Building a USB sensor interface using the ATMEL ATmega16 and the Human Interface Device standard

Mark T. Marshall
Input Devices and Musical Interaction Laboratory
McGill University - Music Technology
555 Sherbrooke St. West
Montreal, QC
Canada
mark.marshall@mail.mcgill.ca

December 28, 2005

Abstract

This document deals with the construction of a device for interfacing sensor systems to a computer through the USB port. The design and sepcifications of the device are given, along with the schematic, parts list, building instructions and usage information.

Introduction

Interfacing computer systems to the outside world is an important aspect of creating new computer music interfaces. We need a means of capturing data from sensors which can be easily used to control a our synthesis engines and musical systems. The interface described in this document gives us such a means. It is designed to be easily accessed on any operating system and from a large number of software applications without the need to write any drivers or additional software.

The heart of this system is the ATMEL ATmega16 microprocessor. The ATmega16 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. It has 16K of program memory, 1K of RAM and 512 bytes of eeprom. It also offers an 8 channel 10-bit analog-to-digital converter, and is programmable using the C language. While it is not a USB-based controller it is possible to easily implement USB on the system. It also has the advantage of lower cost than most dedicated USB microcontrollers, with the total cost of building the interface less than CAN$20.

Layout of this Document

This document is a guide to building the AVR-HID interface. It is divided into a number of sections, which cover various aspects of the AVR-HID system itself and describe the process of building your own device. Later sections of the document also detail the development of the system and describe the code used to program the device, for those who wish to modify the interface or to create their own interface.

The document is divided as follows:

- Section gives an overview of the device and it's functionality.
- Section describes the design of the device, its components, layout and schematic.
- Section covers the process of building the interface, including a parts list and suggested suppliers for the components.
- Section deals with the firmware for the device and how to download it to the device.
- Section details the use of the device with a Mac OS X system, through MAX/MSP.
- Section discusses how to program the device, including the necessary software and libraries.
- Section covers the Human Interface Device standard and details the messages which a compatible device must send and receive and how to modify the existing messages.

- Section gives the location of further information and resources on the AVR-HID device.

Overview

In it's simplest form the AVR-USB device is a 6-channel 10-bit analog-to-digital converter which communicates data to the computer at a rate of 100Hz. All 6 channels are updated at once, so their is no loss of speed when using more sensors with the interface. This version of the system is created by building the board described in this document and downloading the default firmware.

More complex versions of the system, which offer a different number of channels, or a different update rate can be created by editing the source code for the firmware to create your own firmware. Similarly, up to 10 digital inputs can be added to the system in such a way.

The main aim of this document is to describe the building of the basic AVR-HID interface, as this is the one which the majority of users will most likely use. However, it does also cover the details of programming the device so that new firmware can be developed to allow different functionality.

The steps necessary to create your own AVR-HID interface are:

1. Obtain necessary components to build the interface board.
2. Assemble the interface hardware and programmer.
3. Install the necessary programmer software on your computer.
4. Download the firmware to the controller.
5. Make some music.....

Design

This section describes the design of the AVR-HID interface, including the board layout, the schematic and the components.

Layout and Schematic

Figure 1 shows the layout of the AVR-HID board, with the main components labelled. The overall size of the circuit is quite small at around 7.5cm by 7.5cm (3" by 3"). Note that the picture shows an early prototype, so only 2 sensor ports are present, a different ISP socket is used and a USB cable is used in place of a USB socket.

The schematic for the board is shown in figure 2.

Component Descriptions

The following are descriptions of the major components of the AVR-HID board.

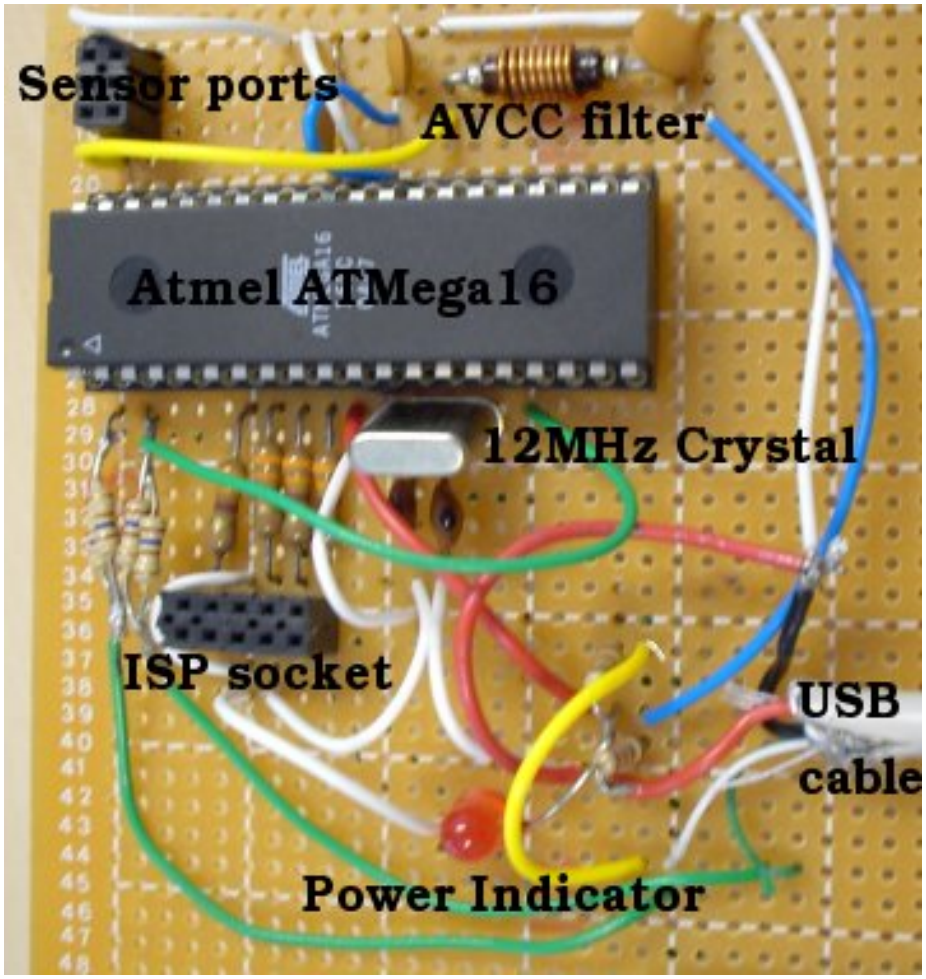

Figure 1: Layout of the AVR-HID board

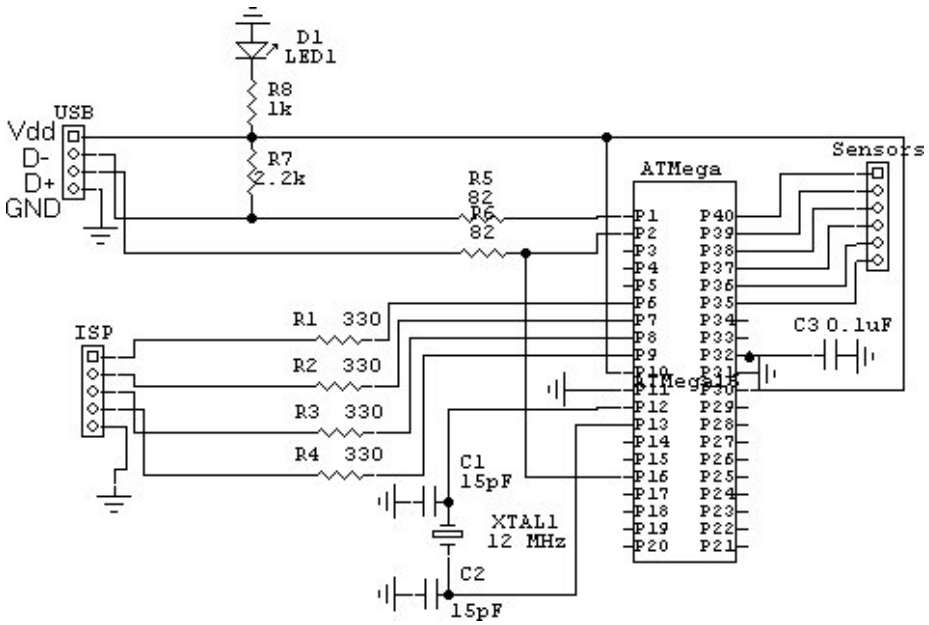

Figure 2: Schematic of the AVR-HID board

USB connector

While figure 1 shows a USB cable, it is recommended that a USB connector socket is used.

The USB connector is a standard USB connector. You can use either a "type B" or a "type A" connector, the only difference being the type of cable required to connect it to the computer. A "type B" connector and an A-to-B cable is recommended, as this is something of a standard for USB-based devices. The USB connector has 4 connections, one for power, one for ground, and two for signal, labelled D+ and D-.

Power Indicator

A standard LED is used to indicate that power has been applied to the circuit from the USB port. It should be noted that the USB system allows a maximum current of 100mA for a single port, which means that should your interface attempt to draw more power than this (such as an interface using many powered sensors) it will not be able to and the LED will not light. In this case, it is necessary to use a seperate 5V DC power supply for the interface.

ISP socket

The In-System Programming (ISP) socket is used to connect the programmer device to the interface to allow dowloading of the firmware.

12MHz Crystal

The 12MHz crystal is required to allow the microcontroller to run at 12MHz, enabling it to operate as a low speed USB device. This speed also allows for other operations, such as analog-to-digital conversion, to be performed by the microcontroller in the time between sending USB messages.

Sensor Ports

These ports are used to allow the connection of up to 6 analog sensors to the AVR-HID interface. The ports have three contacts, for +5V, Ground and Signal. The +5V and ground are used to activate the sensor, and the Signal contact is used for the output from the sensor. Any 3 contact connector can be used for these inputs, but it is suggested that stereo mini-jack connectors are used, as these have the advantages of being small, cheap, readily available and robust. It is also recommended that the sensors are connected to these ports using shielded 3-core audio cable, as this type of cable is strong and flexible and the shielding will reduce the possibilites of interference with the signal, especially over long distances.

The board in figure 1 shows only 2 sensor ports, which use 3-pin header sockets.

ATMEL ATmega16

The ATmega16 microcontroller is the heart of the overall system. It performs all of the message processing, formatting and transmission necessary for USB communication as well as the analog-to-digital conversion for the analog input ports.

AVCC filter

The AVCC filter section is used to filter the +5V supply from the USB port to provide a constant steady 5V supply for use in powering the sensors and as a reference voltage for analog-to-digital conversion.

Building the AVR-HID interface

There are two main steps to building your own AVR-HID interface. These are:

1. Obtain the components
2. Build the circuit board

This section will deal with these steps. Firstly, the necessary components are listed, along with recommended suppliers and prices. Secondly, some advice for the building process is provided.

Obtaining the components

The components are available from a number of sources, the choice of which to use is up to the builder. We recommend ordering the components from Digi-key, as they stock all of the necessary components and usually have the best prices. Alternative suppliers for many of the parts include Mouser, Radio Shack, Active Electronics and Addison Electronics. Please note that to the best of my knowledge, only Digi-key stock the microcontrollers themselves.

Table 1 is a full parts list for a single basic board and the ISP cable for programming the board. It includes part numbers to order from Digi-key and prices, all of which are correct as of June 1st 2005. Prices may be lower if the parts for more than one board are purchased together.

Part	Part No.	Quantity	Unit Price	Total Price
ATMega16	ATMEGA16-16PC-ND	1	8.56	8.56
USB Connector	787780-1-ND	1	1.50	1.50
12MHz Crystal	CTX058-ND	1	1.24	1.24
10uH Inductor	DN2544-ND	1	1.01	1.01
ISP Header	A19341-ND	1	0.50	0.50
40-pin socket	2-641268-1-ND	1	1.88	1.88
Red LED	67-1068-ND	1	0.43	0.43
15pF capacitor	1303PH-ND	2	0.10	0.20
0.1uF capacitor	BC1154TR-ND	3	0.06	0.18
330Ω resistor	330QBK-ND	5	0.074	0.37
62Ω resistor	62EBK-ND	5	0.074	0.37
2.2kΩ resistor	2.2KEBK-ND	5	0.074	0.37
Total				16.61

Table 1: Parts list for 1 AVR-HID board

Please note that you will also need piece of perfboard on which to assemble the circuit and a USB cable to connect to the device. It should also be noted that we order 5 of each resistor, when in some cases only 1 is needed. This is due to minimum order quantities for Digi-key. You could buy them locally in order to reduce the amount you are buying.

Assembling the AVR-HID board

Assembling the AVR-HID board is not particularly difficult for anyone who has some soldering experience. For those assembling the circuit from the diagram, this section will provide some hints and reminders.

Assembly Hints

The only component which needs to be oriented a certain way is the power LED. Be sure to align the longer leg of the LED with the '+' mark on the

circuit diagram.

For best operation, make sure the crystal and its associated capacitors are kept as close to the microcontroller as possible.

The sensor connectors should use the AVCC line for their power supply. This will ensure the most accurate results from the analog-to-digital conversion.

Make sure to connect the D+ line from the USB port to both pins 2 and 16 of the microcontroller.

AVR-HID Firmware

The firmware for the AVR-HID device has been built around a number of libraries which enable it to perform complex tasks, such as sending and receiving the USB and HID messages and analog-to-digital conversion. For efficiency, portions of the firmware have been written in assembly, the rest in C. Section deals in detail with the code and libraries used for the firmware, and how to edit and recompile the firmware should you wish to do so.

Once you have firmware for the device, whether the default firmware which accompanies this document, or your own custom firmware, you need to download the firmware to the device in order to allow the device to work. This section details the downloading of the firmware, including the necessary cable and software to perform the download, and any additional setup required to use the device.

There are two components required to download firmware to the AVR-HID once it has been assembled:

1. Programming cable
2. Programming software

However, there are a number of different software programs available for programming the ATmega16 processor, many of which require their own cable. This document will concentrate on using the AVRDUDE software on both a Linux machine and a Mac running OS X. AVRDUDE is also available for Windows.

Programming cable

Which programming cable you use to program your device depends on two factors: which ports are available on your machine and which software package you are using to program.

The simplest cables are used to program the device from the parallel port of the machine. This is the cable which will be used in this document. If your machine does not have a parallel port, there are also serial port cables which can be used. Details of one of these can be found on the PonyProg [1] website.

[1] http://www.lancos.com/prog.html

Alternatively ATMEL make a serial programmer cable which can be used with all of the software packages and is not very expensive.

The recommended programming cable for OS X is the ATMEL in-system programmer, which can be used in conjunction with a USB/Serial converter to allow programming from the USB port of the machine. It is available from Digi-Key. It is part number ATAVRISP-ND and costs $37.86.

The recommended programmer for Linux systems depends on the available ports on the machine. You can either use the ATMEL in-system programmer from the serial port, in conjunction with a USB/Serial adaptor from the USB port, or build a very simple parallel port adapter as shown in the next section.

Building the parallel port programming cable

In order to make the programmer cable it is necessary only to connect 5 pins of the parallel port to a 5-pin plug which can be attached to the ISP socket on the AVR-HID board. Table 2 shows these connections.

Connector Pin	Parallel Port Pin
1	9
2	10
3	8
4	7
5	18

Table 2: Connections for programmer cable

Programming Software

The programming software used in this document is AVRDUDE by Brian S. Dean. More information on this package is available on his webpage at http://www.bsdhome.com/avrdude/. It is available for Linux and Windows systems and can be downloaded from http://savannah.nongnu.org/download/avrdude/.

Once downloaded and installed to your system the package will allow you to download the firmware to the AVR-HID.

Downloading the firmware

The steps necessary to download the firmware are as follows:

1. Connect the AVR-HID to the USB port
2. Attach the programmer cable between the AVR-HID and the computer
3. Use the AVRDUDE program to download the firmware
4. Set the fuse bits

5. Unplug the programming cable and USB cable

Two of these steps require further explanation, these being how to download the firmware and how to set the fuse bits.

Downloading with AVRDUDE

To download the firmware with AVRDUDE you must execute the one of the following commands:

On Linux systems using the parallel port adapter:

```
avrdude -p m16 -c bsd -U flash:w:avrhid.hex:a
```

The *-p m16* parameter tells the software we are downloading to an ATmega16, the *-c bsd* tells the program indicates that we are using Brian S. Dean's programming cable, *-U flash:w:avrhid.hex:a* tells the program to write (*w*) the file avrhid.hex to the flash memory and to autodetect (*a*) its format.

If using the ATMEL serial programming cable the *-c* parameter changes to *-c avrisp* giving the following command:

```
avrdude -p m16 -c avrisp -U flash:w:avrhid.hex:a
```

If using the ATMEL serial programming cable with a USB/Serial adapter the *-c* parameter changes to *-c avrisp* and you need to add a command giving the correct port:

```
avrdude -p m16 -c avrisp -P /dev/cuUSAXXXXX -U flash:w:avrhid.hex:a
```

where /dev/cuUSAXXXXX is the handle to the USB/Serial adapter. You can find this by typing ls /dev/cu* in a terminal.

Once this is complete, we now need to set the fuse bits.

Setting the fuse bits

The fuse bits are set using AVRDUDE in terminal mode. The fuse bits are used to indicate that the processor is to use an external clock and to determine what forms of programming are allowed for the device. The necessary commands to set these bits are:

```
avrdude -p m16 -c bsd -t
```

or

```
avrdude -p m16 -c avrisp -t
```

or

```
avrdude -p m16 -c avrisp -P /dev/cuUSAXXXX -t
```

This puts the program in terminal mode. Now we must set the high and low fuse bytes. At the prompt type:

```
write hfuse 0x89
```

followed by:

```
write lfuse 0xff
```

Then exit the terminal mode. We have now set the microcontroller to operate using a slowly rising external crystal at ¿1 MHz.

Once this is done you can now unplug the AVR-HID device. When next you connect it to a computer it should automatically be detected as a USB Human Interface Device. The next section deals with how to use the device with MAX/MSP on Mac OS X.

Using the AVR-HID with Max/MSP

Once you have downloaded the firmware to the AVR-HID device, you are ready to use it with your software. The device is detected as a USB joystick and so can be used with any software package which supports joystick or HID input. It should also be automatically detected by your operating system as just such a device. To illustrate how the device is used, this section will take a step-by-step look at using the AVR-HID with Max/MSP.

The first step in using the AVR-HID is to connect it to the USB port of your computer. This should be done before Max/MSP has been started, or the software will not detect the device. Once you have plugged in the AVR-HID, it is time to start up Max/MSP.

When Max/MSP has started, create a new patch and add the "hi" object, giving the following:

This object is the one which is used to interact with human interface devices. Everytime it receives messages from a human interface device it outputs them. Each message is made up of two numbers, the first being the channel on which the message came in, the second being the value of the message. Each analog input of your AVR-HID is assigned to one channel. The first input is channel 7, the last is channel 12.

To read messages from the object you should use a route object, and connect it to the first outlet of the hi object, as shown in the following figure:

This will output messages for each channel to their representative number box.

Now, to have the object focus on the AVR-HID device, we must send it a message telling it the name of the device on which to focus. So, create a message object with the value AVR-HID like this:

Clicking on this message should make the hi object focus on the AVR-HID and it should display a message in the output terminal to this effect.

The final step in communicating with the AVR-HID is make the hi object poll the AV-HID. To do this create a message, which contains the value "poll 10" and connect it to the hi object's first inlet:

Now, to test if this is working ok, connect one or more sensors to the analog inputs of the AVR-HID. When you have done this, click on the poll message. Now, as you manipulate the sensors, you should see messages coming in on the channels corresponding to each sensor. Each message is in the range of 0 to 1023, representing 0 to 5V from the sensor. Congratulations, you can now use your AVR-USB to control patches in Max/MSP.

Notes on the operation of the analog-to-digital converters

The analog-to-digital converters measure inputs in the range of 0 to 5V. The measured voltage is then converted to a 10-bit value by the microcontroller, giving a value in the range of 0 to 1024, which represent 0 and 5V respectively. These numbers are then sent as the data of the HID messages to the computer.

There are six converters active in the AVR-HID device. Therefor each data message from the AVR-HID contains six 10-bit numbers, one for each converter. Even if less than six converters have sensors attached, the message will still contain six numbers.

The value which is transmitted in the message for a converter which does not have a sensor attached will be equal to the value transmitted for the last converter which does have a sensor connected. This means that if you only connect a sensor to converter 1, all converters will transmit the same value. If you connect a sensor to converter 1 and another to converter 3 then converters 1 and 2 will transmit the value of converter 1 and converters 3, 4, 5 and 6 will transmit the value of converter 3.

Programming the AVR-HID

Programming of the AVR-HID is through the C language. In order to program the system, you must have a number of applications and libraries on your computer. The necessary libraries and applications are:

- binutils (available from ftp://ftp.gnu.org/gnu/binutils/)
- gcc (available from ftp://ftp.gnu.org/gnu/gcc/)
- avr-libc (available from http://savannah.nongnu.org/projects/avr-libc/)
- programmer software (see Section)

You will have to download and install the necessary packages to your system. Instructions for installation should be available with each package and may change, so will not be presented here. If a guide to installation is needed the article at http://www.linuxfocus.org/English/November2004/article352.shtml may be of some use. If you are using a Linux system and have installed apt on your

machine, you should be able to easily install all of these packages using the apt system and the PlanetCCRMA repository (see http://ccrma.stanford.edu/planetccrma/software/).

The AVR-HID sourcecode

The AVR-HID sourcecode is built on top of the AVR-USB libraries from Objective Development (see http://www.obdev.at/products/avrusb/). The HID compliance and the additional analog-to-digital code has been developed by the author of this document. The AVR-USB code is included with the source for the AVR-HID and clear comments indicate which portions of the code come from Objective Development.

To compile the sourcecode, you must first download and unpack it. The code is available at: http://somewhere.com. Once you have downloaded it, you should unpack the file to create the AVR-HID directory. Change to this directory and we will begin to examine the structure of the code.

The first thing to do is to make sure that all of the necessary libraries and applications installed correctly. To do this, while in the AVR-HID directory type:

```
make clean
make
```

If there are no errors, we know that all the necessary software is correctly installed. If you do get errors, make sure you have correctly installed the necessary libraries and applications.

Now, we can look at the files which make up the sourcecode. In this AVR-HID directory there are a number of .c and .h files. These are:

- main.c - the main loop for the application
- usbconfig.h - the definitions of various USB and HID configuration data
- a2d.h - the analog to digital conversion routines

There is also a subdirectory named usbdrv, which contains the code for the usb and hid communication. The main files in this directory are:

- usbdrv.h - declarations of usb functions
- usbdrv.c - definitions of usb functions and data structures and HID data structures
- usbdrvasm.S - assembly code for timing-critical sections of Objective Developments AVR-USB

Of these 6 files that make up the AVR-HID system, there are 3 which might need to be changed when programming new firmware for the AVR-HID. These are main.c, which is where the main code for the operation of the AVR-HID goes, usbconfig.h, which contains the definitions of the device configuration data and usbdrv.c which contains the messages which are sent in response to the standard USB and HID requests from the computer.

Description of source files

main.c

The main.c file contains the initialization and main loop code for the AVR-HID. The functions which initialize USB and set up the interrupts to respond to the USB messages are contained here. The main loop (marked in the code) contains the calls to the analog-to-digital conversion functions, the conversion of the data to the format required for the USB messages and a call to the usbSetInterrupt function, which sets the data to be sent the next time data is requested by the computer over the USB connection.

If you wish to change the main operation of the AVR-HID you will need to edit code in the main loop. To add more analog-to-digital channels to the system, you will need to call the conversion functions for the additional channels, format the data in the required message format, and edit the call to the usbSetInterrupt function. If you wished to use digital inputs, or activate digital outputs, this code would also go in the main loop.

The formating of the data messages for USB responses is covered in section .

usbconfig.h

This file contains definitions of the USB parameters such as device type, class, name etc. These can be edited when modifiying the AVR-USB to interact with the USB in a different way. Details can be found in the USB specification.

usbdrv.c

This file contains the main functions for interracting with USB and the definitions of the various decriptors which a USB device must provide during the enumeration process. The parts which you are most likely to need to change are these descriptors. More information on the descriptors can be found in section , on the USB developers site (http://www.usb.org/developers/), and in an article at: http://www.embedded.com/2000/0010/0010ia2.htm.

Building the firmware

The building of the firmeware is controlled by a Makefile. If the changes you have made are only within the files already included in the Makefile then you will not need to change the Makefile. In this case you can build the firmware with the command:

```
make
```

and install the firmware on to the AVR-HID using the command:

```
make flash
```

If you have added new files to the sourcecode, then you must edit the Makefile to add these files to the compilation. In this case, add the name of your file, with a .o extension to the OBJECTS line in the Makefile. This makes the objects line look something like this:

```
OBJECTS = usbdrv/usbdrv.o usbdrv/usbdrvasm.o usbdrv/oddebug.o a2d.o
main.o yourfile.o
```

You can then build and install the firmware using the commands detailed above.

USB and HID standards, descriptors and messages

Device requirements

The USB standard details the way in which USB devices should operate in order to be detected correctly as USB devices. In order to meet this standard, there are certain processes which a device should follow and certain messages which it should recognise.

The requirements for a USB device are:

- must follow USB enumeration process
- it must have a device descriptor and configuration descriptor
- must contain a control endpoint for control messages
- must recognise standard control messages

Additionally, for a HID device, there are the following requirements:

- must have an interrupt in endpoint to send periodic data to host
- must contain a class descriptor and at least one report descriptor
- must respond to the GET REPORT message

The following sections detail these requirements.

Enumeration

The first requirement for a USB device is that it must follow the USB enumeration process. The enumeration process begins when a USB device is plugged in and involves the sending and receiving of messages between the device and the host, which determine what the device is and how it works.

In order to correctly enumerate, a device must recognise the Get Descriptor message, and respond with its device descriptor and its configuration descriptor when these are requested. For a HID device, it must also recognise the Get Descriptor message when the report descriptor is requested.

Descriptors

As mentioned above, a USB device must provide certain descriptors and a HID device must provide some more. This section will describe each of these descriptors and illustrate using the default descriptors from the AVR-HID device.

Device Descriptor

The device descriptor is the main descriptor for the device. It contains information about which version of USB the device supports, the class and subclass of the device, the vendor and product id and the vendor and product name.

Configuration Descriptor

The configuration descriptor or descriptors describe configurations of the device. This includes power consumption, number of interfaces to the device, number of endpoints, device class and version, length of the report descriptor and descriptions of the endpoints present in the device.

Report Descriptor

The report descriptor describes the format of a report from the device. It contains the exact number of fields in the report, the format of each field and minimum and maximum values. This is the most important descriptor in the system, as it determines which information is sent and in which format. Also, an incorrect report descriptor will result in a device not being detected by the USB.

The format of a report descriptor is quite complex. For a detailed description of report descriptors and some tools for creating your own please see http://www.usb.org/developers/hidpage/.

Control Endpoint

A control endpoint is used to send and receive messages between the USB device and the system. The system will send specific control messages along this endpoint and the device will reply along the endpoint.

Interrupt-in endpoint

The interrupt-in endpoint is where interrupts are received to indicate that the device should send its report data. The report data is then sent along this endpoint to the system. It is important that when the interrupt is received the data report is ready in the endpoint buffer. This means that your program should be constantly updating the endpoint buffer so that it is ready to be sent on receipt of the interrupt.

GET REPORT message

The device must also be able to respond to a GET REPORT message from the system. This message is sent at the end of the enumeration process but before the device is activated. It is used to ensure that the device sends the reports its report descriptor says it will send.

Further Information

The AVR-HID project is still undergoing further development. This work includes adapting the circuit for use with smaller Atmel chips such as the ATtiny26, developing PCB layouts for the AVR-HID, developing a version which uses surface mount components to allow for a much smaller device and the development of a wireless version which makes use of RF transmitters and receivers.

Further information and updates are available from the project homepage at: