitcount                ;decrement bits counter in byte
                brne    BitStuffByteLoop        ;if not yet been all bits in byte - go to
next bit
                                                ;otherwise load next byte
                inc     USBBufptrY              ;increment pointer to buffer
                rjmp    BitStuffLoop            ;and repeat
EndBitStuff:
                pop     ByteCount               ;restore buffer length
                pop     USBBufptrY              ;restore pointer to buffer
                bst     BitStuffInOut,0
                brts    IncrementLength                 ;if this is Out buffer - increment
length of Out buffer
DecrementLength:                                ;if this is In buffer - decrement length of
In buffer
                cpi     temp3,0                 ;was at least one decrement
                breq    NoChangeByteCount       ;if no - don't change buffer length
                dec     ByteCount               ;if this is In buffer - decrement buffer
length
                subi    temp3,256-8             ;if there wasn't above 8 bits over
                brcc    NoChangeByteCount       ;then finish
                dec     ByteCount               ;otherwise next decrement buffer length
                ret                             ;and finish
IncrementLength:
                mov     OutBitStuffNumber,temp3         ;remember number of bits over
                subi    temp3,8                 ;if there wasn't above 8 bits over
                brcs    NoChangeByteCount       ;then finish
                inc     ByteCount               ;otherwise increment buffer length
                mov     OutBitStuffNumber,temp3         ;and remember number of bits over
(decremented by 8)
NoChangeByteCount:
                ret                             ;finish
;------------------------------------------------------------------------------------------
--
ShiftInsertBuffer:      ;shift buffer by one bit to right from end till to position: byte-
USBBufptrY and bit-bitcount
                mov     temp0,bitcount          ;calculation: bitcount= 9-bitcount
                ldi     bitcount,9
                sub     bitcount,temp0          ;to bitcount bit position, which is
necessary to clear

                ld      temp1,Y                 ;load byte which still must be shifted from
position bitcount
                rol     temp1                   ;and shift to the left through Carry
(transmission from higher byte and LSB to Carry)
                ser     temp2                   ;FF to mask - temp2
HalfInsertPosuvMask:
                lsl     temp2                   ;zero to the next low bit of mask
```

```
                dec     bitcount               ;till not reached boundary of shifting in
byte
                brne    HalfInsertPosuvMask

                and     temp1,temp2            ;unmask that remains only high shifted bits
in temp1
                com     temp2                  ;invert mask
                lsr     temp2                  ;shift mask to the right - for insertion of
zero bit
                ld      temp0,Y                ;load byte which must be shifted from
position bitcount to temp0
                and     temp0,temp2            ;unmask to remains only low non-shifted bits
in temp0
                or      temp1,temp0            ;and put together shifted and nonshifted
part

                ld      temp0,Y                ;load byte which must be shifted from
position bitcount
                rol     temp0                  ;and shift it to the left through Carry (to
set right Carry for further carry)
                st      Y+,temp1               ;and load back modified byte
ShiftInsertBufferLoop:
                cpse    USBBufptrY,ByteCount   ;if are not all entire bytes
                rjmp    NoEndShiftInsertBuffer ;then continue
                ret                            ;otherwise finish
NoEndShiftInsertBuffer:
                ld      temp1,Y                ;load byte
                rol     temp1                  ;and shift to the left through Carry (carry
from low byte and LSB to Carry)
                st      Y+,temp1               ;and store back
                rjmp    ShiftInsertBufferLoop  ;and continue
;------------------------------------------------------------------------------------------
--
ShiftDeleteBuffer:     ;shift buffer one bit to the left from end to position: byte-
USBBufptrY and bit-bitcount
                mov     temp0,bitcount         ;calculation: bitcount= 9-bitcount
                ldi     bitcount,9
                sub     bitcount,temp0         ;to bitcount bit position, which must be
shifted
                mov     temp0,USBBufptrY       ;backup pointera to buffer
                inc     temp0                  ;position of completed bytes to temp0
                mov     USBBufptrY,ByteCount   ;maximum position to pointer
ShiftDeleteBufferLoop:
                ld      temp1,-Y               ;decrement buffer and load byte
                ror     temp1                  ;and right shift through Carry (carry from
higher byte and LSB to Carry)
                st      Y,temp1                ;and store back
                cpse    USBBufptrY,temp0       ;if there are not all entire bytes
                rjmp    ShiftDeleteBufferLoop  ;then continue

                ld      temp1,-Y               ;decrement buffer and load byte which must
be shifted from position bitcount
                ror     temp1                  ;and right shift through Carry (carry from
higher byte and LSB to Carry)
                ser     temp2                  ;FF to mask - temp2
HalfDeletePosuvMask:
                dec     bitcount               ;till not reached boundary of shifting in
byte
                breq    DoneMask
                lsl     temp2                  ;zero to the next low bit of mask
                rjmp    HalfDeletePosuvMask
DoneMask:
                and     temp1,temp2            ;unmask to remain only high shifted bits in
temp1
                com     temp2                  ;invert mask
                ld      temp0,Y                ;load byte which must be shifted from
position bitcount to temp0
                and     temp0,temp2            ;unmask to remain only low nonshifted bits
in temp0
                or      temp1,temp0            ;and put together shifted and nonshifted
part
```

```
                st      Y,temp1                 ;and store back
                ret                             ;and finish
;------------------------------------------------------------------------------------------
--
MirrorInBufferBytes:
                push    USBBufptrY
                push    ByteCount
                ldi     USBBufptrY,InputBufferBegin
                rcall   MirrorBufferBytes
                pop     ByteCount
                pop     USBBufptrY
                ret
;------------------------------------------------------------------------------------------
--
MirrorBufferBytes:
                add     ByteCount,USBBufptrY    ;ByteCount shows to the end of message
MirrorBufferloop:
                ld      temp0,Y                 ;load received byte from buffer
                ldi     temp1,8                 ;bits counter
MirrorBufferByteLoop:
                ror     temp0                   ;to carry next least bit
                rol     temp2                   ;from carry next bit to reverse order
                dec     temp1                   ;was already entire byte
                brne    MirrorBufferByteLoop    ;if no then repeat next least bit
                st      Y+,temp2                ;save back as reversed byte  and increment
pointer to buffer
                cp      USBBufptrY,ByteCount    ;if not yet been all
                brne    MirrorBufferloop        ;then repeat
                ret                             ;otherwise finish
;------------------------------------------------------------------------------------------
--
;CheckCRCIn:
;               push    USBBufptrY      ;
;               push    ByteCount       ;
;               ldi     USBBufptrY,InputBuffercompare ;
;               rcall   CheckCRC        ;
;               pop     ByteCount       ;
;               pop     USBBufptrY      ;
;               ret     ;
;------------------------------------------------------------------------------------------
--

AddCRCOut:
                push    USBBufptrY
                push    ByteCount
                ldi     USBBufptrY,OutputBufferBegin
AddCRCOut_2:
                rcall   CheckCRC
                com     temp0                   ;negation of CRC
                com     temp1
                st      Y+,temp1                ;save CRC to the end of buffer (at first
MSB)
                st      Y,temp0                 ;save CRC to the end of buffer (then LSB)
                dec     USBBufptrY              ;pointer to CRC position
                ldi     ByteCount,2             ;reverse bits order in 2 bytes CRC
                rcall   MirrorBufferBytes       ;reverse bits order in CRC (transmiting CRC
- MSB first)
                pop     ByteCount
                pop     USBBufptrY
                ret
;------------------------------------------------------------------------------------------
--
CheckCRC:       ;input: USBBufptrY = begin of message ,ByteCount = length of message
                add     ByteCount,USBBufptrY    ;ByteCount points to the end of message
                inc     USBBufptrY              ;set the pointer to message start - omit SOP
                ld      temp0,Y+                ;load PID to temp0
                                                ;and set the pointer to start of message -
omit also PID
                cpi     temp0,DATA0PID          ;if is DATA0 field
                breq    ComputeDATACRC          ;compute CRC16
                cpi     temp0,DATA1PID          ;if is DATA1 field
```

```
               brne    CRC16End              ;if no then finish
ComputeDATACRC:
               ser     temp0                 ;initialization of remaider LSB to 0xff
               ser     temp1                 ;initialization of remaider MSB to 0xff
CRC16Loop:
               ld      temp2,Y+              ;load message to temp2 and increment pointer
to buffer
               ldi     temp3,8               ;bits counter in byte - temp3
CRC16LoopByte:
               bst     temp1,7               ;to T save MSB of remainder (remainder is
only 16 bits - 8 bit of higher byte)
               bld     bitcount,0            ;to bitcount LSB save T - of MSB remainder
               eor     bitcount,temp2        ;XOR of bit message and bit remainder - in
LSB bitcount
               rol     temp0                 ;shift remainder to the left - low byte (two
bytes - through carry)
               rol     temp1                 ;shift remainder to the left - high byte
(two bytes - through carry)
               cbr     temp0,1               ;znuluj LSB remains
               lsr     temp2                 ;shift message to right
               ror     bitcount              ;result of XOR bits from LSB to carry
               brcc    CRC16NoXOR            ;if is XOR bitmessage and MSB of remainder =
0 , then no XOR
               ldi     bitcount,CRC16poly>>8 ;to bitcount CRC polynomial - high byte
               eor     temp1,bitcount        ;and make XOR from remains and CRC
polynomial - high byte
               ldi     bitcount,low(CRC16poly)       ;to bitcount CRC polynomial - low
byte
               eor     temp0,bitcount        ;and make XOR of remainder and CRC
polynomial - low byte
CRC16NoXOR:
               dec     temp3                 ;were already all bits in byte
               brne    CRC16LoopByte         ;unless, then go to next bit
               cp      USBBufptrY,ByteCount  ;was already end-of-message
               brne    CRC16Loop             ;unless then repeat
CRC16End:
               ret                           ;otherwise finish (in temp0 and temp1 is
result)
;------------------------------------------------------------------------------------------
--
LoadDescriptorFromROM:
               lpm                           ;load from ROM position pointer to R0
               st      Y+,R0                 ;R0 save to buffer and increment buffer
               adiw    ZH:ZL,1               ;increment index to ROM
               dec     ByteCount             ;till are not all bytes
               brne    LoadDescriptorFromROM ;then load next
               rjmp    EndFromRAMROM         ;otherwise finish
;------------------------------------------------------------------------------------------
--
LoadDescriptorFromROMZeroInsert:
               lpm                           ;load from ROM position pointer to R0
               st      Y+,R0                 ;R0 save to buffer and increment buffer

               bst     RAMread,3             ;if bit 3 is one - don't insert zero
               brtc    InsertingZero         ;otherwise zero will be inserted
               adiw    ZH:ZL,1               ;increment index to ROM
               lpm                           ;load from ROM position pointer to R0
               st      Y+,R0                 ;R0 save to buffer and increment buffer
               clt                           ;and clear
               bld     RAMread,3             ;the third bit in RAMread - for to the next
zero insertion will be made
               rjmp    InsertingZeroEnd      ;and continue
InsertingZero:
               clr     R0                    ;for insertion of zero
               st      Y+,R0                 ;zero save to buffer and increment buffer
InsertingZeroEnd:
               adiw    ZH:ZL,1               ;increment index to ROM
               subi    ByteCount,2           ;till are not all bytes
               brne    LoadDescriptorFromROMZeroInsert       ;then load next
               rjmp    EndFromRAMROM         ;otherwise finish
```

```
;------------------------------------------------------------------------------------------
--
LoadDescriptorFromSRAM:
               ld     R0,Z                  ;load from position RAM pointer to R0
               st     Y+,R0                 ;R0 save to buffer and increment buffer
               adiw   ZH:ZL,1               ;increment index to RAM
               dec    ByteCount             ;till are not all bytes
               brne   LoadDescriptorFromSRAM;then load next
               rjmp   EndFromRAMROM         ;otherwise finish
;------------------------------------------------------------------------------------------
--
LoadDescriptorFromEEPROM:
               out    EEARL,ZL              ;set the address EEPROM Lo
               out    EEARH,ZH              ;set the address EEPROM Hi
               sbi    EECR,EERE             ;read EEPROM to register EEDR
               in     R0,EEDR               ;load from EEDR to R0
               st     Y+,R0                 ;R0 save to buffer and increment buffer
               adiw   ZH:ZL,1               ;increment index to EEPROM
               dec    ByteCount             ;till are not all bytes
               brne   LoadDescriptorFromEEPROM;then load next
               rjmp   EndFromRAMROM         ;otherwise finish
;------------------------------------------------------------------------------------------
--
LoadXXXDescriptor:
               ldi    temp0,SOPbyte                 ;SOP byte
               sts    OutputBufferBegin,temp0               ;to begin of tramsmiting
buffer store SOP
               ldi    ByteCount,8                   ;8 byte store
               ldi    USBBufptrY,OutputBufferBegin+2        ;to transmitting buffer

               and    RAMread,RAMread                       ;if will be reading from RAM
or ROM or EEPROM
               brne   FromRAMorEEPROM                       ;0=ROM,1=RAM,2=EEPROM,4=ROM
with zero insertion (string)
FromROM:
               rjmp   LoadDescriptorFromROM         ;load descriptor from ROM
FromRAMorEEPROM:
               sbrc   RAMread,2                     ;if RAMREAD=4
               rjmp   LoadDescriptorFromROMZeroInsert       ;read from ROM with zero
insertion
               sbrc   RAMread,0                     ;if RAMREAD=1
               rjmp   LoadDescriptorFromSRAM        ;load data from SRAM
               rjmp   LoadDescriptorFromEEPROM      ;otherwise read from EEPROM
EndFromRAMROM:
               sbrc   RAMread,7                     ;if is most significant bit in
variable RAMread=1
               clr    RAMread                       ;clear RAMread
               rcall  ToggleDATAPID                 ;change DATAPID
               ldi    USBBufptrY,OutputBufferBegin+1        ;to transmitting buffer -
position of DATA PID
               ret
;------------------------------------------------------------------------------------------
--
PrepareUSBOutAnswer:   ;prepare answer to buffer
               rcall  PrepareUSBAnswer              ;prepare answer to buffer
MakeOutBitStuff:
               inc    BitStuffInOut                 ;transmitting buffer - insertion of
bitstuff bits
               ldi    USBBufptrY,OutputBufferBegin  ;to transmitting buffer
               rcall  BitStuff
               mov    OutputBufferLength,ByteCount  ;length of answer store for
transmiting
               clr    BitStuffInOut                 ;receiving buffer - deletion of
bitstuff bits
               ret
```

```
;------------------------------------------------------------------------------------------
--
PrepareUSBAnswer:     ;prepare answer to buffer
              clr     RAMread                        ;zero to RAMread variable - reading
from ROM
              lds     temp0,InputBufferBegin+2       ;bmRequestType to temp0
              lds     temp1,InputBufferBegin+3       ;bRequest to temp1
              cbr     temp0,0b10011111               ;if is 5 and 6 bit zero
              brne    CheckVendor                    ;then this isn't  Vendor Request
              rjmp    StandardRequest                       ;but this is standard Request
CheckVendor:
              cpi            temp0, 0b01000000
              breq    VendorRequest
              cpi     temp0, 0b00100000
              breq    ClassRequest

              rjmp    VendorRequest
;--------------------------


;----------------------------------
ClassRequest:
              cpi     temp1,GET_REPORT               ;
              breq    ComposeGET_REPORT              ;

              cpi     temp1,GET_IDLE         ;
              breq    ComposeGET_IDLE                ;

              cpi     temp1,GET_PROTOCOL             ;
              breq    ComposeGET_PROTOCOL            ;

              cpi     temp1,SET_REPORT               ;
              breq    ComposeSET_REPORT              ;

              cpi     temp1,SET_IDLE         ;
              breq    ComposeSET_IDLE                ;

              cpi     temp1,SET_PROTOCOL             ;
              breq    ComposeSET_PROTOCOL            ;

              rjmp    ZeroDATA1Answer                       ;if that was something
unknown, then prepare zero answer

;--------- Class Requests ------------------

ComposeGET_REPORT:                           ;
              rjmp    ZeroDATA1Answer
ComposeGET_IDLE:
              rjmp    ZeroDATA1Answer
ComposeGET_PROTOCOL:
              rjmp    ZeroDATA1Answer
ComposeSET_REPORT:
              rjmp    ZeroDATA1Answer
ComposeSET_IDLE:
              rjmp    ZeroDATA1Answer
ComposeSET_PROTOCOL:
              rjmp    ZeroDATA1Answer




;-----------------------------------
VendorRequest:

              rjmp    ZeroDATA1Answer                       ;if that it was something
unknown, then prepare zero answer


;----------------------------- USER FUNCTIONS --------------------------------------
```

```
;---------------------------- END USER FUNCTIONS ------------------------------------
END USER FUNCTIONS -----------------------------

OneZeroAnswer:		;send single zero
		ldi	temp0,1				;number of my bytes answers to temp0
		rjmp	ComposeGET_STATUS2




;---------------------------- STANDARD USB REQUESTS ---------------------------------
STANDARD USB REQUESTS ------------------------------
StandardRequest:
		cpi	temp1,GET_STATUS		;
		breq	ComposeGET_STATUS		;

		cpi	temp1,CLEAR_FEATURE		;
		breq	ComposeCLEAR_FEATURE		;

		cpi	temp1,SET_FEATURE		;
		breq	ComposeSET_FEATURE		;

		cpi	temp1,SET_ADDRESS		;if to set address
		breq	ComposeSET_ADDRESS		;set the address

		cpi	temp1,GET_DESCRIPTOR		;if requested descriptor
		breq	ComposeGET_DESCRIPTOR		;generate it

		cpi	temp1,SET_DESCRIPTOR		;
		breq	ComposeSET_DESCRIPTOR		;

		cpi	temp1,GET_CONFIGURATION			;
		breq	ComposeGET_CONFIGURATION	;

		cpi	temp1,SET_CONFIGURATION			;
		breq	ComposeSET_CONFIGURATION	;

		cpi	temp1,GET_INTERFACE		;
		breq	ComposeGET_INTERFACE		;

		cpi	temp1,SET_INTERFACE		;
		breq	ComposeSET_INTERFACE		;

		cpi	temp1,SYNCH_FRAME		;
		breq	ComposeSYNCH_FRAME		;
						;if not found known request
		rjmp	ZeroDATA1Answer				;if that was something
unknown, then prepare zero answer

ComposeSET_ADDRESS:
		lds	MyUpdatedAddress,InputBufferBegin+4	;new address to
MyUpdatedAddress
		rjmp	ZeroDATA1Answer				;send zero answer

ComposeSET_CONFIGURATION:

		lds	temp0,InputBufferBegin+4	;number of configuration to variable
ConfigByte
		sts	ConfigByte,temp0		;
ComposeCLEAR_FEATURE:
ComposeSET_FEATURE:
ComposeSET_INTERFACE:
ZeroStringAnswer:
		rjmp	ZeroDATA1Answer				;send zero answer
ComposeGET_STATUS:
TwoZeroAnswer:
		ldi	temp0,2				;number of my bytes answers to temp0
ComposeGET_STATUS2:
		ldi	ZH, high(StatusAnswer<<1)	;ROMpointer to  answer
```

```
                ldi     ZL,  low(StatusAnswer<<1)
                rjmp    ComposeEndXXXDescriptor              ;and complete
ComposeGET_CONFIGURATION:
                lds     temp0,ConfigByte
                ;and    temp0,temp0                  ;if I am unconfigured
                ;breq   OneZeroAnswer                ;then send single zero - otherwise
send my configuration
                ldi     temp0,1                      ;number of my bytes answers to temp0
                ldi     ZH, high(ConfigAnswerMinus1<<1)      ;ROMpointer to  answer
                ldi     ZL,  low(ConfigAnswerMinus1<<1)+1
                rjmp    ComposeEndXXXDescriptor              ;and complete
ComposeGET_INTERFACE:
                ldi     ZH, high(InterfaceAnswer<<1)  ;ROMpointer to answer
                ldi     ZL,  low(InterfaceAnswer<<1)
                ldi     temp0,1                      ;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor              ;and complete
ComposeSYNCH_FRAME:
ComposeSET_DESCRIPTOR:
                rcall   ComposeSTALL
                ret
ComposeGET_DESCRIPTOR:

                                        ; check if we received HID Class Descriptor request
                lds     temp0,InputBufferBegin+2      ;bmRequestType to temp0
                cpi temp0, 0b10000001
                breq    ComposeClassDescriptor
                        ; if not, process standard descriptor requests
                lds     temp1,InputBufferBegin+5      ;DescriptorType to temp1
                cpi     temp1,DEVICE                  ;DeviceDescriptor
                breq    ComposeDeviceDescriptor              ;
                cpi     temp1,CONFIGURATION           ;ConfigurationDescriptor
                breq    ComposeConfigDescriptor              ;
                cpi     temp1,STRING                  ;StringDeviceDescriptor
                breq    ComposeStringDescriptor              ;

                ret
ComposeClassDescriptor:

                lds     temp1,InputBufferBegin+5      ;DescriptorType to temp1
                cpi     temp1,CLASS_HID                      ;HID class descripto
                breq    ComposeHIDClassDescriptor            ;
                cpi     temp1,CLASS_Report            ;ConfigurationDescriptor
                breq    ComposeReportDescriptor              ;
                cpi     temp1,CLASS_Physical                 ;StringDeviceDescriptor
                breq    ComposePhysicalDescriptor
                ret
                                ;
ComposeDeviceDescriptor:
                ldi     ZH, high(DeviceDescriptor<<1) ;ROMpointer to descriptor
                ldi     ZL,  low(DeviceDescriptor<<1)
                ldi     temp0,0x12                    ;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor              ;and completen
ComposeConfigDescriptor:
                ldi     ZH, high(ConfigDescriptor<<1) ;ROMpointer to descriptor
                ldi     ZL,  low(ConfigDescriptor<<1)
                ldi     temp0,9+9+9+7                 ;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor              ;and complete

ComposeHIDClassDescriptor:
                ldi     ZH, high(HIDDescriptor<<1)    ;ROMpointer to descriptor
                ldi     ZL,  low(HIDDescriptor<<1)
                ldi     temp0,9               ;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor              ;and complete

ComposeReportDescriptor         :       ;
                ldi     ZH, high(ReportDescriptor<<1) ;ROMpointer to descriptor
                ldi     ZL,  low(ReportDescriptor<<1)
                ldi     temp0,ReportDescriptorSize                  ;number of my bytes
answers to temp0
                rjmp    ComposeEndXXXDescriptor              ;and complete
```

```
ComposePhysicalDescriptor:
		rjmp	ComposeEndXXXDescriptor		;and complete

ComposeStringDescriptor:
		ldi	temp1,4+8			;if RAMread=4(insert zeros from ROM
reading) + 8(behind first byte no load zero)
		mov	RAMread,temp1
		lds	temp1,InputBufferBegin+4	;DescriptorIndex to temp1
		cpi	temp1,0				;LANGID String
		breq	ComposeLangIDString		;
		cpi	temp1,2				;DevNameString
		breq	ComposeDevNameString
		cpi	temp1,3				;NameString
		breq	ComposeNameString
		cpi	temp1,1				;ComposeVendorString
		breq	ComposeVendorString


		rjmp	ZeroStringAnswer
		rjmp	ZeroDATA1Answer			;if is DescriptorIndex higher than 2
- send zero answer

ComposeVendorString:
		ldi	ZH, high(VendorStringDescriptor<<1)	;ROMpointer to descriptor
		ldi	ZL,  low(VendorStringDescriptor<<1)
		ldi	temp0,(VendorStringDescriptorEnd-VendorStringDescriptor)*4-2
	;number of my bytes answers to temp0
		rjmp	ComposeEndXXXDescriptor		;and complete
ComposeDevNameString:
		ldi	ZH, high(DevNameStringDescriptor<<1)	;ROMpointer to descriptor
		ldi	ZL,  low(DevNameStringDescriptor<<1)
		ldi	temp0,(DevNameStringDescriptorEnd-DevNameStringDescriptor)*4-2
	;number of my bytes answers to temp0
		rjmp	ComposeEndXXXDescriptor		;and complete
ComposeNameString:
		ldi	ZH, high(NameStringDescriptor<<1)	;ROMpointer to descriptor
		ldi	ZL,  low(NameStringDescriptor<<1)
		ldi	temp0,(NameStringDescriptorEnd-NameStringDescriptor)*4-2	;number
of my bytes answers to temp0
		rjmp	ComposeEndXXXDescriptor		;and complete
ComposeLangIDString:
		clr	RAMread
		ldi	ZH, high(LangIDStringDescriptor<<1)	;ROMpointer to descriptor
		ldi	ZL,  low(LangIDStringDescriptor<<1)
		ldi	temp0,(LangIDStringDescriptorEnd-LangIDStringDescriptor)*2;number
of my bytes answers to temp0
		rjmp	ComposeEndXXXDescriptor		;and complete


ComposeEndXXXDescriptor:
		lds	TotalBytesToSend,InputBufferBegin+8	;number of requested bytes to
TotalBytesToSend
		cp	TotalBytesToSend,temp0		;if not requested more than I
can send
		brcs	HostConfigLength		;transmit the requested number
		mov	TotalBytesToSend,temp0		;otherwise send number of my answers
HostConfigLength:
		mov	temp0,TotalBytesToSend		;
		clr	TransmitPart			;zero the number of 8 bytes answers
		andi	temp0,0b00000111		;if is length divisible by 8
		breq	Length8Multiply			;then not count one answer
(under 8 byte)
		inc	TransmitPart			;otherwise count it
Length8Multiply:
		mov	temp0,TotalBytesToSend		;
		lsr	temp0				;length of 8 bytes answers will reach
		lsr	temp0				;integer division by 8
		lsr	temp0
		add	TransmitPart,temp0		;and by addition to last non entire
8-bytes to variable TransmitPart
```

```
                ldi     temp0,DATA0PID                ;DATA0 PID - in the next will be
toggled to DATA1PID in load descriptor
                sts     OutputBufferBegin+1,temp0     ;store to output buffer
                rjmp    ComposeNextAnswerPart


;------------------------------------------------------------------------------------------
--
ZeroDATA1Answer:
                rcall   ComposeZeroDATA1PIDAnswer
                ret
;------------------------------ END USB REQUESTS ------

PrepareOutContinuousBuffer:
                rcall   PrepareContinuousBuffer
                rcall   MakeOutBitStuff
                ret
;------------------------------------------------------------------------------------------
--
PrepareContinuousBuffer:
                mov     temp0,TransmitPart
                cpi     temp0,1
                brne    NextAnswerInBuffer            ;if buffer empty
                rcall   ComposeZeroAnswer             ;prepare zero answer
                ret
NextAnswerInBuffer:
                dec     TransmitPart                  ;decrement general length of answer
ComposeNextAnswerPart:
                mov     temp1,TotalBytesToSend ;decrement number of bytes to transmit
                subi    temp1,8               ;is is necessary to send more as 8 byte
                ldi     temp3,8               ;if yes - send only 8 byte
                brcc    Nad8Bytov
                mov     temp3,TotalBytesToSend ;otherwise send only given number of bytes
                clr     TransmitPart
                inc     TransmitPart          ;and this will be last answer
Nad8Bytov:
                mov     TotalBytesToSend,temp1 ;decremented number of bytes to
TotalBytesToSend
                rcall   LoadXXXDescriptor
                ldi     ByteCount,2           ;length of output buffer (only SOP and PID)
                add     ByteCount,temp3               ;+ number of bytes
                rcall   AddCRCOut             ;addition of CRC to buffer
                inc     ByteCount             ;length of output buffer + CRC16
                inc     ByteCount
                ret                           ;finish




;------------------------------------------------------------------------------------------
--
.equ    USBversion              =0x0100       ;for what version USB is that (1.00)
.equ    VendorUSBID             =0x0003       ;vendor identifier (Atmel=0x03EB) 0x0777
.equ    DeviceUSBID             =0x0001       ;product identifier (USB Joystick)
.equ    DeviceVersion           =0x0001       ;version number of product (version=0.01)
                                              ;(0.01=First USB Joystick with internal ADC)
.equ    MaxUSBCurrent           =0xA0         ;current consumption from USB (50mA) -
together with MAX232
;------------------------------------------------------------------------------------------
--
DeviceDescriptor:
                .db     0x12,0x01             ;0 byte - size of descriptor in byte
                                              ;1 byte - descriptor type: Device descriptor
                .dw     USBversion            ;2,3 byte - version USB LSB (1.00)
                .db     0x00,0x00             ;4 byte - device class
                                              ;5 byte - subclass
                .db     0x00,0x08             ;6 byte - protocol code
                                              ;7 byte - FIFO size in bytes
                .dw     VendorUSBID           ;8,9 byte - vendor identifier
                .dw     DeviceUSBID           ;10,11 byte - product identifier
```

```
               .dw     DeviceVersion          ;12,13 byte - product version number
               .db     0x01,0x02              ;14 byte - index of string "vendor"
                                              ;15 byte - index of string "product"
               .db     0x00,0x01              ;16 byte - index of string "serial number"
(0=none)
                                              ;17 byte - number of possible configurations
DeviceDescriptorEnd:
;------------------------------------------------------------------------------------------
--
ConfigDescriptor:
               .db     0x9,0x02               ;length, descriptor type
ConfigDescriptorLength:
               .dw     9+9+9+7                ;entire length of all descriptors + HID
       ConfigAnswerMinus1:                    ;for sending the number - congiguration
number (attention - addition of 1 required)
               .db     1,1                    ;numInterfaces, congiguration number
               .db     2,0x80                 ;string index (0=none), attributes; bus
powered
;InterfaceDescriptor-1:
               .db     MaxUSBCurrent/2,0x09  ;current consumption,    interface descriptor
length
               .db     0x04,0                 ;interface descriptor; number of interface
       InterfaceAnswer:                       ;for sending number of alternatively
interface
               .db     0,1                    ;alternatively interface; number of
endpoints except EP0
               .db     0x03,0                 ;interface class - HID; interface subclass
               .db     0,3                    ;protocol code; string index - Device name
HIDDescriptor:
               .db 0x09,0x21  ; HID descriptor length , HID descriptor type (defined by
USB)
               .dw 0x101              ; HID Class Specification release number
               .db     0,0x01                 ;Hardware target country.;    ;Number of HID
class descriptors to follow.
               .db     0x22,ReportDescriptorSize                    ;Report descriptor
type.; length LSB
               .db     0, 0x07                ;Total length of Report descriptor MSB,
EndPointDescriptor length
;EndPointDescriptor:
;.db    0x07,0x5              ;length, descriptor type - endpoint
               .db     0x5, 0x81              ;, descriptor type - endpoint
;.db    0x81,0                ;eendpoint address; transfer type
               .db     0x3, 0x08                      ;endpoint address In 1; transfer type
-interrupt;max packet size LSB
;.dw    0x08                  ;max packet size
               .db     0, 10                  ;max packet size MSB,polling interval [ms];
;.db    10,0                  ;polling interval [ms]; dummy byte (for filling)

ConfigDescriptorEnd:
;-------------------------

StatusAnswer:
               .db 0,0        ;2 zero answers


;-----------------------------------

.equ    ReportDescriptorSize =85
.equ    JoystickReportCount =2

.equ    JoystickReport1Size =5
.equ    JoystickReport2Size =4

ReportDescriptor:
               .db 0x05,0x01          ;Usage_Page (Generic Desktop)
               .db 0x15,0x00          ;Logical_Minimum (0)
               .db 0x09,0x04          ;Usage (Joystick)
               .db 0xA1,0x01          ;Collection (Application)
               .db 0x05,0x02                  ;Usage_Page (Simulation Controls)
               .db 0x85,0x01                  ;Report_ID (1)
               .db     0x05,0x01                      ;Usage_Page (Generic Desktop)
```

```
                .db 0x09,0x01                   ;Usage (Pointer)
                .db 0xA1,0x00                   ;Collection (Physical)
                .db 0x09,0x30                           ;Usage (Dial) - OK
                .db 0x15,0x81                   ;Logical_Minimum (-127)
                .db 0x25,0x7F                   ;Logical Maximum (127)
                .db 0x75,0x08                           ;Report_Size (8)
                .db 0x95,0x01                           ;Report_Count (1)
                .db 0x81,0x02                           ;Input (Data, Var, Abs)
                .db 0x09,0x37                           ;Usage (X) - OK
                .db 0x09,0x31                           ;Usage (Y) - OK
                .db 0x16,0x00                   ;Logical_Minimum (-512)
                .db 0xFE,0x26
                .db 0xFF,0x01                   ;Logical Maximum (511)
                .db 0x75,0x0A                           ;Report_Size (10)
                .db 0x95,0x02                           ;Report_Count (2)
                .db 0x81,0x02                           ;Input (Data, Var, Abs)
                .db 0xC0,0x16                           ;End_Collection
                .db 0x01,0xFE                   ;dummy logical minimum
                .db 0x75,0x04                   ;Report_Size (4)
                .db 0x95,0x01                   ;Report_Count (1)
                .db 0x81,0x03                   ;Input (Const, Var, Abs)


                .db 0x85,0x02                   ;Report_ID (2)
                .db 0x05,0x09                   ;Usage_Page (Button)
                .db 0x19,0x01                   ;Usage_Minimum (Button 1)
                .db 0x29,0x18                   ;Usage_Maximum (Button 24)
                .db     0x15,0x00                       ;Logical_Minimum (0)
                .db 0x25,0x01                   ;Logical_Maximum (1)
                .db 0x75,0x01                   ;Report_Size (1)
                .db 0x95,0x02                   ;Report_Count (2)
                .db 0x55,0x00                   ;Unit_Exponent (0)
                .db 0x65,0x00                   ;Unit (None)
                .db 0x81,0x02                   ;Input (Data, Var, Abs)
                .db 0x75,0x16                   ;Report_Size (22)
                .db 0x95,0x01                   ;Report_Count (1)
                .db 0x81,0x03                   ;Input (Const, Var, Abs)
                .db 0xC0,0                      ;End_Collection , dummy padding

ReportDescriptorEnd:


;------------------------------------------------------------------------------------
--
LangIDStringDescriptor:
                .db     (LangIDStringDescriptorEnd-LangIDStringDescriptor)*2,3
        ;length, type: string descriptor
                ;.dw    0x0409                  ;English
                .dw     0x0009                  ;English
LangIDStringDescriptorEnd:
;------------------------------------------------------------------------------------
--
VendorStringDescriptor:
                .db     (VendorStringDescriptorEnd-VendorStringDescriptor)*4-2,3
        ;length, type: string descriptor
CopyRight:
                .db     "Mindaugas Milasauskas (c) 2004, Ing. Igor Cesko, Copyright(c)
2003"
CopyRightEnd:
VendorStringDescriptorEnd:
;------------------------------------------------------------------------------------
--
DevNameStringDescriptor:
                .db     (DevNameStringDescriptorEnd-DevNameStringDescriptor)*4-2,3;length,
type: string descriptor
                .db     "HMTD"
DevNameStringDescriptorEnd:

NameStringDescriptor:
                .db     (NameStringDescriptorEnd-NameStringDescriptor)*4-2,3;length, type:
string descriptor
```

```
                .db     "Head Mounted Tracking Device"
NameStringDescriptorEnd:


;------------------------------------------------------------------------------------------
--
;*********************************************************************
;*  End of program
;*********************************************************************
;------------------------------------------------------------------------------------------
--
;*********************************************************************
;*  EEPROM contents
;*********************************************************************
;------------------------------------------------------------------------------------------
--
.eseg           ;data in EEPROM (at final version comment)
;.org   0x400   ;.org   0x400   ;for filling EEPROM give on right addresses - behind the
program code (at final version uncomment)
EEData:
;------------------------------------------------------------------------------------------
--
;*********************************************************************
;*  End of file
;*********************************************************************
```

Appendix B

```
;***************************************************************************
;*
;* Date              :12.07.2006
;* Version           :1.0
;* Target MCU        :ATmega8
;* AUTHORS           :Adam Thompson, Nick Sorenson
;*
;* DESCRIPTION:
;*  Requests data from address 0x0F from the gyroscope, across the SPI interface, ouputs this
;*    data on port D.
;*
;*
;***************************************************************************
.include "m8def.inc"

.equ  DD_SS              =DDB2
.equ  DD_MOSI            =DDB3
.equ  DD_MISO            =DDB4
.equ  DD_SCK             =DDB5
.equ  DDR_SPI            =DDRB

.equ  RAMEND128          =96+127

.equ  StackBegin         =RAMEND128

.cseg
;-----------------------------------------------------------------------
.org 0                                    ;after reset
            rjmp  reset
;-----------------------------------------------------------------------

;-----------------------------------------------------------------------
;*******************************************************************
;*  Init program
;*******************************************************************
;-----------------------------------------------------------------------
reset:                  ;initialization of processor and variables to
right values
      ldi         r18, 0b11111111   ;set output on PORTD
      out         DDRD,r18


      ldi         r18, StackBegin   ;initialization of stack
      out         SPL,r18

      ;Set SS high
      SBI         PORTB, 2
      rcall SPI_MasterInit


      CBI         PORTB, 2
      ldi         r16, 0x8F
      rcall SPI_MasterTransmit
```

```
	ldi		r16, 0x00
	rcall SPI_MasterTransmit
	SBI		PORTB, 2
;-----------------------------------------------------------------------
;*********************************************************************
;*  Main program
;*********************************************************************
Main:

	;Send something
	;ldi		r16, 0b10000100
	;rcall		SPI_MasterTransmit
	;ldi		r16, 0b10000000
	;rcall		SPI_MasterTransmit
	;ldi		r16, 0x3C


	;rcall		SPI_MasterTransmit
	;CBI		PORTB, 2
	;ldi		r16, 0xBE
	;rcall		SPI_MasterTransmit
	;ldi		r16, 0x01
	;rcall		SPI_MasterTransmit
	;SBI		PORTB, 2


	;rcall		Delay


	;CBI		PORTB, 2
	;ldi		r16, 0xB5
	;rcall		SPI_MasterTransmit
	;ldi		r16, 0x04
	;rcall		SPI_MasterTransmit
	;SBI		PORTB, 2

	rcall Delay
	rcall Delay
	rcall Delay


	CBI		PORTB, 2
	ldi		r16, 0x0F
	rcall SPI_MasterTransmit

	in		r17, SPDR
	out		PORTD, r17


	ldi		r16, 0x0F
	rcall SPI_MasterTransmit
	SBI		PORTB, 2


	in		r18, SPDR

	lsl		r17
```

```
      lsl         r17
      lsl         r17
      lsr         r18
      lsr         r18
      lsr         r18
      lsr         r18
      lsr         r18
      or          r17,r18
      out         PORTD, r17
      rjmp  Main



SPI_MasterInit:
      ;set MOSI, SCK, SS output, all others input
      ldi         r17, (1<<DD_MOSI)|(1<<DD_SCK)|(1<<DD_SS)
      out         DDR_SPI, r17
      ;Enable SPI, Master, set clock rate fck/16
      ldi         r17,
(1<<SPE)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA);|(1<<SPR1)|(1<<SPr17)
      out         SPCR, r17
      ret


SPI_MasterTransmit:
      ;Start transmission of data (r16)
      out         SPDR, r16
Wait_Transmit:
      ;Wait for transmission complete
      sbis  SPSR, SPIF
      rjmp  Wait_Transmit
      ret

Delay:
      ldi         r19, 0xFF
Delayloop:
      dec         r19
      brne  Delayloop

      ret

;------------------------------------------------------------------------------------------
;**********************************************************************
;* End of program
;**********************************************************************
;------------------------------------------------------------------------------------------
;**********************************************************************
;* End of file
;**********************************************************************
```

Appendix C – References

Data Sheets

ATmega8 Microcontroller
http://www.atmel.com/dyn/products/product_card.asp?family_id=607&family_name=AVR+8%2DBit+RISC+&part_id=2004

ADXL203 Accelerometer
http://www.analog.com/en/prod/0%2C2877%2CADXL203%2C00.html

ADIS16250 Programmable Low Power Gyroscope
http://www.analog.com/en/prod/0%2C2877%2CADIS16250%2C00.html

ATSTK500 Starter Kit
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2735

Innovatek V-490 Video Glasses
http://www.innovatek.cn/products/video%20glasses/v-490.htm

Websites

Source of countless articles about all aspects of USB interfacing
http://www.usb.org/home

An example of how a Super Nintendo controller was attached over USB
http://www.raphnet.net/electronique/snes_nes_usb/index_en.php

An example of how to make a USB joystick
http://www.mindaugas.com/projects/MJoy/

Several other USB projects
http://www.cesko.host.sk/

Appendix D – Bill of Materials

Part	Our cost	True Cost	Vendor	Description and Notes
ATSDK500	$75	$75	DigiKey	Software development kit for the ATmega8 microcontrollers. This is an optional item, but is extremely useful.
ATmega8(x2)	$4 each	$4 each	DigiKey	The current software requires two of these. The assembly code could be rewritten to reduce this to one. This may cause the device to poll slower and therefore less accurately.
ADXL203(x2)	Free	$22.50	Analog Devices	These parts could be replaced with a single 3 axis gyroscope or accelerometer. They are extremely accurate, however. Plus, nothing beats free.
ADIS16250	$75	$42	Analog Devices	If accelerometers are used anyway, it may be better to replace this part with an integrator chip. The reason the real price is lower than ours is that we ordered an "evaluation" gyroscope, with built in PCB and integrator chip.
Innovatek V-490 Video Glasses	$120	Varies	Ebay	Any head mounted display will do. These were cheap and easy to install
Wiring	free	Varies	None, leftovers from a garage door sensor	Wires are needed to connect the microcontrollers to each other, the output to the computers USB, and the accelerometer/gyro to the microcontrollers. Considering the length of the cord form the device to the computer, It may be best to convert to Coaxial cable, or add a wireless transmitter.
Resistors, 2.2kOhm, 4.7kOhm	$.5 each	$.5 each	UoU Parts store	Used to initiate USB connection.
1N4733	$.15 each	$.15 each	UoU Parts store	Used to initiate USB connection.
Solder less development board	$7	$7	UoU Parts store	For a retail product, this would be replaced with a PCB.