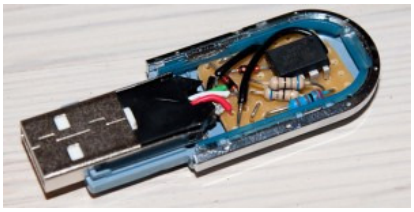
Mar 03 2012

DIY USB password generator

Electronics, V-USB tutorials Add comments

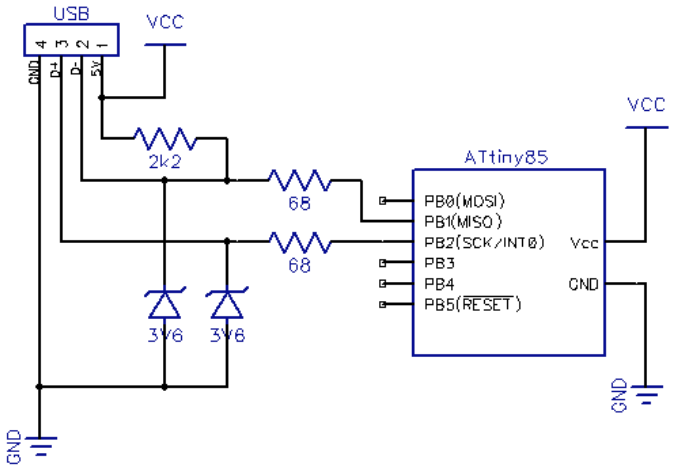


Having done half a dozen V-USB tutorials I decided it's time to whip up something cool. As USB keyboards were an area untouched, I decided to make a small USB HID keyboard device that types a password stored in EEPROM every time it's attached. A new password can be generated just by tabbing CAPS LOCK a few times (4 times to start password regeneration and one tab for each password character generated, 10 is the default password length). Below you can see the device in action:

The place I work at requires me to change my password every few months so this would be one way to skip remembering a new password altogether (as long as I remember to write it down before regenerating a new one so password can be changed :).

What is inside?

The device is powered with a simplified version of the hardware I used in my ATtiny85 USB tutorial – I stripped away the LCD, reset pullup and both capacitors. If you're better in cramming components inside enclosures I suggest adding at least a 0.1 uF capacitor between VCC and GND, but it seems to work fine even without it:



The enclosure was graciously donated by an old 512 MB flash drive. I couldn't make myself to break the USB connector from the circuit board inside, so I stripped appart a short USB cable instead (shown on left):

Recent Writings

- DIY USB password generator
- DIY resistor folder for 9.90€
- V-USB: Outputting Data with usbFunctionRead()
- 7 Segment Multiplexing With ULN2003 & PNP Transistors
- Default feed not disabled with Suffusion
- V-USB with ATtiny45 / ATtiny85 without a crystal

Follow Me

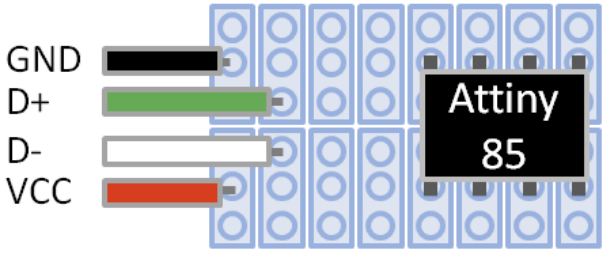
Archives

- March 2012 (1)
- February 2012 (11)
- January 2012 (3)
- December 2011 (1)
- March 2011 (1)
- July 2009 (1)
- May 2008 (1)
- April 2008 (1)
- February 2008 (1)
- January 2008 (7)

Check these out

- Code Igniter
- jGoBoard
- My homepage
- Slashdot

After some thinking and iterative soldering, I managed to cram everything on a tripad veroboard with 2×8 pads with the following initial setup:



I soldered the connector first, then the zener diodes, then resistors and jumpers, and finally VCC, GND and the ATtiny itself. I used the following tricks to make all ends meet:

- D+ zener diode goes to the pad under ATtiny that is connected to GND pin
- After the D- zener diode, only 1 pad is left for 2k2 pullup and 68 ohm resistor, so I used a jumper wire to the next pad
- 2k2 pullup goes to a pad connected to ATtiny VCC
- VCC goes to the pad under the ATtiny using a black jumper wire
- I soldered the D+ 68 ohm resistor to a wrong tripad, so I used another jumper wire just barely visible behind the top left black jumper wire for GND

I was pretty satisfied the result and the fact that it actually worked! The board did not initially fit into the very snug space in the plastic enclosure, so I had to use a Dremel to trim its insides a bit, but after that, everything snapped right back (click for larger versions):

Software

The device presents itself to the computer as a USB HID keyboard. To enable communication to the device, it is a boot-compliant keyboard that can receive LED status changes from the computer. HID descriptor is from Frank Zhao's USB business card example and I also looked at Frank's code to understand how LED state is sent to the device (in short, PC sends a control message with 1 byte of data, the LED state bit mask).

The code is mostly based on my USB HID mouse example except for the `usbsconfig.h` and HID descriptor changes required to implement a boot keyboard. I've documented the code but here are some highlights if you want to understand it better:

- `PASS_LENGTH` defined in the beginning controls the length of generated passwords
- `SEND_ENTER` can be defined to 1 if you want the device also to send ENTER after typing the keyboard
- `measuring_message` and `finish_message` contain the messages that are displayed when generating / saving a new password
- `buildReport()` is called by the program main loop to send keypresses to PC one by one – it translates characters in `messageBuffer` to USB key codes on the fly
- `usbFunctionWrite()` is implemented to receive the 1-byte LED state from PC – it calls `caps_toggle()` function every time the LED state changes
- `generate_character()` is used to return random keypresses – it is currently written to return alphanumerics, hyphen and underscore (64 symbols make it simple to select one so each has equal chance of being selected without additional logic)
- `caps_toggle()` does the caps-lock counting and password generation/saving

I've packed the source files with the schematic, critical pictures and a Makefile. In addition to "make flash" you of course need to update the fuse bits to use the PLL clock source – see details from my previous tutorial for that. I also very strongly recommend testing the device using a breadboard before soldering it, because otherwise reflashing will be a **major** pain.

And of course, if you build it, try it at your own risk – and remember that once you reprogram the password, nothing will be able to restore it. I recommend storing passwords generated with the device to a safe place just to be sure.

Posted by jokkebk at 13:00 Tagged with: ATtiny, AVR, diy, hack, password, usb, V-USB

Leave a Reply

Name (required)

E-mail (required)

URI

Your Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Submit Comment

DIY resistor folder for 9.90€