# Vdl191v fejlesztői dokumentáció

## Alapok

Az alapvető elképzelésem a projekt mögött az volt, hogy minden embernek, aki találkozik a Voltcraft DL-191V feszültségmérővel, tudjak biztosítani egy ingyenes, nyílt forráskódú megoldást ennek az eszköznek a kezeléséhez linux alatt. Mivel az eredeti, gyári program egy rakás szépség, én személy szerint Windows alatt is inkább a saját megoldásomat használnám, de erre majd kitérek később. Mivel arról (természetesen) semmilyen dokumentáció nem létezik, hogy pontosan az eszköz milyen kommunikációt folytat a különbötő műveletekhez, ezért ennek a megfejtése is rám hárult (kicsit egyiptomi hieroglifa-fejtőnek éreztem magam). Ehhez Wiresharkot használtam.

## Az eszköz működése

Az adatgyűjtő működése viszonylag egyszerű, de nem feltétlenül illeszkedik minden pontja az emberi logikához. A gyári program segítségével fel kell konfigurálnunk az eszközt minden mérés kezdete előtt, ahol különböző dolgokat állíthatunk be, ilyen például a mintavétel periódusa, vagy hogy milyen gyorsan villogjon a rajta lévő LED (igen, ez egy nagyon fontos beállítás). Mivel az eszköz egy nem újratölthető, speciális kialakítású, 3,6V-os elemről működik, ezért az energiatakarékossági törekvéseinek érthetőnek kellene lenniük, de mégsem azok. A konfiguráció feltöltését követően a műszer készenléti állapotba kerül, amit időnkénti dupla zöld villanással jelez felénk. Választhatjuk azt a lehetőséget is, hogy a konfiguráció kiírása utána mérés egyből elinduljon, ez akkor hasznos, ha az eszközt nem akarjuk eltávolítani a számítógépből, azt közvetlenül onnan akarjuk működtetni. Egyébként a mérés az eszközön (elég idióta helyen) lévő piros gomb megnyomásával indul. Ezt követően a feszültségmérő el is kezd adatokat rögzíteni. Fontos megjegyezni, hogy kizárólag 0 és 30V között képes mérni, ez azt jelenti, hogy nem cserélhetjük fel a polaritást sem, tehát erre oda kell figyelni. Szabványos módon, a piros drót megy a pozitívba, a fekete a negatívba, de ez külön fel is van tüntetve az eszközön. A mérés két módon állítható le: Az adatok letöltésével (a gyári programmal, egyébként a mérés minden esetben a setup lekérő bináris jellel áll le), vagy ha az adatok száma eléri a mérés elején felkonfigurált maximális adatszámot, ami a 64kB-os memóriára való tekintettel maximum 32000 lehet – az adatokat short integerekben tárolja, a számítástechnikában használt szabványos módon (ez majd a programozást később nagyban megkönnyíti). Itt lényeges az a tény, hogy utóbbi esetben, bár a mérés leáll, az eszköz nem kapcsol ki, tehát továbbra is értékes energiát fog felhasználni a „nem megújuló” energiaforrásából. Ezért nincs sok értelme 32000 helyett bármilyen más számra állítani a mérési adathatárt. Az adatok letöltését követően a gyári program eltárolja azokat, és nagyon idióta módon onnan le lehet ezeket kérni, kiíratni, primitív grafikont rajzoltatni belőle, vagy exportálni. Egyébként ez a program a legutolsó frissítését 2012-ben kapta, ami azért gáz, mert ennek létezik egy fizetős verziója is (amit feltételezésem szerint ugyan úgy nem fejlesztenek, csak elkérik érte a pénzt).

## A kommunkiáció megfejtése

Mint azt már a bevezetésben említettem, ehhez Wireshark programot használtam. Ez gyakorlatilag minden USB-n kiküldött (vagy bárhol máshol, de most ez a lényeg) vagy fogadott adatcsomagot monitorozni képes. Innen már „csak” annyit kellett tennem, hogy a gyári program segítségével szabványos kommunikációt folytatok az eszközzel, és figyelem az átküldött adatokat. Az első két csomag minden esetben egy USB_CONTROL_TRANSFER volt, ahol a gépem és az eszköz technikai jellegű információkat cseréltek. Ezek nem tudom hogy pontosan mit jelentenek, valahol az USB kommunkiáció protokolljáról állapodhatnak meg benne gondolom, de nem is kell hogy tudjam, csak egyszerűen lemásoltam a windowson átküldött beállításokat linuxon, és működött. Ezt követően a kommunikáció lezárásáig kizárólag BULK_TRANSFER kommunikáció történik a két eszköz között.

Mi közöljük az eszközzel hogy mit akarunk, egy 3 byte-os header-ben. 16-os számrendszerben ezek a következő értékek lehetnek:

| | |
|---|---|
| Konfigurációs adatok lekérése | 00 10 01 |
| Eszköz memórájának olvasása | 00 x y |
| Konfigurációs adatok küldése | 01 40 00 |

Majd ezekre az eszköz válaszol, általában szintén egy 3 byte-os headerben. Memóriaolvasáskor ez 02 00 00 minden esetben, konfigurációs adatok lekérésénél 02 (legutóbbi mérésnél rögzített adatok száma short-ban). Konfiguráció írásakor azonban csak 1 byte-ot kapunk vissza, ami ff ha jó, ha nem jó akkor meg gondolom valami hibakódot, de ez elég zavaros. A műszer memóriája 4kB-os blokkokra van osztva, összesen 16-ra. Ezek számozása 0-val kezdődik. Memória olvasáskor x értéke adja meg, hogy hanyadik blokkból akarunk olvasni az eszközön, y pedig, hogy hányszor 64 byte adatot kérünk az adott blokkból (maximum nyilván a teljes blokkot kérhetjük, ami 64*64 byte). Ezért van az, hogy a konfiguráció lekérdezésekor 00 10 01 a header: ekkor ugyanis a 16. blokk első 64 elemét kérjük le, ami, úgy néz ki, az a memóriaterület, ahol a feszültségmérő a konfigurációs beállításokat tárolja.

Ezt követően a konfigurációs beállításokat kellett megfejtenem. Ehhez változtattam a gyári programban a beállításokat, és Wireshark-kal figyeltem, hogy pontosan ez melyik bitet/byteot változtatja. A konfig egy 64 byte-os adatsorozat, a következő módon, 16-os számrendszerben:

| Byteok sorszáma (darab) | Funkció | Kód/kódolás |
|---|---|---|
| 0-3 (4) | Indítási jel | ce 00 00 00 |
| 4-7 (4) | Max. mérésszám | Integer |
| 8-11 (4) | Rögzített adatok száma (csak lekérdezésnél, amúgy 0) | Integer |
| 12-15 (4) | Rögzítés periódusa(s)[0=400Hz] | Integer |
| 16-19 (4) | Rögzítés kezdetének éve | Integer |
| 20-23 (4) | Alacsony feszültég riasztás | Ismeretlen |
| 24-27 (4) | Magas feszültség riasztás | Ismeretlen |
| 28 (1) | Rögzítés kezdetének hónapja | Char |
| 29 (1) | Rögzítés kezdetének napja | Char |
| 30 (1) | Rögzítés kezdetének órája | Char |
| 31 (1) | Rögzítés kezdetének perce | Char |
| 32 (1) | Rögzítés kezdetének másodperce | Char |
| 33 (1) | Ismeretlen (placeholder?) | 00 |
| 34 (1) | Q (bővebben lent) | Bitenkénti |
| 35-50 (16) | Konfiguráció neve | Char[] |
| 51 (1) | Mérés indulása (gomb/instant) | Bitenkénti |
| 52-55 (4) | Alacsony feszültég riasztás | Ismeretlen |
| 56-59 (4) | Magas feszültség riasztás | Ismeretlen |
| 60-63 (4) | Záró jel | ce 00 00 00 |

(És igen, ez az a táblázat, amit ha a gyártó közzétenne, akkor lényegesen kevesebb idő lett volna megírni ezt a programot). Itt azonban néhány dolog további magyarázatra szorul. A kódolásoknál az integerek szabványos, 4 byteos integerek pont úgy letárolva, ahogy azt mondjuk a C csinálná. A char-ok maguktól értetődően 1 byteos adatok. A konfiguráció neve egy 16 karakteres string, aminek nincs sok értelme, ezért a programomban ez konstans „Brutus”. A Q byte egy érdekes dolog (ezt csak én neveztem el így). Ott több dolog van bitenként kódolva. Az első 5 bit (jobbról balra) a led villogásának periódusát mondja meg másodpercben, az utolsó bit pedig azt, hogy legyen-e riasztás vagy sem. Mivel a riasztás értékeinek kódolását nem sikerült megfejtenem, ezért ez a bit egyelőre konstans 0. A kettő közötti 2 bit pontosan nem tudom hogy mit csinál, valamilyen frekvencia-módot állíthat, de ha rosszul van beállítva egy adott adatgyűjtési időperiódusra, akkor a mérés valamilyen módon sikertelen lesz, erre majd később visszatérek még. A másik bitenkénti kódolással rendelkező byte az 51-es, ami a mérés elindulását állítja, hogy intant elinduljon-e a konfig átküldése után, vagy csak gombnyomásra. Ennek az első 2 bitje 01 ha gombnyomásra indul, és 10 ha instant. A többi bit mindig 0, kivéve 400 Hz-es frekvenciánál, mert ott valamiért az 5. bit 1-es lesz.

Ezzel el is érkeztünk ahhoz a ponthoz, hogy ezeknek az értékeknek a megfelelő beállítását kicsit részletezzem. Sajnos nem lehet bármilyen mérési periódusidőt beállítani, mert nem minden fog működni. A gyári programban ezt úgy oldják meg, hogy egy menüből lehet ide számokat kiválasztani, és nem lehet kézzel megadni – ezek szerint okkal. A gyárilag megadott értékeken kívül néhány működik, néhány nem. Probléma az is, hogy ezekhez nem tudom, hogy pontosan hogyan kéne beállítani az 51. byte 6. és 7. bitjét. Kézzel megadtam pár értéket a programban amik működőnek tűnnek, de közel sem biztos. Ha egy érték nem működik, a következő tünetei lehetnek:
- A mérés nem történik meg egyáltalán, a feszültségmérő adatletöltéskor egy darab nullát ad vissza adatként
- A mérés megtörténik, de minden adat 0 lesz
- A mérés megtörténik, és valós adatokat kapunk vissza, de az időperiódus tájékoztatás nélkül visszaugrik az egyik gyári értékre, pl. 6 másodpercet adunk meg, de mérés 5 másodpercenként történik.

Ez azért nagy probléma, mert az eszköz nem tárolja el minden egyes adathoz a rögzítés időpontját, csak a rögzítés kezdetét ismerjük. Így az adatok idejének meghatározásához szükséges az, hogy pontosan tudjuk, hogy azok milyen gyakran keletkeztek. Ha ezt rosszul ismerjük, akkor az időpontok nem fognak stimmelni. Ezért (is) veszélyes ezzel a beállítással játszani. A programom alaphelyzetben ezeket a rossz beállításokat szűri, és visszadob mindent, ami nem gyári érték, de ezt a -f (force) kapcsolóval felül lehet bírálni.

Most nézzük az adatok letöltésének módját. Az szköz memóriájának bármely részét elméletileg bármikor kiolvashatjuk a 00 x y header elküldésével, amit már fentebb megmagyaráztam. Az eszköz a konfiguráció lekérésével elküldi nekünk, hogy a legutóbbi méréssel (idáig, hiszen általában még fut a mérés amikor kiolvassuk az adatokat) hány adat keletkezett. Ennyit kell leolvasnunk a memóriából. Elméletileg nincs akadálya annak, hogy többet, vagy akár bármelyik másik memóriaterületet is olvassuk, ekkor memóriaszemetet fogunk visszakapni, az előző mérések eredményeivel. Fontos megjegyezni, hogy egyszerre nem tudjuk lekérni az egész memóriáját, hiszen egy header elküldésével csak 1 blokkot tudunk maximálasan olvasni. Ezért ha mondjuk 16000 mérési adatunk van, akkor legalább 4-szer ki kell küldenünk neki a headert, úgy hogy 00 00 40, majd 00 01 40, stb. Az eszköz egy 02 00 00-es „OK, küldöm” header után maximum 512 byteos csomagokban elkezdi küldeni nekünk az adatokat. Ezeket egyesével végig ki kell olvasni, majd egy üres csomaggal le kell okézni, különben nem kapunk tőle semmit. Mivel a minimum adat, amit olvashatunk, az 64 byte, ezért ebben valószínűleg lesz memóriaszemét. Ennek a mennyiségét a konfigurációs beállításokban lekért mérési adatszámmal állapíthatjuk meg. Hogy melyik headerben hányszor 64 byte adatot kérünk tőle, azt is ez dönti el, értelemszerűen. A konfigurációs adatok megfejtéséhez sokat segített ez a projekt, ami egy hasonló eszközt kódol le C-ben: https://github.com/mildis/vdl120 (de sajnos a dolgok többségét így is wiresharkkal kellett kinyomoznom, kiindulópontnak viszont jó volt)

## A program

Három oldalt követően végre elérkeztünk oda, hogy beszélhessünk a program szerkezeti felépítéséről, ami valószínűleg egyszerűbb lesz, mint maga az eszköz titkainak megfejtése. A program három modulból áll, ezekhez 2+1 header tartozik. A main végzi a felhasználói bemenet ellenőrzését, a logger a kiolvasott adatok eltárolásáért felel, és a lényegi részt, az eszközkezelést a voltcraft osztály végzi. A +1 header a datastructs.h, ami adatstruktúrákat tartalmaz, elsősorban a konfigurációs beállítások adatstruktúráját.

**Voltcraft.cpp:**

A constructor a main-től átvett vid és pid (VendorID, ProductID, ezek azonosítják az eszközt) letárolását végzi csak. Miután léterjött az objektum, az eszközt a configure függvénnyel konfigurálhatjuk egy méréshez. Ehhez libusb-t használok. A libusb_device_handle gyakorlatilag kezeli az eszközt, ezen keresztül tudok neki adatokat küldeni. Ahhoz, hogy rácsatlakokzhassak az eszközhöz, előbb megvizsgálom, hogy létezik-e ilyen eszköz egy for ciklus segítségével. Az elérhető eszközlistát a libusb_get_device list függvénnyel teszem. Ezt követően a vid és pid alapján megpróbálok rácsatlakozni az eszközre. Ehhez általában root jog szükséges, ezért ha ez nem sikerül, feltétlezzük, hogy valaki elfelejtett egy sudo-t. Majd megkísérlejük lecsatlakoztatni a kernel által az eszközhöz rendelt drivert, hiszen mi magunk akarunk vele alacsony szintű kommunikációt folytatni. Sikeres művelet esetén claimeljük az interface-t. Ezt követően kiküldjük neki a control adatokat, ezt csak szimplán a windowsos verzióról másoltam. Itt a kommunikációnk az eszközzel már él. Létrehozzuk a konfigurációs beállításokat a függvény paramétereiből (itt nem ellenőrizzük, hogy ezek jók-e, itt hagyunk minden hülyeséget felkonfigurálni, azt előtte kell megtenni). Beállítjuk a headereket is. Innen pedig már csak elküldjük az eszköznek ezeket az adatokat. Ennek szintaxisa: libusb_bulk_transfer(dev_handle,endpoint,data,datacount,&kiírt adatok,timeout). A műszer endpointjait lsusb paranccsal, vagy wiresharkkal tudjuk kideríteni, amik itt írásra 2-nek (0x02), olvasásra 130-nak (0x82) adódtak. A különböző struktúrákat a github projektjében mildis nekem elsőre nagyon furcsa módon char*-ra castolta, ami második ránézésre megtetszett nekem is (mert ez nagyságrendekkel egyszerűbb, mint bárhogy máshogy), így én is ezt használtam. Ez alatt az elmélet az, hogy a libusb transfer függvénye csak char* formátumú pointerrel tud dolgozni, ezért ha a struktúrákban a változók sorrendje szigorúan adott, akkor az unsigned char*-ra való cast úgy fog a struktúrára mutatni, hogy annak bytekódjának sorrendjét megőrzi, így az eszköz a megfelelő sorrendben fogja a megfelelő adatokat kapni. Ez egy gyors és működő módszer (aztán nem tudom, lehet hogy ezt standard mindenhol így csinálják, nekem ez az első kódom amit látok valami ilyesmivel). Miután ezeket a csomagokat átküldtük, be is zárhatjuk a kapcsolatot.

A download függvény először megcsinálja ugyan azokat a dolgokat, amiket a configure – a forrásban részleteztem, hogy ezeket miért nem tudtam külön szedni, és miért kell kétszer bemásolni tök ugyan azt a kódrészletet (libusb-ben van valahol egy segment faul hiba ilyenkor, nem teljesen értem hogy miért, valószínűleg valami dinamikus memória felszabadul a függvény végén aminek nem kéne (lehet hogy meg lehetne oldani ha dinamikus lenne a dev_handle meg a coontext), nem sokat foglalkoztam ennek a kinyomozásával). Ezt követően lekérjük a megfelelő header elküldésével a konfigurációs beállításokat, ezzel megállítva az esetlegesen még mindig működő mérést. Majd egy halom for ciklusban kiolvassuk a megfelelő memóriaterülteket a feszültségmérőből. Fontos, hogy az eszköznek rendszeresen válaszolnunk kell, hogy megkaptuk az adatokat egy üres csomaggal, különben nem áll velünk többé szóba. Egy dinamikusan lefoglalt tömbben visszaadjuk paraméterként a mérési adatokat, és visszatérési értékként pedig azt, hogy hány adatunk van. Minden mérési adat 2 byte, ezért unsigned short int-be teszem őket bele.

**Main.cpp:**

Ebben a main függvényen kívül 4 másik függvény található: argmis, argbad, isNum, és a validateConf. Az argmis egy hibaüzenet, ami a hiányzó paraméter miatt reklamál. Az argbad ugyan ez, csak rossz paraméter miatt. Az isNum eldönti egy stringről, hogy csak pozitív számot tartalmaz-e. A validateConf lecsekkolja a az eszköznek való elküldés előtt, hogy jók-e az adatok (ezt nem részletezném, csak egy rakás if). Minden felhasználói bement kezelést a main-be raktam bele, mert különben nem nagyon csinálna semmit, és végül is miért ne. Miután a megfelelő kapcsolókból származó információkat kiolvastuk, a megadott parancstól függően letöltjük az adatokat a megadott paraméterek szerint, vagy kiírjuk a konfigurációt. Ehhez a voltcraft osztályt használom.

**Logger.cpp:**

Ez az osztály felel a letöltött adatok fájlban való tárolásáért. A constructor az ehhez szükséges adatokat tárolja (pl. kell-e fejléc, kellenek-e az idők, stb). A fejlécben a # a gnuplot miatt kell, így egyszerűbb belőle grafikont rajzolni. Fájlformátumnak .dsv-t választottam, ez a Delimiter-separated values rövidítése, vagyis tabulátorokkal tagolt szöveg. A perzisztens tárolást itt egy << operátor túlterheléssel valósítottam meg. A main-ben tehát csak létre kell hozni egy logger objektumot, és belenyomni valamilyen filestreambe <←el. Itt kicsit erőltetett módon, de kitörlöm a destructorban a voltcraft.cpp-ben lefoglalt dinamikus memóriát (igen, ez így elég csúnya, de szeretném a feladat formai követeléneyeit teljesíteni, és ez legalább egy értelmes destructor)

A felhasználói kézikönyvnek az angol nyelvű README.md-t szántam.