Build your Own USB Devices

AVR Microcontrollers and the V-USB Library

Elliot Williams

elliot@littlehacks.org
github: hexagon5un

June 20, 2014

Outline

Outline

Outline

Rolling Your Own

Why?

- Parallel port is dead, serial port is emulated
- Compatibility
- Make what you want
- Fun

Why V-USB?

- Software-emulated USB? Limited to low-speed devices?
- Cheap, hackable, and you'll learn a lot
- Can run on very minimal hardware

Today

Projects

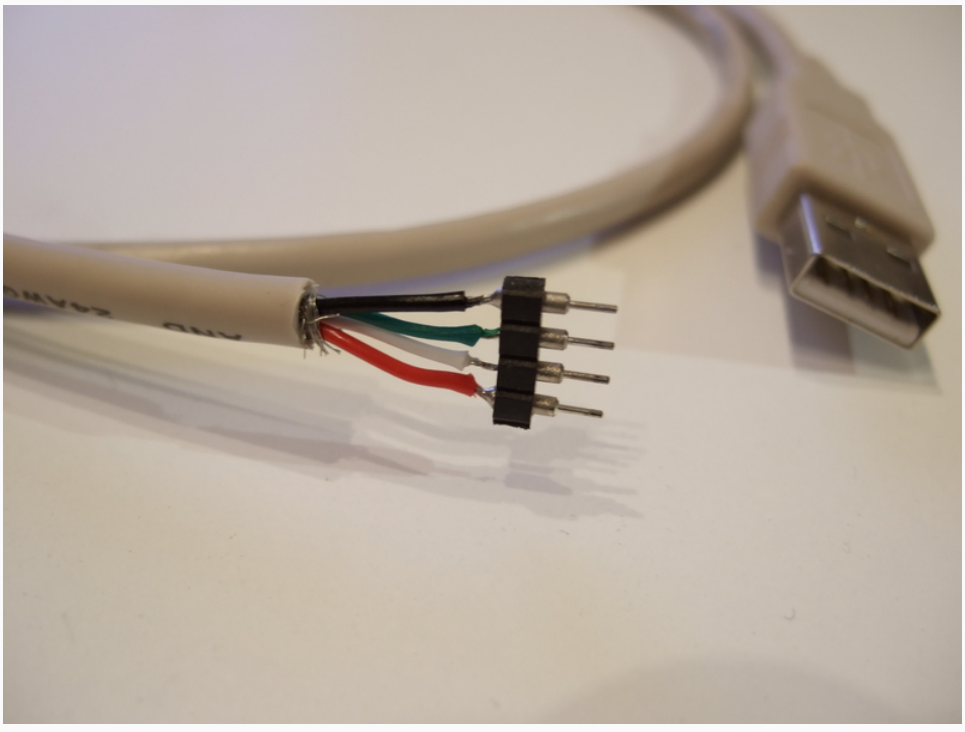
- USB Scrollwheel
- Weather Thing

Topics

- USB basics
- V-USB particulars
- Human Interface Devices
- Custom (Vendor-Specific) USB Commands

Outline

This is USB



Wiring: The Physical Layer

- Pull apart a USB cable: 4 colorful wires
- White & Green: `Data-` & `Data+`
  differential signaling: opposite voltage levels signify data
  `D+` $>$ `D-` $\rightarrow$ 1
  `D+` $<$ `D-` $\rightarrow$ 0
- The data line signalling voltages are **3.6V** and 0V
- Data lines are also used to detect devices:
  `D-` pulled high = low-speed device
  `D+` pulled high = full-speed or high-speed USB
- Baud rate (low-speed) 1.50Mb/s $\pm 1.5\%$
  NRZI, bit-stuffed (handled by V-USB library)
- Red & Black: 5V power supply and ground (V_{BUS})

The USB Protocol is **very** complicated.

Don't Panic!

- Spec is 600+ pages long – not including HID usage tables
- Hacker Approach: Just learn what we need to get job done.

Jargon

- *Human Interface Device* (HID):
  Standard USB device classes that people interact with
- *Host*: Your computer
- *Function*: The USB function on your device
- *Endpoint*: A data source or sink (on your device)
- *Control Endpoint*: Endpoint 0.
  Must be present, two-way, used to control the device.

USB is a Host-Controlled Bus

What does this mean?

- All transactions are initiated by the host:
  your device must wait to be asked
- "You will speak only when spoken to"
- *IN* and *OUT* are defined from the perspective of the host:
  data going *IN* is leaving your microcontroller

Enumeration: What happens when I plug my device in?

- Host resets device then asks for device descriptor
  assigns the device an address
  then resets again and asks for a whole bunch of descriptors
- (Almost) All of this is handled by V-USB library

USB Data Flow Modes: The Ones We Will Use

Four modes

- One mode per endpoint: what does the device need?

Control Transfers: configuration by host

- *IN*/*OUT* through Endpoint 0: mandatory
- Also for custom control protocols (Weather Thing)

Interrupt Transfers: high-priority, scheduled transfers

- Host periodically polls for data
- Device should have data ready when asked
- Intended for small/medium amounts of data (Mouse)

USB Data Flow Modes: The Ones We Won't Use

Bulk Transfers: large scheduled transfers

- Just like Interrupt, but with lower priority
- Meant for moving large amounts of data (Flash drive)

Isochronus Transfers: guaranteed bandwidth

- But no error-checking
- Meant for large amounts of non-critical data (Sound card)

USB Data Flow Modes: Summary

Control Endpoint / Transfer Mode:

Use for device enumeration (handled by V-USB library)
Use for "Vendor-Specific" custom control data

Interrupt Endpoint / Transfer Mode:

Use for keyboard/mouse/gamepad data

Outline

How Does V-USB Help?

What needed doing?

- Read incoming serial data, NRZI unpack
- Parse it, respond to our device address
- NRZI pack, send out serial data when able
- All this with tight timing
- Bookkeeping: Respond to host's requests for descriptors

What Do We Need To Do?

Configure

- Configure descriptors, endpoints, AVR pinouts, CPU speed
  `usbconfig.h`

Write Code

- Attach/reattach to fake the device resets
- Write functions to handle control data
  `usbFunctionSetup()`
- Get our interrupt data ready if we use it
  `usbInterruptIsReady()` and `usbSetInterrupt()`
- Update the V-USB system at least every 50 ms
  `usbPoll()`

Getting Started / Getting Oriented

- Download the zip file or clone the repo
  `https://github.com/obdev/v-usb.git`
- The subdirectory `usbdrv` contains files to link into your code:
  `usbdrv/usbdrv.o, usbdrv/usbdrvasm.o`
- `usbdrv/usbdrv.h` describes the API
  (you should read this to see all the possibilities)
- Copy the file `usbconfig-prototype.h` to your code
  directory and edit it to fit your particular configuration
- The `examples/` directory is full of goodies.
  My scrollwheel builds off of `hid-mouse`
  The weather gadget builds off of `custom-class`

Outline

HID!

Welcome to Infinity

- The variety of Human Interface Devices is mindblowing
- Don't believe me? 100 pages of "HID Usage Tables"
- "Usage" is USB for what the thing does
- Usages are stardard

Usage Pages

- So many usages, that they're grouped in pages
- Generic Desktop Controls, Simulation Controls, VR Controls Medical Equipment, Telephony, Camera

HID!

Usages

- Mouse, Joystick, Keyboard, Gamepad
- Tank, Submarine, Spaceship, Magic Carpet Simulation
  (Turret Position, Wing Flaps, Chaff Release, Dive Break)
- Head Tracker, Body Suit (points and positions for all joints)
- Golf Club (Sand Wedge, Stick Face Angle, Follow Through)
- Media controls, application launchers, 3D digitizers

Report Descriptors

Overview

- Nested structure:
  Usage Table $\rightarrow$ Usage $\rightarrow$ Lower-level Usage
- If your device has multiple usages, you can group them logically into *Collections*:
  a Graphic EQ is a collection of sliders
  our Mouse is a collection of X, Y, buttons, and scrollwheel
- You can collect things together by "Application", "Logical" layout, or "Physical" proximity
- Define your data structures:
  Buttons are binary bits (on/off),
  Axes need a full byte (or more!) each
  Scrollwheel is probably a signed integer

Report Descriptors

Resources

- USB.org's HID Page:
  http://www.usb.org/developers/hidpage/
  Download and read the "HID Usage Tables"
  Especially see the examples in the Appendix
- USB.org's HID Report Tool:
  http://www.usb.org/developers/hidpage#HID Descriptor Tool
- Good overview tutorial:
  http://eleccelerator.com/tutorial-about-usb-hid-report-descriptors/

Report Descriptors

... or Cheat

- Find a device that's like yours and rip off it's descriptor
- Modify slightly to match your situation, and you're done
- Making a mouse? You're not the first.
  No need to re-invent the scrollwheel!
- Using V-USB? Tons of documented projects out there.
- I'm ripping my descriptor off the V-USB example mouse, which is in turn copied from Logitech

Code or Go Home

Let's get started

- Making an Interrupt-mode device – a mouse
- Need to define the HID Report Descriptor: `descriptor.h` also create variable to store the report data
- Need to set up an Interrupt-OUT endpoint to talk to host: `usbconfig.h`
- Need to generate HID mouse reports so that they're ready when host asks: our main routine `scrollWheel.c`

Report Descriptor (Part I)

```
/* USB report descriptor, size must match usbconfig.h */
const PROGMEM char usbHidReportDescriptor[52] = {
        0x05, 0x01,                     // USAGE_PAGE (Generic Desktop)
        0x09, 0x02,                     // USAGE (Mouse)
        0xa1, 0x01,                     // COLLECTION (Application)
        0x09, 0x01,                     //   USAGE (Pointer)
        0xA1, 0x00,                     //   COLLECTION (Physical)
        0x05, 0x09,                     //     USAGE_PAGE (Button)
        0x19, 0x01,                     //     USAGE_MINIMUM
        0x29, 0x03,                     //     USAGE_MAXIMUM
        0x15, 0x00,                     //     LOGICAL_MINIMUM (0)
        0x25, 0x01,                     //     LOGICAL_MAXIMUM (1)
        0x95, 0x03,                     //     REPORT_COUNT (3)
        0x75, 0x01,                     //     REPORT_SIZE (1)
        0x81, 0x02,                     //     INPUT (Data,Var,Abs)
        0x95, 0x01,                     //     REPORT_COUNT (1)
        0x75, 0x05,                     //     REPORT_SIZE (5)
        0x81, 0x03,                     //     INPUT (Const,Var,Abs)
        0x05, 0x01,                     //     USAGE_PAGE (Generic Desktop)
        0x09, 0x30,                     //     USAGE (X)
        0x09, 0x31,                     //     USAGE (Y)
        0x09, 0x38,                     //     USAGE (Wheel)
        0x15, 0x81,                     //     LOGICAL_MINIMUM (-127)
        0x25, 0x7F,                     //     LOGICAL_MAXIMUM (127)
        0x75, 0x08,                     //     REPORT_SIZE (8)
        0x95, 0x03,                     //     REPORT_COUNT (3)
        0x81, 0x06,                     //     INPUT (Data,Var,Rel)
        0xC0,                           //   END_COLLECTION
        0xC0,                           // END COLLECTION
};
```

Report Descriptor (Part II)

```
/* This is the same report descriptor as seen in a Logitech mouse. The data
 * described by this descriptor consists of 4 bytes:
 *      .  .  .  .  . B2 B1 B0 .... one byte with mouse button states
 *     X7 X6 X5 X4 X3 X2 X1 X0 .... 8 bit signed relative coordinate x
 *     Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0 .... 8 bit signed relative coordinate y
 *     W7 W6 W5 W4 W3 W2 W1 W0 .... 8 bit signed relative coordinate wheel
 */

typedef struct{
        uchar   buttonMask;
        char    dx;
        char    dy;
        char    dWheel;
}report_t;
static report_t reportBuffer;
```

Changes to `usbconfig.h`

```
// Need Interrupt Endpoint
#define USB_CFG_HAVE_INTRIN_ENDPOINT    1

// Declare as HID Class
#define USB_CFG_DEVICE_CLASS        0
#define USB_CFG_INTERFACE_CLASS     3
#define USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH    52

// Identification Stuffs
#define  USB_CFG_VENDOR_ID       0x66, 0x66
#define  USB_CFG_DEVICE_ID       0xfe, 0xca

#define USB_CFG_VENDOR_NAME    'D', 'e', 'm', 'o'
#define USB_CFG_VENDOR_NAME_LEN 4
#define USB_CFG_DEVICE_NAME     'S', 'c', 'r', 'o', 'l', 'l', ' ' , 'W', 'h', 'e', 'e', 'l', '
#define USB_CFG_DEVICE_NAME_LEN 21
```

Finally, The Code

```
#include "scrollWheel.h"
/* This function used to respond to Vendor-Specific Control commands
   We're not using any, so just returning 0. */
usbMsgLen_t usbFunctionSetup(uchar data[8]){
        return 0;
}

int main(void){
        /* Reconnect and re-enumerate */
        usbInit();
        usbDeviceDisconnect();
        _delay_ms(250);
        usbDeviceConnect();
        sei();

        LED_DDR |= (1 << LED);
        BUTTON_PORT |= (1<<BUTTON);

        while(1){
                usbPoll();
                if(usbInterruptIsReady()){
                        if (bit_is_clear(BUTTON_PIN, BUTTON)){ /* button is pressed */
                                LED_PORT |= (1<<LED);
                                reportBuffer.dWheel = -1 * SCROLLFACTOR;
                        } else {
                                reportBuffer.dWheel = 0;
                                LED_PORT &= ~(1<<LED);
                        }
                        usbSetInterrupt((void *)&reportBuffer, sizeof(reportBuffer));
                }
        } /* endless while */
}    /* mainloop */
```

The Include File

```
// scrollWheel.h

#include <avr/io.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>  /* for sei() */
#include <util/delay.h>     /* for _delay_ms() */
#include <avr/pgmspace.h>   /* required by usbdrv.h */
#include "usbdrv.h"
#include "descriptor.h"                /* HID descriptor */

#define LED_DDR         DDRB
#define LED_PORT        PORTB
#define LED             PB0

#define BUTTON          PD3
#define BUTTON_PORT     PORTD
#define BUTTON_PIN      PIND

#define SCROLLFACTOR    1  // can increase for faster scrolling
```

Circuit Concerns

Power

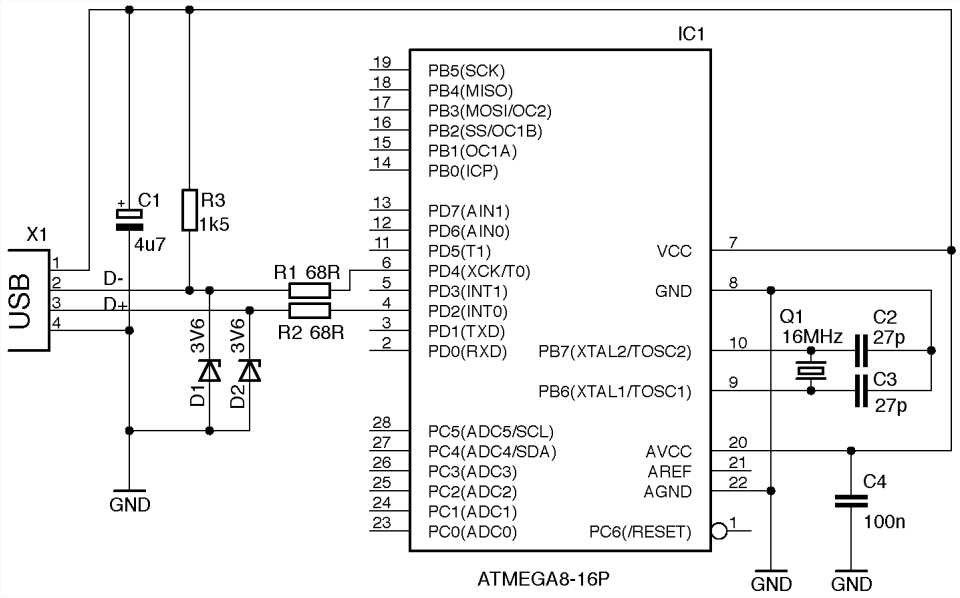
- V_{BUS} is a 5V power supply,
  but we need to transmit using 3.6V signals
- Devices *can* be self-powered: Run it on a 3.6V battery. Done.
- We all like bus-powered devices
  $\rightarrow$ we'll have to work around the two voltages
- Two methods:
  convert 5V down to 3.6V first, run AVR at 3.6V
  run AVR at 5V but limit output signal voltage to 3.6V
- Good discussion of pros/cons at
  `http://vusb.wikidot.com/hardware`
- Choose to run AVR at 5V:
  use zener diodes to limit output voltage

Circuit Concerns

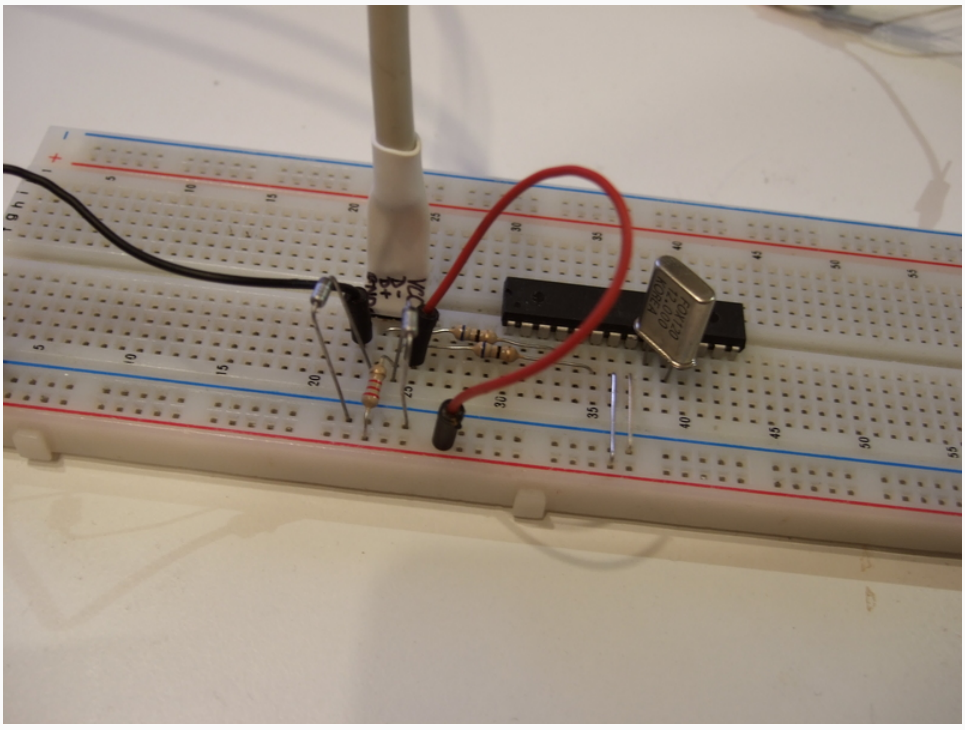
Timing

- Baud rate with 1%ish precision: 1.50Mb/s
- Easy way: Use an external crystal for the AVR's CPU clock
  12, 12.8, 15, 16, 16.5, 18 or 20 MHz
- Cheap, kludgy way: run CPU clock out of spec
  Calibrate the AVR's 8 MHz internal clock using USB
- Viable option with ATTiny 45, 85, 261 chips
  have a high-speed (64MHz) internal oscillator
  (Adafruit Trinket, Sparkfun AVR Stick, Digispark)

Circuit Diagram

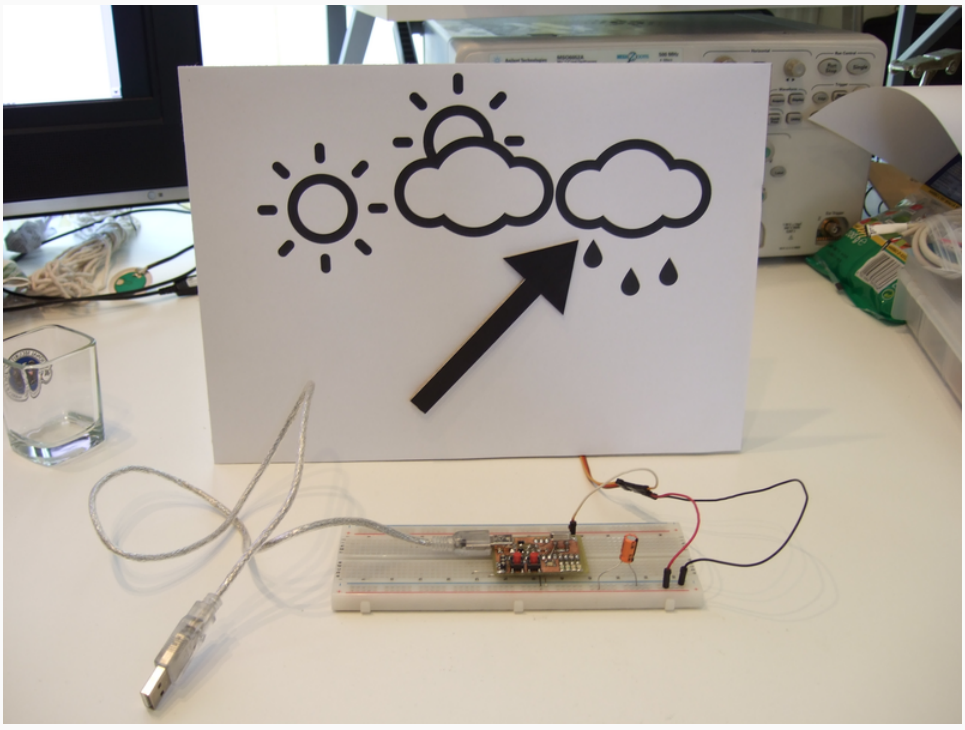


Ghetto USB Hookup



Outline

The Weather Thing



Making a Custom Device

Advantages

- Don't have to write a HID Report Descriptor
- Can do non-standard things:
  Control servomotors, for instance

Disadvantages

- Requires writing host-side code
- Have to define and handle the commands yourself
- Requires a bit of detailed know-how: Control Transfers

Control Transfers

What you need to know

- Three Packet Types: Setup, Data, Acknowledgement
- Starts with Host sending a Setup packet:
  specifies type, direction, and amount of data that follows
- Optional Data packet(s) can go either way (IN or OUT):
  Device or Host can send the data packets
- If Host sent data, V-USB handles the ACK automagically

Setup Packet

The heart and soul of Control Transfers

- 8 Bytes
- `bmRequestType`:
  Includes transfer direction (IN/OUT),
  Control Type (Standard, Class, Vendor-specific)
  Recipient (Device, Interface, Endpoint, Other)
- `bRequest`: The request command itself – one byte
- `wValue` and `wIndex`:
  Two words (each 2 bytes) of request options
- `wLength`: Length of the optional data stage (in bytes)

Setup In Detail

`bmRequestType`

- IN/OUT matters. Remember relative to host.
- For custom commands, select "Vendor-specific"
- When V-USB gets a custom Setup packet, it calls the function `usbFunctionSetup`.

`bRequest`

- Can specify 256 commands for the Device
- Host and Device agree on what the commands are / do
- For OUT data, you can send four bytes of arguments without even using the data stage (`wValue` and `wIndex`)

Control Transfers

Using the Data Stage

- To get data from the AVR to the host (IN) you need to specify `wLength`
- If you need to send more than 4 bytes OUT, you need to specify `wLength`
- V-USB calls the functions `usbFunctionRead()` and `usbFunctionWrite()` to handle these data transactions ("read" reads data into the host)
- Gotcha: need to enable read/write functions in the config
- See the example `hid-data` and `usbdrv.usbdrv.h` for details

Enough Talk, Let's Code

Host-Side: Python

Setup

- Install Python and the `pyusb` library
- Win/Mac will also need `libusb`
- See `https://github.com/walac/pyusb` for install help
- Quick test:

```
import usb.core
for device in usb.core.find(find_all=True):
    print "%04x:%04x" % (device.idVendor, device.idProduct)
```

  Prints out VID:PID of all devices attached (in hexadecimal)

Host-Side: Python

Code Outline:

- Fetch weather forecast data over Internet
- Convert to pulse length for servomotor
- Push pulse length over USB to AVR using Control Transfer

Control Transfer:

- Need to build `bmRequestType`:
  OUT transfer, Vendor-specific
- Set `wValue` to the desired pulse length
- No data. Done.

Host-Side: Internet Part

```
## Stripped-down Example of Computer-VUSB Device Communication

import urllib2
import json

## First the Internet side
forecastURL =  'http://api.openweathermap.org/data/2.5/forecast/daily'
forecastURL += '?q=Munich,DE&cnt=2&mode=json&units=metric'

webpage = urllib2.urlopen(forecastURL).read()
j = json.JSONDecoder()
weather = j.decode(webpage)

## The following navigates through the nested list/dict
##  structure that's returned
tomorrowsWeather = weather['list'][1]['weather'][0]['main']

## Result is a string: ('Clear', 'Clouds', or 'Rain')
print tomorrowsWeather

## Convert weather string to servo pulse length
pointerDict = {'Clear':2500, 'Clouds':1700, 'Rain':1000}
servoPulseLength = pointerDict[tomorrowsWeather]
```

Host-Side: USB Part

```
import usb.core
import usb.util
import time

commandDict = {'setServo':0x42, 'relax':0x01}

requestType = usb.util.build_request_type(
        usb.util.CTRL_OUT,
        usb.util.CTRL_TYPE_VENDOR,
        usb.util.CTRL_RECIPIENT_DEVICE
        )

dev = usb.core.find(idVendor=0x6666, idProduct=0xbeef)

dev.ctrl_transfer(  ## Turn servo on, set position
        bmRequestType = requestType,
        bRequest      = commandDict['setServo'],
        wValue        = servoPulseLength
        )

time.sleep(1)

dev.ctrl_transfer(  ## Turn servo off, save power
        bmRequestType = requestType, bRequest = commandDict['relax'] )
```

AVR Side Code

Writing Custom Commands

- All the action here is in the Control transfer handling
- When the AVR receives a Control transfer, the standard responses are handled by the V-USB library
- When the AVR receives a Control transfer in Vendor-specific mode, it passes the setup packet along to the user-implemented function `usbFunctionSetup`.
- Note that we first cast the data as a `usbRequest_t` type, which lets us read in the individual

AVR Code Control Setup

```
/* WeatherThing
   A minimal V-USB Custom Control Transfer Demo */

#define CMD_SET_SERVO       0x42
#define CMD_RELAX           0x01

usbMsgLen_t usbFunctionSetup(uchar data[8]) {
        usbRequest_t    *rq = (void *)data;
        if(rq->bRequest == CMD_SET_SERVO){
                setServo((rq->wValue).word);
        } else if (rq->bRequest == CMD_RELAX){
                servoOff();
        }
        return 0;               /* sets return data length: no data stage */
}
```

AVR Code Main Loop

```
/* WeatherThing
   A minimal V-USB Custom Control Transfer Demo */

int main(void) {

        usbInit();
        usbDeviceDisconnect();  /* enforce re-enumeration */
        _delay_ms(250);
        usbDeviceConnect();
        sei();

        initServo();
        LED_PORT_DDR |= _BV(LED_BIT);   /* make the LED bit an output */

        for(;;){                /* main event loop */
                usbPoll();
        }
}
```

The Build

(Such As It Is)

- Grabbed some weather icons in SVG off the web (Thanks Alessio Atzeni!)
- Printed the icons and a pointer out on paper, spray-glued to cardboard
- Hot-glued arrow to a servo horn, and the servo motor to the back of the sign
- Connect servo power up to AVR, adding an extra capacitor
- The control pin to the servo hooked up to AVR PB1: a direct-out from the 16-bit Timer1
- That's it!

Demo

Comments

Custom commands are easy

- Intimidating at first?
- Less work here than in figuring out an HID Usage report
- You have to keep track of the command dictionary on the host application and in the AVR
- You have to keep track of what data's being passed
- But then writing one or two functions to handle it all is easy
- The cost is that you have to write host code, but that's not that bad either?

Summary

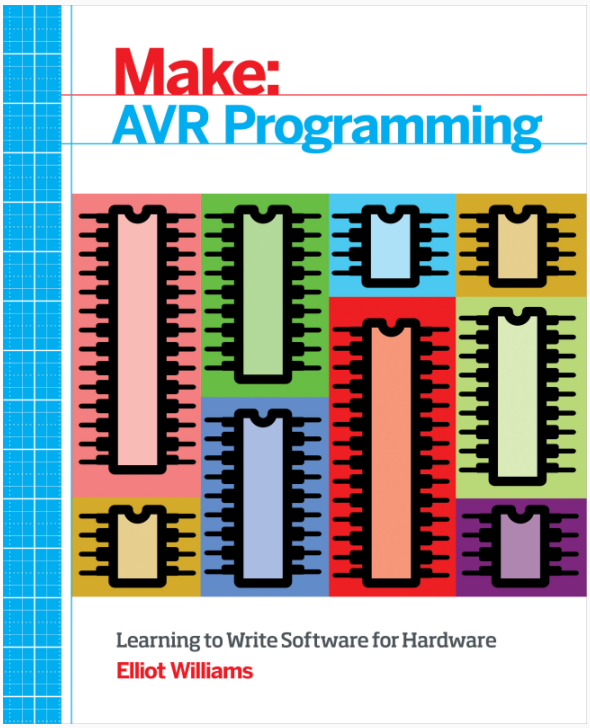
What have we learned?

- How to make HID and Custom USB devices,
  and write host-side code when necessary (whew!)
- How to hook up a regular AVR chip as a simple USB device
- Far too much detail about USB – I hope it serves you well

Resources

- V-USB website and Wiki
  http://www.obdev.at/products/vusb/index.html
- All code, slides, etc
  http://www.githib.com/hexagon5un/vusbWebinar
- My AVR Site: www.littlehacks.org
- Hackaday, Make Blog, Sparkfun, LadyAda for inspiration
- "USB Made Simple"
  http://www.usbmadesimple.co.uk/index.html
- "USB in a Nutshell"
  http://www.beyondlogic.org/usbnutshell/usb1.shtml

Oh yeah, I wrote a book



Questions?

The End

Outline