Professor Dr Thomas C. Baier, DG8SAQ

University of Applied Sciences, Prittwitzstrasse 10, 89075 Ulm, Germany, baier@hs-ulm.de

A Low-Cost, Flexible USB Interface

No parallel port on your laptop PC?
Use this interface and a USB port to control external devices.

After publication of my DDS-based, parallel-printer-port-controlled vector network analyzer, I have received a lot of requests asking to consider using the USB interface instead, since many modern notebook computers don't have a parallel printer port.[1] This was motivation enough for me to look into how a USB interface could be realized in a simple and low cost way.

If special tasks are required, such as controlling a VNA, a simple USB-to-parallel or USB-to-serial device won't suffice. A microcontroller will be necessary. There are many microcontrollers available on the market with built-in USB hardware, but generally these are more expensive than simple ones without USB support.

During an Internet search, I stumbled over a freeware firmware-only USB solution on the Objective Development Web site. This interface is realized with a low cost ATMEL AVR IC chip.[2] After a couple of weeks of learning how to program AVRs, I could easily modify the PowerSwitch reference example given on the Objective Development Web site, to control a DDS and do other things.

To me, there seem to be a multitude of possible Amateur Radio applications for this AVR-USB IC, but this solution doesn't seem to be widely known among hams. Therefore I have decided to describe it here in a simple way.

Since I know little about USB ports, I don't dare to write anything about USB theory. I would like to show, however, that even without knowing the intricacies of USB, such a device can easily be built and configured.

Hardware

The necessary hardware to control a DDS unit via USB is seen in Figure 1. The circuit consists of an ATMEL ATTiny2313

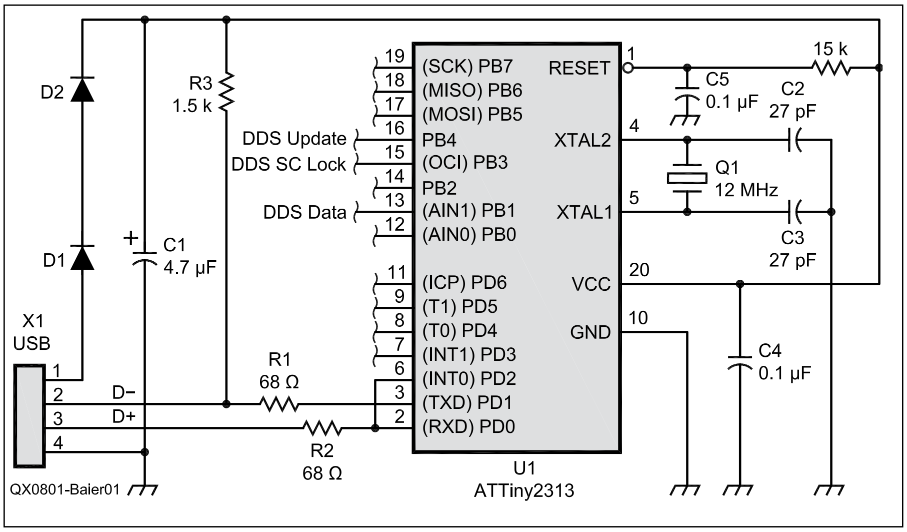
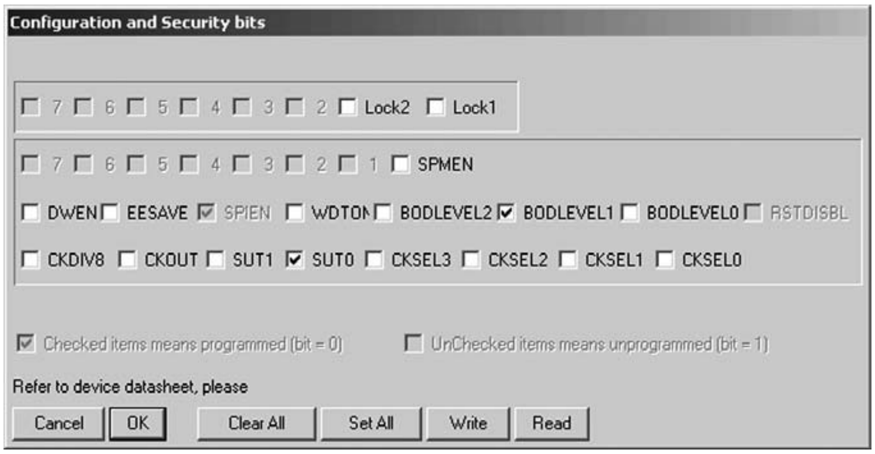


Figure 1 — The schematic diagram of the USB interface shows the computer USB connector and an ATMEL ATTiny2313 IC. Only a few additional components are required to complete the circuit.


Configuration and Security bits

☐ 7 ☐ 6 ☐ 5 ☐ 4 ☐ 3 ☐ 2 ☐ Lock2 ☐ Lock1

☐ 7 ☐ 6 ☐ 5 ☐ 4 ☐ 3 ☐ 2 ☐ 1 ☐ SPMEN

☐ DWEN ☐ EESAVE ✓ SPIEN ☐ WDTON ☐ BODLEVEL2 ✓ BODLEVEL1 ☐ BODLEVEL0 ☐ RSTDISBL

☐ CKDIV8 ☐ CKOUT ☐ SUT1 ✓ SUT0 ☐ CKSEL3 ☐ CKSEL2 ☐ CKSEL1 ☐ CKSEL0

✓ Checked items means programmed (bit = 0)   ☐ UnChecked items means unprogrammed (bit = 1)

Refer to device datasheet, please

Figure 2 — This screen shot shows the *Pony Prog* menu with the correct "fuse" settings for the interface. Note the check marks in the BODLEVEL1 and SUT0 boxes.

[1]Notes appear on page 13.

Table 1
Data Structure of a USB Data Transfer

Data	*Size*	*Usage*	*Access in usbFunctionSetup*		
request type	1 Byte	driver usage	data[0]	equ.	rq->bmRequestType
request	1 Byte	user command	data[1]	equ.	rq->bRequest
value	2 Bytes	user data	data[2] (lo), data[3] (hi)	equ.	rq-> wValue
index	2 Bytes	user data	data[4] (lo), data[5] (hi)	equ.	rq->wIndex
length	2 Bytes	high byte ignored by driver	data[6] (lo), data[7] (hi)	equ.	rq-> wLength
optional data payload	length bytes	user data	see usbFunctionWrite		

MCU and a 12 MHz crystal. The USB driver requires an MCU clock frequency of 12 MHz. The latest driver version also allows a 16 MHz crystal clock, or on some devices, even 16.5 MHz derived from the internal RC oscillator. The MCU is powered from the USB hub. Diodes D1 and D2 are necessary to reduce the output voltage swing of the MCU at the USB D+ and D– lines to the USB specification limits. It is important to mention that the hardware inside the MCU can be configured by programming so-called "fuse bits," which define, for example, whether the internal RC oscillator or the external crystal is used. The AVR-USB device will not work with the factory fuse settings. The fuses need to be programmed as shown in Figure 2.

Figure 3 shows my USB interface with a DDS board and serial programming interface connected for *PonyProg*, described in the next section of the text.

DG8SAQ

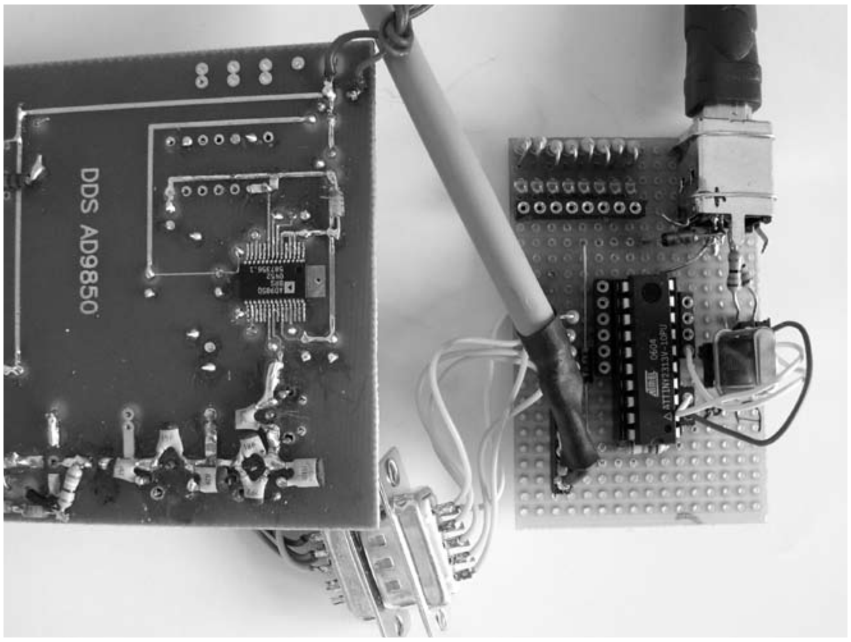


Figure 3 — This photo shows the USB interface connected to a DDS board and serial programming interface.

Development Tools

Before an MCU can do its job, software needs to be written and transferred into the device. For all these tasks there are excellent free software tools available. The software package *WinAVR* is a powerful yet free software development platform for ATMEL AVR MCUs based on the well-known GNU GCC compiler for C and C++.[3] I found that the older version, *WinAVR-20060421,* produces the smallest code size since the firmware was apparently optimized on that compiler version. *AVR Studio 4* from ATMEL is a free integrated development environment for AVRs, including Assembler and a code simulator. In its latest version, C-code can be programmed and debugged from within *AVR Studio* through the usage of *WinAVR* as a plug-in.[4] This makes the combination of *WinAVR* and *AVR Studio* very comfortable to use. Both are running on Microsoft's *Windows XP* operating system. All that is necessary to transfer the compiled code from the PC into the MCU flash memory is a cable connecting 4 lines of the parallel printer port with the MCU and the free software *PonyProg*, which can also be accessed as plug-in from within AVR Studio.[5, 6] *PonyProg* can also program the AVR fuse bits.

Firmware

The AVR USB firmware is partly written in Assembler for the time critical sections and in C for ease of interface to user code. The driver itself consumes about 1.4 kbytes of the 2 kbytes flash memory available in the selected MCU. This leaves about 600 bytes of space for user code. For larger user programs, it is helpful that all 8-bit AVR MCUs share the same CPU core, thus the driver can easily be adapted to any other AVR type, such as those with bigger flash memory.

Even though the USB specification and the firmware allow for several modes of data transfer, I have used only the simplest one, which is the so-called USB control transfer. It allows you to send or receive up to 254 bytes of data in one shot, and it offers high priority on the host side (this is the PC side, as described later).

Table 1 shows the structure of the transmitted data in a single USB control transfer. If only a few data bytes need to be transferred from the host to the USB device, the value and index words (2 bytes) can be used, and no data payload is needed, which means a length of 0.

Program Listing 1 shows my user interface in the main.c program to the firmware driver.

As can be seen from the listing, depending on the request value, data[1], different user tasks are performed. The simplest task is the first one (ECHO value), which simply sends the two value bytes back to the PC. This function is useful for diagnostics only. The return value of the discussed function specifies the number of bytes (stored in replyBuf) to be answered back to the host machine on the control transfer. Obviously, the USB device implemented here can do much more than just control a DDS. It can write or read any of the MCU port pins, widening its usage even further, to other switching and controlling applications without firmware changes.

Request 5 is a special request, with return value 0xff, which instructs the driver that a data payload is available and the user function usbFunctionWrite is to be called. That function is shown in Program Listing 2.

Here, the data payload is received and sent without modification to the DDS chip. If the value is nonzero, a DDS data update pulse is issued. No DDS type specific code is implemented in the firmware, but the firmware supports any DDS type. For different types, only the data payload has to be adapted on the PC side. Also, the data doesn't need to be sent in one chunk, but the DDS control word and data can be sent in separate control transfers.

Host Software

In order to enable a Windows PC to access the USB hardware, a device driver is necessary. Just like Objective Development's reference example, I use the freeware LibUSB driver.[7] It is also possible to use the HID driver integrated in Microsoft *Windows XP* if the USB device is configured as a Human Interface Device. Once the driver is installed and the device is plugged in, it can be accessed by means of any programming language on the host PC side. Since my personal preference is *Pascal*, I have written the host software in *Delphi*. In order to access the interface from within *Delphi*, a LibUSB to Pascal headerfile is necessary. This describes the driver interface in a *Pascal* way.[8] A considerable amount of *Delphi* coding is necessary to establish a connection to our USB device and to diagnose it. This part of the code is identical for any application, though, and can be reused. The only thing that needs to be adapted is the data transmitted in the USB control transfer call. Program Listing 3 shows this section of the *Pascal* host code.

It displays the function call, which transmits "len" bytes of data stored in "buffer" from the host PC to the USB device. The variables "request," "value" and "index" have the same meaning as discussed in Firmware section of this article. Varying these before the control transfer call will let the USB device do all kinds of desired jobs. The return value of the usb_control_msg function is the number of bytes answered back from the USB device in "buffer." The control transfer has high priority on the Windows host system, and requires about 5 ms in order to reach the USB device.

Summary

A simple, flexible and low cost USB hardware interface based on an Atmel AVR microcontroller and on Objective Development's free firmware has been introduced in a hopefully instructive way. I hope this will enable readers who are not MCU programming specialists to customize the solution to their own needs. The source codes, binaries and schematics can be downloaded from the author's Web site.[9] For those who prefer to download the files from the ARRL Web site, they are also available at **www.arrl.org/qexfiles**.[10]

Thanks to the following Lars Kvenild of Atmel Norway for excellent software support and to Christian Starkjohann of Objective Development for great forum support and for reviewing this article.

Notes

[1]Professor Dr Thomas C. Baier, DG8SAQ, "A Low Budget Vector Network Analyzer for AF to UHF," *QEX*, Mar/Apr 2007, ARRL, pp 46-54. See also **www.mydarc.de/DG8SAQ/VNWA/**

[2]**www.obdev.at/products/avrusb/**

[3]**http://sourceforge.net/projects/winavr/**

[4]AVR Studio 4.13, build 528 (Release) from **www.atmel.no/beta_ware/**

[5]**s-huehn.de/elektronik/avr-prog/avr-prog.htm.**

[6]**www.lancos.com/**

[7]**libusb-win32.sourceforge.net/**

[8]The LibUSB.pas header file was written and provided by Yvo Nelemans through private communication. It can be downloaded from Objective Development's Web page in the PowerSwitch reference example. See Note 2.

[9]**www.mydarc.de/dg8saq/AVR-USB/**

[10]The program files associated with this article are available for download from the ARRL Web site. Go to **www.arrl.org/qexfiles** and look for the file **1x08_Baier.zip**. Be aware that the author and manufacturer's Web sites may have updated listings available for download.

Professor Dr. Thomas Baier, MA, teaches physics, mathematics and electronics at the University of Applied Sciences in Ulm, Germany. Before his teaching assignment, he spent 10 years of work on research and development of surface acoustic wave filters for mobile communication with Siemens and EPCOS. He holds 10 patents.

Tom, DG8SAQ, has been a licensed radio amateur since 1980. He prefers the soldering iron to the microphone, though. His interests span from microwave technology to microcontrollers. Lately, he has started Windows *programming with* Delphi. *Tom spent one year in Oregon USA rock climbing and working on his master's degree.*

See Program Listings starting on next page.

ATOMIC TIME

1010 Jorie Blvd. #332
Oak Brook, Il 60523
1-800-985-8463
www.atomictime.com

ADWA101 - $49.95

14" LaCrosse Black Wall
WT-3143A $26.95
This wall clock is great for an office, school, or home. It has a professional look, along with professional reliability. Features easy time zone buttons, just set the zone and go! Runs on 1 AA battery and has a safe plastic lens.

Digital Chronograph Watch
ADWA101 $49.95
Our feature packed Chrono-Alarm watch is now available for under $50! It has date and time alarms, stopwatch backlight, UTC time, and much more!

WT-3143A - $26.95

WS-8248 - $64.95

LaCrosse Digital Alarm
WS-8248U-A $64.95
This deluxe wall/desk clock features 4" tall easy to read digits. It also shows temperature, humidity, moon phase, month, day, and date. Also included is a remote thermometer for reading the outside temperature on the main unit. approx. 12" x 12" x 1.5"

1-800-985-8463
www.atomictime.com
Quantity discounts available!

WS-9412U - $19.95

LaCrosse WS-9412U Clock $19.95
This digital wall / desk clock is great for travel or to fit in a small space. Shows indoor temp, day, and date along with 12/24 hr time. apx 6"x 6"x 1"

Tell time by the U.S. Atomic Clock -The official U.S. time that governs ship movements, radio stations, space flights, and warplanes. With small radio receivers hidden inside our timepieces, they automatically syncronize to the U.S. Atomic Clock (which measures each second of time as 9,192,631,770 vibrations of a cesium 133 atom in a vacuum) and give time which is accurate to approx. 1 second every million years. Our timepieces even account automatically for daylight saving time, leap years, and leap seconds. $7.95 Shipping & Handling via UPS. (Rush available at additional cost) Call M-F 9-5 CST for our free catalog.

Program Listing 1

The C function usbFunctionSetup in the main.c program listing is the user interface to the firmware driver.

```
USB_PUBLIC uchar usbFunctionSetup(uchar data[8])
{
usbRequest_t *rq = (void *)data;
static uchar    replyBuf[3];
   usbMsgPtr = replyBuf;
   if(rq->bRequest == 0){                          // ECHO value
      replyBuf[0] = data[2];             // rq->bRequest identical data[1]!
      replyBuf[1] = data[3];
      return 2;
   }
   if(rq->bRequest == 1){                          // set port directions
      DDRA = data[2];
      DDRB = data[3];
      DDRD = data[4] & (~USBMASK & ~(1 << 2));// protect USB interface
      return 0;
   }
   if(rq->bRequest == 2){                          // read ports
      replyBuf[0] = PINA;
      replyBuf[1] = PINB;
      replyBuf[2] = PIND;
      return 3;
   }
   if(rq->bRequest == 3){                          // read port states
      replyBuf[0] = PORTA;
      replyBuf[1] = PORTB;
      replyBuf[2] = PORTD;
      return 3;
   }
   if(rq->bRequest == 4){                          // set ports
      PORTA = data[2];
      PORTB = data[3];
      PORTD = data[4];
      return 0;
   }
   if(rq->bRequest == 5){                          // use usbFunctionWrite to transfer len bytes to DDS
      usb_val = data[2];                           // usb_val!=0 => DDS update pulse after data transfer
      return 0xff;
   }
   if(rq->bRequest == 6){
      PORTB = PORTB | DDS_UPDATE;                          // issue update pulse to DDS
      PORTB = PORTB & ~DDS_UPDATE;
      return 0;
   }
   replyBuf[0] = 0xff;                             // return value 0xff => command not supported
   return 1;
}
```

Program Listing 3

Issue a control transfer command in the out direction (USB_ENDPOINT_OUT) with the data payload stored in the buffer with length len bytes to be sent to the USB device. The variables "request," "value" and "index" have the same meaning as discussed in the Firmware section.

```
usb_control_msg(handle, USB_TYPE_VENDOR or USB_RECIP_DEVICE or USB_ENDPOINT_OUT, request,
                                   value, index, buffer, len, 5000);
```

Program Listing 2

The usbFunctionWrite command in the main.c program listing sends the data payload directly to the DDS chip.

```
USB_PUBLIC uchar usbFunctionWrite(uchar *data, uchar len) //sends len bytes to DDS_SDA
{
uchar i;
uchar b;
uchar adr=0;
        while (len!=0){
                b=1;
                for (i=0;i<8;i++){
                        if (b & data[adr]){
                                PORTB = (PORTB | DDS1_SDA) & ~DDS_SCL;
                                PORTB = PORTB | DDS_SCL;
                        }
                        else{
                                PORTB = PORTB & (~DDS1_SDA & ~DDS_SCL);
                                PORTB = PORTB | DDS_SCL;
                        }
                        b=b<<1;
                }
        len--;
        adr++;
        }
if (usb_val){
  PORTB = PORTB | DDS_UPDATE;          // update DDS
  PORTB = PORTB & ~DDS_UPDATE;
  }
return 1;
}
```

QEX