CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

by Yoshiyasu Takefuji

Programmable Power

Build a Simple USB DAC

Yoshiyasu describes the step-by-step construction of a simple USB DAC around an ATtiny45 and a MAX517. You can use the system as a programmable power supply.

In this article, I'll explain how USB communication can be implemented on an Atmel ATtiny45 eight-pin DIP with open-source software packages (libusb, Cygwin, WinAVR, and AVR-USB). WinAVR compiles the target firmware program under Windows. AVR-USB is an open-source USB protocol stack for firmware, which can be compiled by a GNU C compiler under WinAVR. libusb is also an open-source USB protocol stack for a host PC, which is used under Cygwin on Windows, to compile a USB application program.

With the open-source software packages, the USB 1.1 or USB 2.0 low-speed function can be easily achieved without a USB chip. The size of a USB protocol stack for firmware embedded in the ATtiny45 is about 2 KB. Therefore, more than 2 KB is available on the ATtiny45 for user programming. If you use an ATtiny85, 6 KB of space will be available for programming. The USB connector has four pins: 5 V, GND, D–, and D+. The D– and D+ pins are used on the ATtiny45 for USB communication with another two pins for XTAL1 and XTAL2 with a 12-MHz resonator. Note that 1.5 Mbps of USB streaming data can be decoded by the open-source AVR-USB package's USB protocol stack. The open-source USB protocol stack gives you a user-friendly API, where usb_control_msg and usbFunctionSetup functions are used for USB communication between a host PC and an ATtiny45.

The circuit is built in a breadboard with a 1.5-kΩ resistor for the 5-V pullup of D– and two 68-Ω series resistors directly connected to PD0 and PD2 from D– and D+, respectively. The 3.3-V CMOS voltage regulator is used to feed the ATtiny45. Because six ATtiny45 pins (5 V, GND, D–, D+, XTAL1, and XTAL2) are used, two pins are left for MAX517 two-wire serial programming. The MAX517 is a two-wire, 8-bit DAC. The necessary commands must be supplied from an ATtiny45 for D/A conversion.

Figure 1 shows the circuit diagram of the programmable USB power supply. A female USB connector can be custom-made for a user-friendly breadboard (see Photo 1). Photo 2 shows the

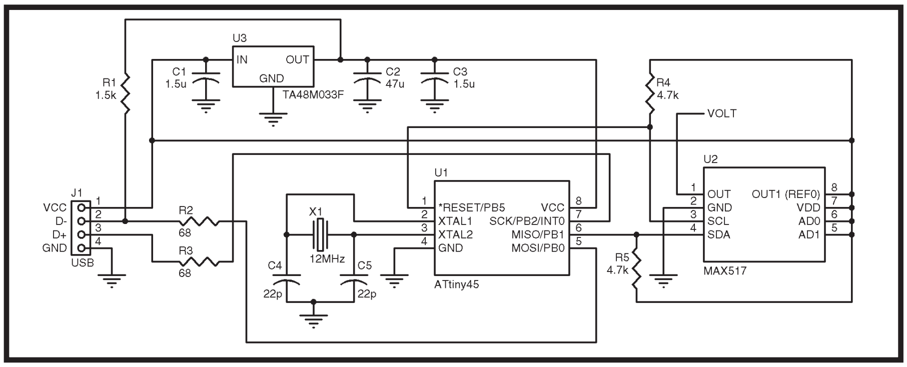

Figure 1—This is a circuit diagram of a programmable USB power supply using an Atmel ATtiny45 and a MAX517 8-bit DAC.

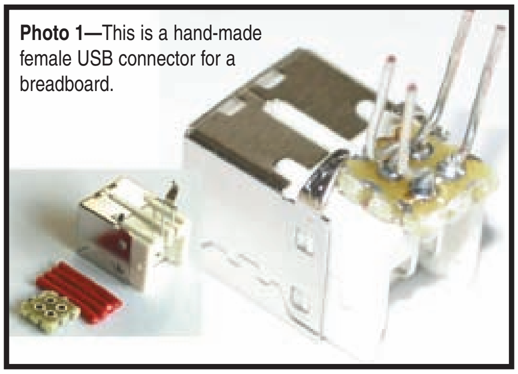
Photo 1—This is a hand-made female USB connector for a breadboard.

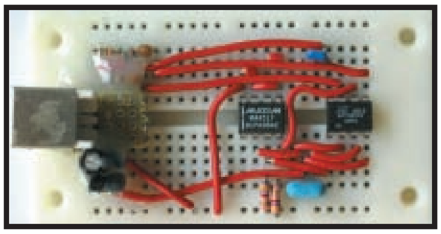
Photo 2—This is a completed programmable USB power supply using an ATtiny45 and a MAX517.

complete circuit.

The system features an ATtiny45, a MAX517, a breadboard, a 3.3-V voltage regulator, a 12-MHz ceramic resonator, and a female USB connector. It also includes one 1.5-kΩ resistor, two 68-Ω resistors, and two 4.7-kΩ resistors.

The installed software packages on Windows include Cygwin (gcc-core, gcc++, libusb, and other necessary libraries). Download the setup file and double-click it to install the necessary packages (www.cygwin.com/setup.exe). WinAVR must also be installed. Download the latest WinAVR-xxx-install.exe file and double-click the downloaded installation file (http://winavr.sourceforge.net/download.html).

To install Cygwin first, double-click setup.exe in Windows and select the root install directory (in my case, c:\cygwin is given). Next, pick direct connection, choose the nearest site from the list, and select the necessary packages for your system. You must at least install gcc-core, gcc++, and libusb.

USB COMMUNICATIONS

USB 2.0 has three communication types: 1.5 Mbps (low speed), 12 Mbps (full speed), and 480 Mbps (high speed). I established 1.5-Mbps low-speed communication using an ATtiny45 with a 12-MHz ceramic resonator without a USB chip.

An open-source software protocol stack can take care of non-return-to-zero inverted (NRZI) encoding, decoding, and bit stuffing for synchronization. The conventional USB chip uses NRZI encoding/decoding with automated bit stuffing. In the bit-stuffing technique, each time a series of five consecutive "0" bits is transmitted, a "1" bit is automatically added to force a transition. In this article, usb_control_msg and usbFunctionSetup functions are used for USB communications. usb_control_msg is used for host PC USB communications. usbFunctionSetup is used for ATtiny45 USB communications.

FIRMWARE

The avrusb protocol stack is embedded in the firmware of the USB device. It provides you with user-friendly APIs including the usbFunctionSetup function. The usbFunctionSetup function is used to send/receive data via USB between the host PC and the USB device (ATtiny45). The data mapping is detailed in Figure 2. Because usbFunctionRead and usbFunctionWrite are not used in this article, those functions are described at www.obdev.at/products/avrusb/index.html.

In AVR-USB, usbconfig.h plays a key role in firmware configuration. The USB configuration port is defined in Listing 1. usbconfig.h device descriptions are defined in Listing 2.

Listing 1—This is part of a usbconfig.h file for hardware configuration where PortB, PB0 for D–, and PB2 for D+ are set.

```
/*    usbconfig.h   */
#define    USB_CFG_IOPORT       PORTB
/* This is the port where the USB bus is connected. When you
 * configure it to "PORTB", the registers PORTB, PINB (=PORTB+2)
 * and DDRB (=PORTB+1) will be used.
 */
#define    USB_CFG_DMINUS_BIT   0
/* This is the bit number in USB_CFG_IOPORT where the USB D- line
 * is connected. This MUST be bit 0. All other values will result
 * in a compile error!
 */
#define    USB_CFG_DPLUS_BIT    2
/* This is the bit number in USB_CFG_IOPORT where the USB D+ line
 * is connected. This may be any bit in the port. Please note
 * that D+ must also be connected to interrupt pin INT0!
 */
```

Listing 2—This is part of a usbconfig.h file for hardware configuration where VENDOR_ID, DEVICE_ID, VENDOR_NAME, and DEVICE_NAME are set and disabling FN_WRITE and FN_READ.

```
#define USB_CFG_VENDOR_ID              0x84, 0x13
#define USB_CFG_DEVICE_ID              0x88, 0x88
#define USB_CFG_DEVICE_VERSION         0x00, 0x01
#define USB_CFG_VENDOR_NAME            'D', 'e', 'v', 'D', 'r', 'v'
#define USB_CFG_VENDOR_NAME_LEN        6
/* These two values define the vendor name returned by the USB
 * device. The name must be given as a list of characters under
 * single quotes. The characters are interpreted as Unicode
 * (UTF-16) entities.If you don't want a vendor name string,
 * undefine these macros.
 */
#define USB_CFG_DEVICE_NAME            'U', 'S', 'B', '-', 'K', 'O'
#define USB_CFG_DEVICE_NAME_LEN        6
/* Same as above for the device name. If you don't want a device
 * name, undefine the macros.
 */

Disable usbFunctionWrite and usbFunctionRead in this example.
#define USB_CFG_IMPLEMENT_FN_WRITE     0
/* Set this to 1 if you want usbFunctionWrite() to be called for
 * control-out transfers. Set it to 0 if you don't need it and
 * want to save a couple ofbytes.
 */
#define USB_CFG_IMPLEMENT_FN_READ      0
/* Set this to 1 if you need to send control replies which are gen-
 * erated "on the fly" when usbFunctionRead() is called. If you only
 * want to send data from a static buffer, set it to 0 and return
 * the data from usbFunctionSetup(). This saves a couple of bytes.
 */
#define USB_CFG_DEVICE_CLASS   0xff
#define USB_CFG_DEVICE_SUBCLASS        0
#define USB_CFG_INTERFACE_CLASS        0
#define USB_CFG_INTERFACE_SUBCLASS     0
#define USB_CFG_INTERFACE_PROTOCOL     0
```

Listing 3—This is a full source listing of a USB device firmware program for an ATtiny45 and a MAX517.

```
/* Programmable USB power supply using TINY45 and MAX517 */
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/wdt.h>
#include "usbdrv.h"

void delay(unsigned int p)
{ unsigned char i, j;                      //one loop is 3.8us with 12MHz
    for(i=0;i<p;i++)  for(j=0;j<10;j++);    }

void pulse(unsigned char data)
{ if((data & 0x01)==0){PORTB = 0x00; }      //SDA=0
    else {PORTB = 0x02;  }          //SDA=1
delay(1);        PORTB = PORTB | 0x20;      //SCL=1
delay(1);        PORTB = PORTB & 0x02;      //SCL=0
delay(1);        }

void start_strm()
{               PORTB = 0x20;               //SDA=0
    delay(1);  PORTB= 0x00; delay(1);       }

void stop_strm()
{ PORTB = 0x20;                             //SCL=1
    delay(1);    PORTB = 0x22;              //SDA=1
    delay(1);    }

void ack_strm()
{ PORTB = 0x00;                             //SDA=0
    delay(1);    PORTB = 0x20;              //SCL=1
    delay(1);    PORTB = 0x00;              //SCL=0
    delay(1);    }

/*-------------generate pulse stream---------------*/
void pulse_strm(unsigned char sda)
{   unsigned char i,ret;
    for(i=0;i<8;i++){ret=(sda>>(7-i)); pulse(ret); }      }

uchar usbFunctionSetup(uchar data[3])
{ static uchar replybuf[1];
usbMsgPtr = replybuf;
    if(data[1] == 0){
       replybuf[0]=55;
       start_strm(); pulse_strm(0x5E);      //01011110 AD1=AD0=1
       ack_strm();  pulse_strm(0x00);       //00000000 RST=PD=A0=0
       ack_strm(); pulse_strm(data[2]);     // data[2] is for voltage
       ack_strm();   stop_strm(); }
    else if(data[1] == 1){
       replybuf[0]=11;}
    else if(data[1] == 2) {
       replybuf[0]=22;}
       return 1; }

int main(void)
{ DDRB = 0x22;                              // 0010 0010 PB1=SDA and PB5=SCL
                                            // are output
    PORTB = 0x22;                           //SDA=SCL=1
    usbInit();
    sei();
    for(;;){     usbPoll();         }
    return 0;    }
/************************end of program*************************/
```

Two-byte data from the host is sent to the target USB device using usbFunctionSetup to initialize the MAX517 (two-wire, 8-bit DAC) and set the voltage of the D/A output. The first data byte sets the mode. The second data byte sets the voltage of the D/A output. A pulse stream from the ATtiny45 is sent to the MAX517 to program a DAC. WinAVR is used for compiling the described source program. PB1 and PB5 (reset) are used for the MAX517 two-wire programming setting. Once the RESET pin is set up by writing the lfuse and hfuse, the conventional serial programmer can no longer write the ATtiny45's flash memory because the reset function is disabled. To reactivate the ATtiny45's RESET pin for further programming, high-voltage serial programming must be used.

To compile the source program, type "make" in a Cygwin window or execute "Make All" on the programmer's notepad in WinAVR. After a successful make operation, a da.hex file should be generated. A da.hex file is used to write the ATtiny45's flash memory with the program writer. Modify WRITER=xxx in a makefile according to your own program writer. In a Cygwin window, type "make fuse" and "make hfuse" to update the fuse bytes (see Listing 3).

MAX517 (DAC) PROGRAMMING

The MAX517 is an 8-bit DAC where two-wire programming is needed to set analog OUT0. Three bytes (slave address byte, command byte, and output byte) must be sent to the MAX517 to set the analog output (see Figure 3).

As you can see in Figure 1, AD1=AD0=1 makes the slave address byte 01011110. The command byte becomes 00000000. The output byte is given by "data[2]" with a range of 0 to 255 in the usbFunctionSetup function in the firmware, which is equivalent to the value of the parameter "i" in the usb_control_msg function in the host program.

Analog output can be determined by:

$$V_{REF0}\left(\frac{\text{data}[2]}{256}\right)$$

In Figure 1, the REF0 pin is connected to 5 V and the DAC resolution is 0.0195 V. Therefore, the analog output should be between 0 and 4.98 V. Figure 4 is a simplified DAC diagram.

FIRMWARE, SOFTWARE, & LIBUSB

The makefile is used to generate a da.hex file, which is written in the ATtiny45's flash memory. The makefile is posted on the *Circuit Cellar* FTP site.

LibUsb-Win32 is an open-source USB protocol stack library for Windows. There are two important functions: usbOpenDevice and usb_control_msg in this application. usbOpenDevice opens the USB device to ensure the target VendorID (VID)

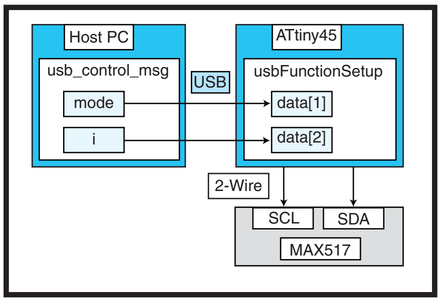

Figure 2—This is parameter mapping between a host computer and a USB device using an ATtiny45 and a MAX517.

and ProductID (PID) are there. VID and PID are my unique commercial IDs, but you may use them for your personal use. usb_control_msg is a user-friendly function in the libusb protocol stack package. usb_control_msg sends and receives the data between a host PC and the ATtiny45. Two important parameters (mode and i) in usb_control_msg on the host application are transferred to two parameters (data and data) via usbFunctionSetup on firmware. The parameter mapping between usb_control_msg on your host PC's software and usbFunctionSetup on the ATtiny45's firmware is illustrated in Figure 2.

To generate the application program called volt.exe, type "gcc volt.c –lusb –o volt" in a Cygwin window. You may view the code on the *Circuit Cellar* FTP site.

libusb is an open-source project for the Linux, FreeBSD, NetBSD, OpenBSD, Darwin, Mac, and Windows operating systems (Windows 98, Win Me, Windows 2000, and Windows XP). Simple data transfer in a libusb protocol stack package can be accomplished using a usb_control_msg function for small data transfer. The usb_bulk_write and usb_bulk_read functions can be used for transferring large data. Definitions of the three functions are posted on the *Circuit Cellar* FTP site.

HOST DEVICE DRIVER INSTALLATION

When connecting the target USB device to a host PC for the first time, the Hardware Update Wizard will ask you to connect the new device driver. You must install the new device driver at your own risk.

First, connect your USB device to the host PC. The Hardware Update Wizard will ask you for a new device driver. Ignore this window for the moment. Instead, open a Windows Explorer window and double-click c:\cygwin\lib\inf-wizard.exe. In its window, select 0x1384 and click the Next button. Click the Next button again and save the file as usb-ko.inf. Now, go back to the Hardware Update Wizard window and select "install from a list or specific location." Check the "include this location in the search" box. Click on the Browse button and browse to c:\cygwin\lib\libusb. Finally, click on the OK button to continue. You have now successfully installed the generated device driver on your host PC. Two important files, libusb0.dll and libusb0.sys, have been installed by the Hardware Update Wizard.

TESTING

When you can successfully generate volt.exe and write da.hex files in the ATtiny45's flash memory with the fuse settings, type "volt 2.5" in the Cygwin window and measure the voltage at a MAX517's PIN1 with a voltmeter.

In this article, I described a simple programmable USB DAC featuring an ATtiny45 and a MAX517, which can be used as a programmable power supply. Because two-wire serial programming is used to set the D/A data in a MAX517, multiple channel DACs can be easily implemented with a MAX518 or a MAX519.

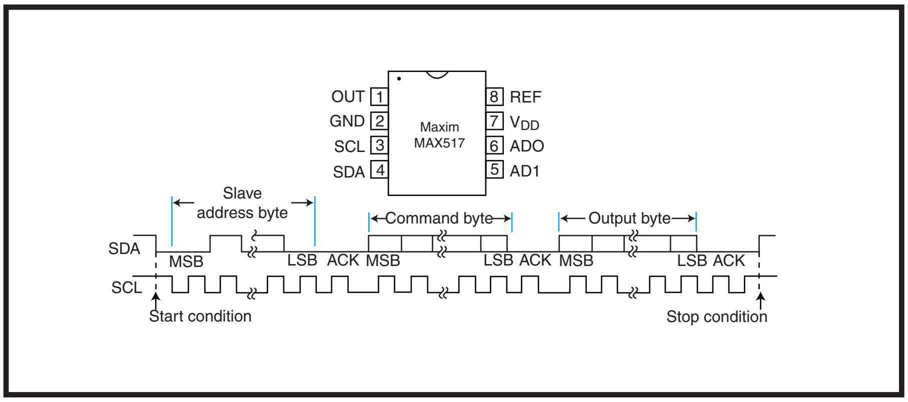

Figure 3—This is a MAX517 8-bit DAC programming timing chart.

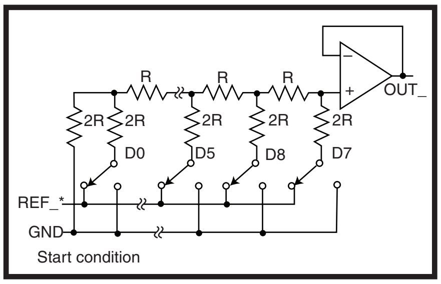

Figure 4—This is a simplified DAC diagram of a MAX517.

Yoshiyasu Takefuji (takefuji@sfc.keio.ac.jp) holds a PhD in Electrical Engineering from Keio University in Japan, where he is a tenured professor. He is also a tenured faculty member at Case Western Reserve University in Cleveland, OH. Yoshiyasu's design interests include computer architecture, neural computing, and computer security.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2008/213.

SOURCES

ATtiny45 Microcontroller
Atmel Corp.
www.atmel.com

AVR-USB
Objective Development Software
www.obdev.at/products/avrusb/index.html

Cygwin
Red Hat, Inc.
www.cygwin.com/setup.exe

WinAVR
SourceForge
http://winavr.sourceforge.net/download.html