AVR USB Bootloader

Programare microcontroler pe USB

Introducere - metode de programare

Microcontrolerele AVR execută cod din memoria Flash internă (numită şi memorie program). Codul se asamblează sau compilează pe PC, rezultând în final un fişier .hex ce conţine o imagine a memoriei program (cum ar fi o imagine de BIOS pentru PC). Această imagine trebuie scrisă în memoria microcontrolerului (operaţie denumită "programare") printr-una din următoarele metode:

- la fabrică. După ce codul a ajuns la o versiune stabilă, se trimite la producător, care va vinde controlerele gata programate. Numai pentru producţie de serie mare (mii de bucăţi).

- programarea cipului în modul paralel (cu 12V aplicaţi pe reset (nu puneţi niciodată mai mult de 5V pe orice alt pin!), datele pe portul B şi diverse comenzi pe A şi D, ceas aplicat pe XTAL 1) - necesită multe fire, se efectuează cu un programator specializat prevăzut cu o serie de socluri (pentru toate tipurile de microcontrolere - unele au 8 pini, altele 20, altele 28, altele tot 40 dar cu semnalele dispuse altfel). Cipul se introduce în soclul corespunzător, se programează rulând comenzile specifice pe PC, apoi se scoate şi se pune pe placa unde va funcţiona. Este o metodă dezavantajoasă pentru dezvoltare, implicând extracţia şi inserţia repetată a cipului, o activitate ce necesită atenţie şi îndemânare şi care oricum va duce în final la ruperea pinilor.

- programarea în circuit / în sistem. Se realizează fără ca cipul să trebuiască scos de pe placă. În mod normal pinii controlerului ce fac parte dintr-un port (PORTA, B, C, PORTD şi mai departe pentru cipurile mai mari) sunt sub controlul direct al programului sau al dispozitivelor periferice integrate (controlate tot de program). În reset (pinul /RESET tras la nivel logic 0) 3 dintre pini (botezaţi de obicei SCK, MOSI, MISO - serial clock, master out slave in, master in slave out, sau PDO şi PDI - program data out, in) au funcţie de programare serială. Protocolul fizic folosit este SPI (serial peripheral interface), un protocol serial sincron (cu ceas) folosit şi în funcţionarea normală pentru a "vorbi" cu diverse circuite integrate. Protocolul logic este definit în datasheet/manual. Softul de pe PC (PonyProg, uisp, avrdude etc.) se ocupă de toate detaliile protocolului, putând efectua operaţii de scriere şi citire asupra memoriei program a microcontrolerului. Este cel mai folosit mod de programare deoarece necesită un hardware deosebit de simplu: un cablu cu 5 fire pentru portul paralel, respectiv acelaşi cablu + câteva rezistenţe şi diode pentru portul serial. Problema este că în ultima vreme porturile seriale şi paralele încep să dispară, de unde nevoia de a folosi alt mod de programare. **Majoritatea convertoarelor USB-serial NU sunt capabile să programeze controlerul** în modul SPI, deoarece nu implementează toate semnalele auxiliare RS-232 (cum ar fi RTS şi DTR), ci doar TXD şi RXD (comunicaţia serială propriu-zisă).

- programarea în circuit prin portul de depanare JTAG (Joint Test Action Group). Se foloseşte un dispozitiv de depanare/emulare în circuit (spre exemplu Atmel JtagICE) care se conectează pe 4 pini (TCK, TMS, TDI, TDO). Spre deosebire de interfaţa de programare SPI, care este activă numai în reset, interfaţa JTAG este tot timpul activă dacă s-a configurat din fuse-uri (JTAGEN, OCDEN). Aceasta permite, pe lângă citirea şi scrierea memoriei program, şi depanarea programului ce rulează (cu AVR Studio sau avarice+gdb), şi inspectarea şi comanda porturilor şi perifericelor. De aici noţiunea de emulare în circuit: citirea şi comanda pinilor controlerului prin protocolul JTAG de către un program rulând pe PC. **Atenţie**: Portul JTAG ocupă 4 pini/biţi (de pe portul C la ATmega16) care nu pot fi folosiţi de program dacă este activat (şi vine activat din fabrică). Trebuie şters fuse-ul JTAGEN (setat pe 1) dacă doriţi ca programul să poată folosi pinii respectivi.

- **autoprogramarea folosind un bootloader.** Bootloader-ul este o mică bucată de cod aflată (pe AVR) la sfârşitul memoriei program, care are posibilitatea să scrie în restul memoriei. În mod normal (din fabrică) execuţia programului după ieşirea din reset începe de la adresa 0, unde există un jump către codul propriu zis (codul de aplicaţie). Este însă posibil, prin programarea fuse-ului BOOTRST (setarea lui pe 0) să se înceapă execuţia din zona de boot. Dimensiunea zonei de boot e dată de fuse-urile BOOTSZ. Pentru ATmega16:

BOOTSZ1	BOOTSZ0	Dimensiune boot	Adresă boot
1	1	256 B	16384 - 256 = 0x3F00
1	0	512 B	16384 - 512 = 0x3E00
0	1	1 kiB	16384 - 1024 = 0x3C00
0	0	2 kiB	16384 - 2048 = 0x3800

În tabelul din datasheet valorile sunt date divizate cu 2, deoarece memoria program e organizată în cuvinte de 16 biţi. gcc şi binutils pe de altă parte lucrează cu adrese de octet.

Utilitatea acestei zone de boot constă în faptul că ea poate conţine orice fel de cod, care poate folosi porturile şi perifericele microcontrolerului în orice mod. Astfel, se poate folosi orice interfaţă fizică pentru a aduce codul aplicaţie în memoria program a microcontrolerului prin intermediul bootloader-ului.

O problemă constă în transferul execuţiei de la bootloader la aplicaţie. Interfaţa fizică folosită de bootloader pentru transferul programului poate fi folosită de aplicaţie la altceva, şi aplicaţia nu trebuie să aştepte comenzi legate de programare pe vreo interfaţă, ci să-şi facă treaba. Trebuie să existe o separare totală între aplicaţie şi boot, pe motive de portabilitate şi generalitate. Deci bootloader-ul, care porneşte la reset, are mai multe variante:

- să citească valoarea logică a unui pin, care dacă e pus la masă printr-un jumper înseamnă activarea modului de programare şi aşteptarea de comenzi de la PC, în caz contrar transferându-se imediat controlul aplicaţiei (cea mai folosită metodă)

- să aştepte un timp (2-5 secunde) comenzi, rulând apoi aplicaţia dacă nu primeşte nimic (o metodă bună dacă în funcţionare normală nu apar conflicte cu dispozitivele conectate pe pinii folosiţi la programare în primele secunde după pornire)

- să aştepte comenzi, printre care o comandă de lansare în execuţie a aplicaţiei.

Bootloaderul trebuie încărcat în memoria program printr-una din metodele de mai sus (the chicken and egg problem). Deci trebuie să aveţi acces la un programator SPI sau JTAG pentru a scrie boot-ul şi a configura fuse-urile, după care puteţi folosi o metodă mai comodă. O parte din aceste metode sunt descrise în continuare.

Bootloader serial RS-232

Cea mai simplă interfaţă este cea serială asincronă, care foloseşte modulul USART (universal async or sync receiver transmitter) al microcontrolerului. Pinii RXD şi TXD intră într-un convertor de nivel MAX232 care translatează nivelele logice din +5V, 0 în -10, +10 V şi invers, conform standardului RS-232, folosit pe portul serial al PC-ului. Această metodă **poate** fi folosită cu convertoare USB-serial din comerţ, deoarece nu sunt necesare semnalele auxiliare. Portul serial cu MAX232 de pe placă îl aveţi deja probabil.

Cod pentru astfel de bootloader-e se găseşte pe net; vor apărea şi aici nişte variante recomandate.

Bootloader Ethernet

Specific pentru proiectele care folosesc o conexiune Ethernet în cadrul aplicaţiei. Aceasta se poate realiza cu un controler Ethernet dedicat, spre exemplu un ENC28J60 conectat la AVR pe portul SPI (care în afară de reset e controlat de software). Bootloaderul presupune iniţializarea controlerului, setarea unor adrese MAC şi IP şi ascultarea comenzilor pe un port UDP.

Bootloader USB

Microcontrolerele AVR sunt suficient de rapide pentru a implementa un dispozitiv USB complet în software. Un bootloader pe USB încape în mai puţin de 2kB de memorie program (1k instrucţiuni).

Despre USB

Despre USB se poate citi aici.

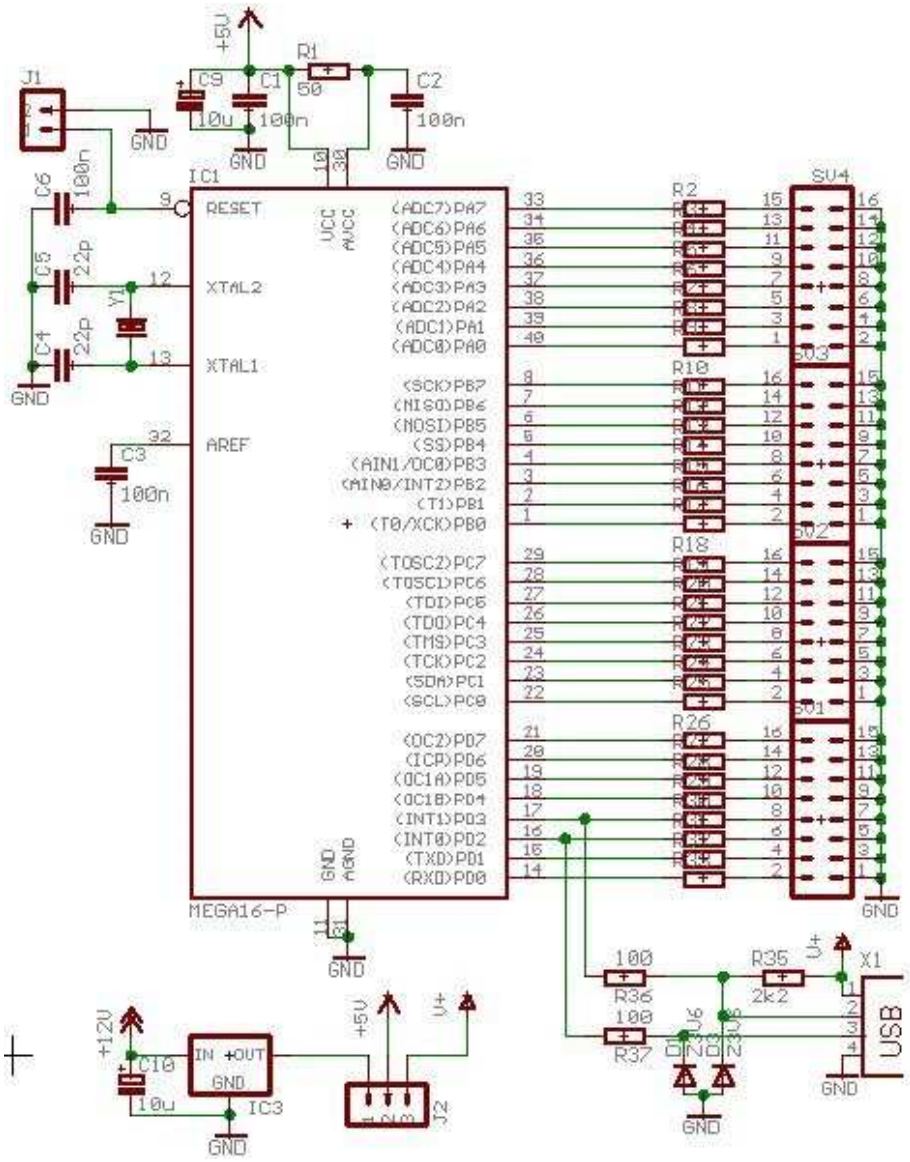
Conectorii şi cablurile USB au 4 pini / conductori: 2 pentru alimentarea dispozitivelor şi 2 pentru comunicaţie. Alimentarea înseamnă nominal 5V faţă de masă, care pot scădea până la 4.4V dacă se trage mult curent. 500mA este limita, peste trebuie alimentator extern. Deci microcontrolerul se poate alimenta direct de pe cele 2 linii, cu un condensator de 10uF în paralel. Datele se transmit diferenţial pe cele 2 linii D+ şi D-, adică au nivele logice opuse (cu excepţia unor stări speciale), astfel încât pot fi citite cu un amplificator diferenţial spre a rejecta diversele interferenţe ce pot apărea pe modul comun. Transmisia este half-duplex, adică se poate transmite în ambele direcţii dar nu simultan. Protocolul este master-slave, adică un dispozitiv nu iniţiază niciodată comunicaţia, ci doar host-ul (PC-ul). Evident, biţii se transmit serial şi asincron. Se foloseşte un cod de linie specific (NRZI with bit stuffing) pentru a asigura suficiente tranziţii 0-1 şi 1-0 la recepţie (altfel s-ar pierde sincronizarea).

Un dispozitiv îşi anunţă prezenţa către hub (hub-ul este cel care distribuie mai multe porturi downstream către un port upstream; există cel puţin un root hub integrat în host) printr-o rezistenţă pull-up pe una din liniile de date (explicaţii suplimentare găsiţi pe beyondlogic la linkul de mai sus), indicând astfel şi viteza de comunicaţie.

USB pe AVR

Pe AVR nu putem implementa direct decât comunicaţia low-speed, la 1.5 Mb/s. Această viteză e relativ mică faţă de posibilităţile cablului, deci nu sunt necesare circuite de transmisie şi recepţie specializate, fiind suficiente cele ale portului AVR. Singura precizare dpdv electric este că liniile de date folosesc nivele logice de 0 şi 3.3 V, deci a pune 5V pe ele nu este indicat. Astfel, ori se alimentează AVR-ul la 3.3 V cu un stabilizator low-dropout, fie se alimentează la 5 şi se folosesc 2 diode Zener de 3.3 sau 3.6V pe liniile de date spre a limita tensiunea de ieşire. AVR-ul poate citi corect nivele logice high de 3.3V chiar când este alimentat la 5V, deoarece intrările sale au prag logic tip TTL.

O schemă de conectare a mufei USB la AVR este următoarea:

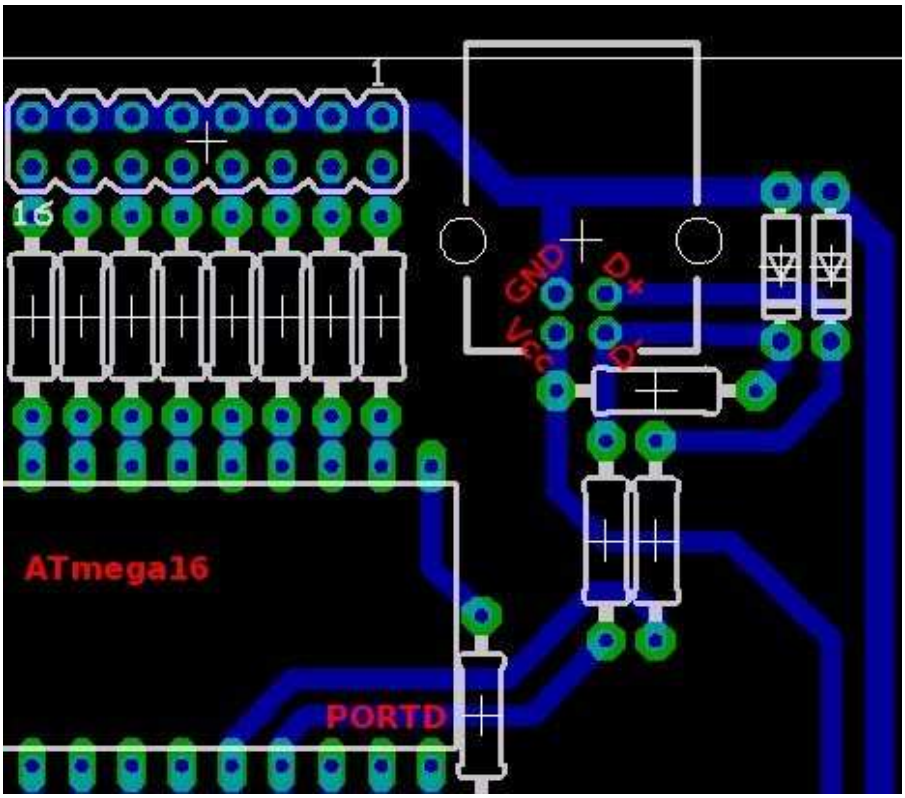


Interesează în particular partea din dreapta-jos. Restul reprezintă condensatoarele de decuplare de pe alimentări (cât mai aproape de pinii de alimentare!), cristalul pentru oscilatorul de ceas, nişte pin headers (conectori) pentru toate porturile, protejate la supratensiune prin rezistenţe de 1k, respectiv un stabilizator de tensiune de 5V (7805) ce permite alimentarea, selectabilă prin jumper, de la o tensiune de 8-15V sau direct de pe USB.

Liniile de date ale portului USB se conectează la AVR prin rezistenţe de circa 100 ohmi (pot fi şi mai mici, s-a testat cu 50, 68 şi 100, probabil şi un pic mai mari). Diodele Zener de 3.6 sau 3.3V trebuie neapărat puse pentru a limita tensiunea aplicată liniilor de date. Se foloseşte o rezistenţă de 2.2k (merge şi mai mică, până în 1.5k) pentru a semnala prezenţa dispozitivului pe bus şi a indica viteza de 1.5 Mb/s. În loc de alimentare, rezistenţa se poate conecta la unul din pinii AVR-ului, permiţând conectarea şi deconectarea logică de la bus din software. Dar se pierde un pin, care trebuie ales cu grijă pentru a nu avea nevoie de el în aplicaţie.

Din constrângeri software, linia D+ (pinul 3 al mufei USB) trebuie să ajungă pe pinul INT0 al AVR-ului. Linia D- trebuie să ajungă pe orice alt pin al aceluiaşi port. Într-o versiune mai veche trebuia ca una din linii să ajungă şi pe bitul 0 al portului, dar nu mai e cazul. Nu luaţi în seamă schemele de pe net care zic să puneţi D+ pe INT0 şi pe alt pin simultan, pierdeţi un pin degeaba.

Un exemplu de layout PCB (printed circuit board) cu mufă USB tip B (pătrată):



Pentru mufa USB tip A (dreptunghiulară lungă) pinii sunt în linie:

Poate fi şi ea montată pe placă, dacă o lăsaţi în aer folosiţi cablu torsadat şi pentru date.

Driverul USB este dezvoltat de Objective Development şi distribuit sub licenţă GNU GPL. Aceasta înseamnă că dacă dezvoltaţi un produs integrând acest driver şi îl distribuiţi, trebuie să distribuiţi şi codul sursă al aplicaţiei de pe microcontroler ce integrează driverul sub aceeaşi licenţă.

Folosirea driverului în aplicaţiile proprii este foarte simplă, constând în includerea directorului usbdrv în

proiect şi apelarea funcţiei usbPoll la intervale regulate. Driverul vă va apela apoi funcţiile usbFunctionSetup, usbFunctionWrite şi usbFunctionRead după cum este configurat din usbconfig.h. Vedeţi exemplele date de obdev (reference implementations) şi proiectele altora.

Timing-ul pe USB e strict (este un protocol asincron, adică ceasul nu este trimis de la emiţător la receptor pe altă linie simultan cu datele, ci receptorul trebuie să se sincronizeze cu emiţătorul bazându-se doar pe stream-ul de date, cunoscând faptul că durata unui bit este foarte precisă - la fel ca pe portul serial). În consecinţă codul de nivel jos al driverului este scris în limbaj de asamblare pentru diferite frecvenţe ale ceasului microcontrolerului, ţinând cont de timpul de execuţie al fiecărei instrucţiuni. Frecvenţele permise în versiunea curentă sunt: 12 MHz, 15 MHz, 16 MHz şi 16.5MHz. Primele trebuie generate de un cristal (foarte precis), ultima poate fi generată şi de un oscilator imprecis cum ar fi oscilatorul RC intern prezent pe anumite microcontrolere. Frecvenţa trebuie specificată în usbconfig.h pentru a compila o anumită variantă de driver. Nu uitaţi să setaţi fuse-urile pentru a folosi oscilatorul cu cristal extern. Pe net găsiţi multe proiecte cu versiunea veche de driver, care nu suportă decât 12MHz. Se poate înlocui cu versiunea nouă dintr-un reference implementation al obdev relativ simplu.

Un bootloader pe USB

Un bootloader pe USB găsiţi aici. Nu a mai fost updatat de mult, aşa că i-am făcut nişte modificări (updatat driverul, schimbat din ATmega8 în ATmega16). Arhiva o găsiţi în josul paginii.

În arhivă se găseşte în directorul firmware un fişier main.hex ce trebuie încărcat în memoria program printr-o metodă standard (spre exemplu programare SPI folosind un port serial sau paralel). Codul este compilat pentru ATmega16, cu cristal de 16 MHz, cu conexiunile USB din schema de mai sus. Pentru altă arhitectură, schimbaţi în Makefile denumirea, adresa bootloader-ului (octeţi memorie minus 2048, în hexa) şi eventual fuse-urile. Schimbaţi în usbconfig.h frecvenţa şi pinii. Daţi make clean şi make de la consolă în directorul firmware pentru a regenera fişierul main.hex. Trebuie de asemenea schimbat în linia AVRDUDE numele programatorului şi portul pe care e conectat pentru a putea programa controlerul cu comenzi de tipul make flash, make fuse. Altfel puteţi folosi alt soft de programare.

Fuses (din Makefile):

```
# Fuse high byte:
# 0xc0 = 1 1 0 0   0 0 0 0 <-- BOOTRST (boot reset vector)
#        ^ ^ ^ ^   ^ ^ ^------ BOOTSZ0 (2kB
#        | | | |   | +-------- BOOTSZ1      boot size)
#        | | | |   + --------- EESAVE  (preserve EEPROM over chip erase)
#        | | | +-------------- CKOPT   (full output swing)
#        | | +---------------- SPIEN   (allow serial programming)
#        | +------------------ JTAGEN  (disable JTAG, save 4 pins)
#        +-------------------- OCDEN   (disable on-chip debug)
# Fuse low byte:
# 0x3f = 0 0 1 1   1 1 1 1
#        ^ ^ \ /   \--+--/
#        | |  |       +------- CKSEL 3..0 (external >8M crystal)
#        | |  +--------------- SUT 1..0   (long startup time)
#        | +------------------ BODEN      (BrownOut Detector enabled)
#        +-------------------- BODLEVEL   (4V)
```

Atenţie: pentru că sunt fuse-uri ("siguranţe fuzibile" emulate cu memorie eeprom) sunt activate dacă sunt arse (zero). Conform datasheet: 1 = unprogrammed, 0 = programmed. În anumite softuri trebuie bifat un checkbox pentru a face bitul respectiv 0. Să nu puneţi invers biţii, altfel puteţi rămâne fără ceas (dezactiva accidental oscilatorul) - nu mai merge nimic.

Utilizare

O dată scris bootloader-ul în flash-ul microcontrolerului şi setate fuse-urile, se poate vedea dispozitivul cu lsusb în Linux sau în Device Manager în Windows. În ultimul caz vi se va cere un driver pentru el, pe care îl găsiţi în win/driver - daţi install from a specific location sau ceva de genul. Este vorba de un driver generic pentru biblioteca libusb care permite realizarea de drivere user-mode pentru dispozitive usb. E posibil să vă ceară driverul dacă schimbaţi portul USB pe care conectaţi placa sau principial când are chef,

deci să-l aveţi la îndemână.

În directorul software găsiţi programul ce trebuie rulat pe PC pentru a controla bootloader-ul. Executabilul este compilat pentru Linux, găsiţi unul pentru Windows în directorul win.

Rulaţi avrusbboot -r pentru a rula pe microcontroler aplicaţia memorată în flash (a părăsi bootloaderul).

(poate aţi vrea să copiaţi avrusbboot în /usr/bin sau windows\ ca să îl puteţi accesa de oriunde).

Rulaţi avrusbboot filename.hex pentru a scrie în microcontroler programul conţinut de filename.hex şi a-l lansa în execuţie.

Există un flash.hex cu care puteţi testa că vă merge (emulează o tastatură USB).

Nu este nevoie să specificaţi tipul microcontrolerului ca la avrdude, trebuie doar codul să fie compilat pe arhitectura corespunzătoare.

Deoarece accesează un device hardware direct, avrusbboot are nevoie de privilegii root pe Linux (rulaţi cu sudo).

Exerciţiu: modificaţi bootloader-ul astfel încât dacă nu a primit comandă de la PC (prin avrusbboot) în n secunde pentru a executa aplicaţia sau a scrie flash-ul, să execute aplicaţia direct.

Atenţie: o dată rulată aplicaţia pe microcontroler, avrusbboot va spune că nu mai găseşte dispozitivul. Acest lucru e normal, deoarece bootloader-ul şi-a terminat execuţia şi structura de date ce descria dispozitivul a dispărut. Pentru a scrie o nouă variantă de cod în microcontroler, acesta trebuie resetat (cu şurubelniţa între reset şi masă, cu un jumper, cu un buton, din softul aplicaţie, power cycle etc.) pentru a reactiva bootloader-ul.

Dezvoltare

Probabil veţi vrea să compilaţi bootloader-ul şi programul de control, pentru a le customiza sau dacă n-aveţi încredere în executabilele mele :)

Pentru a compila bootloader-ul (directorul firmware) pe Linux aveţi nevoie de avr-gcc (pachetul s-ar putea numi gcc-avr), binutils-avr şi avr-libc. Pentru a-l scrie în flash puteţi folosi avrdude şi un cablu pe portul paralel sau pe serial cu diode zener (pe un calculator care are aşa ceva). Pinoutul cablului (care pin de pe mufa de port serial / paralel merge la care pin de pe microcontroler) se poate alege şi deduce din /etc/avrdude.conf.

Pentru a compila programul de control (avrusbboot) e nevoie de g++ şi libusb-dev. Dacă vrea cineva să-l ia şi să scape de clase e binevenit.

Pentru a compila bootloader-ul pe Windows puteţi instala WinAVR care vine cu tot ce trebuie (inclusiv make), însă ultima dată când l-am folosit nu se înţelegea prea bine cu Cygwin.

Pentru a compila programul de control aveţi nevoie de headere şi lib-uri de la libusb-win32.sourceforge.net. Tot acolo găsiţi şi driverul. Eu l-am cross-compilat în Linux.

Download

avrusbboot-ret.2008-03-4.tar.gz - corecţie eroare signed/unsigned char în software

Încheiere

Alt bootloader USB găsiţi aici. Foloseşte HID (human interface device), o clasă de dispozitive USB pentru care există un driver generic în Windows. Astfel nu mai trebuie instalat driverul libusb, programul de control folosind API-ul HID. Puteţi să-l încercaţi, mie nu mi-a mers.

Spor!